Verification of Hierarchical Artifact Systems

Data-driven workflows, of which IBM's Business Artifacts are a prime exponent, have been successfully deployed in practice, adopted in industrial standards, and have spawned a rich body of research in academia, focused primarily on static analysis. The present work represents a significant advance on the problem of artifact verification, by considering a much richer and more realistic model than in previous work, incorporating core elements of IBM's successful Guard-Stage-Milestone model. In particular, the model features task hierarchy, concurrency, and richer artifact data. It also allows database key and foreign key dependencies, as well as arithmetic constraints. The results show decidability of verification and establish its complexity, making use of novel techniques including a hierarchy of Vector Addition Systems and a variant of quantifier elimination tailored to our context.

CCS Concepts: • **Theory of computation → Verification by model checking**; **Logic and databases**; • **Applied computing → Business process management**;

ACM Reference Format:

. 2018. Verification of Hierarchical Artifact Systems. 1, 1 (May 2018), 70 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The past decade has witnessed the evolution of workflow specification frameworks from the traditional process-centric approach towards data-awareness. Process-centric formalisms focus on control flow while under-specifying the underlying data and its manipulations by the process tasks, often abstracting them away completely. In contrast, data-aware formalisms treat data as first-class citizens. A notable exponent of this class is IBM's *business artifact model* pioneered in [52], successfully deployed in practice [12, 14, 20, 25, 67] and adopted in industrial standards. Business artifacts have also spawned a rich body of research in academia, dealing with issues ranging from formal semantics to static analysis (see related work).

In a nutshell, business artifacts (or simply "artifacts") model key business-relevant entities, which are updated by a set of services that implement business process tasks, specified declaratively by pre-and-post conditions. A collection of artifacts and services is called an *artifact system*. IBM has developed several variants of artifacts, of which the most recent is Guard-Stage-Milestone (GSM) [22, 40]. The GSM approach provides rich structuring mechanisms for services, including parallelism, concurrency and hierarchy, and has been incorporated in the OMG standard for Case Management Model and Notation (CMMN) [15, 47].

Author's address:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2018 Association for Computing Machinery.
XXXX-XXXX/2018/5-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Artifact systems deployed in industrial settings typically specify very complex workflows that are prone to costly bugs, whence the need for verification of critical properties. Over the past few years, an active line of research on the verification of artifact systems has emerged. Rather than relying on general-purpose software verification tools suffering from well-known limitations, the aim of this work, described below, has been to identify practically relevant classes of artifact systems and properties for which *fully automatic* verification is possible. This is an ambitious goal, since artifacts are infinite-state systems due to the presence of unbounded data. Approaches to this problem rely critically on the declarative nature of service specifications, bringing into play a novel marriage of database and computer-aided verification techniques.

In previous work [21, 26], the verification problem was studied for a bare-bones variant of artifact systems, without hierarchy or concurrency, in which each artifact consists of a flat tuple of evolving values and the services are specified by simple pre-and-post conditions on the artifact and database. More precisely, the problem considered was to statically check whether all runs of an artifact system satisfy desirable properties expressed in LTL-FO, an extension of linear-time temporal logic where propositions are interpreted as ∃FO sentences on the database and current artifact tuple. In order to deal with the resulting infinite-state system, in [26], a symbolic approach was developed to allow a reduction to finite-state model checking and yielding a PSPACE verification algorithm for the simplest variant of the model (no database dependencies and uninterpreted data domain). In [21] the approach was extended to allow for database dependencies and numeric data testable by arithmetic constraints. Unfortunately, decidability was obtained subject to a rather complex semantic restriction on the artifact system and property (feedback freedom), and the verification algorithm has non-elementary complexity.

The present work represents a significant advance on the artifact verification problem on several fronts. We consider a much richer and more realistic model, called *Hierarchical Artifact System* (HAS), abstracting core elements of the GSM model. In particular, the model features task hierarchy, concurrency, and richer artifact data (including updatable artifact relations). We consider properties expressed in a novel *hierarchical* temporal logic, HLTL-FO, that is well-suited to the model. Our main results establish the complexity of checking HLTL-FO properties for various classes of HAS, highlighting the impact of various features on verification. The results require qualitatively novel techniques, because the reduction to finite-state model checking used in previous work is no longer possible. Instead, the richer model requires the use of a hierarchy of Vector Addition Systems with States (VASS) [16]. The arithmetic constraints are handled using quantifier elimination techniques, adapted to our setting.

We next describe the model and results in more detail. A HAS consists of a database and a hierarchy (rooted tree) of *tasks*. Each task has associated to it local evolving data consisting of a tuple of artifact variables and an updatable artifact relation. It also has an associated set of *services*. Each application of a service is guarded by a pre-condition on the database and local data and causes an update of the local data, specified by a post condition (constraining the next artifact tuple) and an insertion or retrieval of a tuple from the artifact relation. In addition, a task may invoke a child task with a tuple of parameters, and receive back a result if the child task completes. A run of the artifact system consists of an infinite sequence of transitions obtained by any valid interleaving of concurrently running task services.

In order to express properties of HAS's we introduce a subset of LTL-FO called *hierarchical* LTL-FO (HLTL-FO). Intuitively, an HLTL-FO formula uses as building blocks LTL-FO formulas acting on runs of individual tasks, called local runs, referring only to the database and local data, and can recursively state HLTL-FO properties on runs resulting from calls to children tasks. The

language HLTL-FO closely fits the computational model and is also motivated on technical grounds discussed in the paper. A main justification for adopting HLTL-FO is that LTL-FO (and even LTL) properties are undecidable for HAS's.

Hierarchical artifact systems as sketched above provide powerful extensions to the variants previously studied, each of which immediately leads to undecidability of verification if not carefully controlled. Our main contribution is to put forward a package of restrictions that ensures decidability while capturing a significant subset of the GSM model. This requires a delicate balancing act aiming to limit the dangerous features while retaining their most useful aspects. In contrast to [21], this is achieved without the need for unpleasant semantic constraints such as feedback freedom. The restrictions are discussed in detail in the paper, and shown to be necessary by undecidability results.

The complexity of verification under various restrictions is summarized in Tables 1 (without arithmetic, page 40) and 2 (with arithmetic, page 42). As seen, the complexity ranges from PSPACE to non-elementary for various packages of features. The non-elementary complexity (a tower of exponentials whose height is the depth of the hierarchy) is reached for HAS with cyclic schemas, artifact relations and arithmetic. For acyclic schemas, which include the widely used Star (or Snowflake) schemas [42, 64], the complexity ranges from PSPACE (without arithmetic or artifact relations) to double-exponential space (with both arithmetic and artifact relations). This is a significant improvement over the previous algorithm of [21], which even for acyclic schemas has non-elementary complexity in the presence of arithmetic (a tower of exponentials whose height is the square of the total number of artifact variables in the system). Moreover, we recently implemented a verifier that can handle HAS with acyclic schemas, and has excellent performance, demonstrating the practical potential of our approach. The implementation relies on the techniques developed in the present paper and is described in [2].

The paper is organized as follows. The HAS model is presented in Section 2. We present its syntax and semantics, including a representation of runs as a tree of local task runs, that factors out interleavings of independent concurrent tasks. The temporal logic HLTL-FO is introduced in Section 3, together with a corresponding extension of Büchi automata to trees of local runs. Section 4 justifies the restrictions imposed on the HAS model by showing that lifting any of them leads to undecidability of verification. In Section 5 we prove the decidability of verification for HAS without arithmetic, and establish its complexity. To this end, we develop a symbolic representation of HAS runs and a reduction of model checking to state reachability problems in a set of nested VASS (mirroring the task hierarchy). In Section 6 we show how the verification results can be extended in the presence of arithmetic. Finally, we discuss related work in Section 7 and conclude. The appendix provides more details, proofs and tables of notations.

2 FRAMEWORK

In this section, we present the syntax and semantics of Hierarchical Artifact Systems (HAS's). The formal definitions are illustrated with an intuitive example of the HAS specification of a travel booking business process inspired by Expedia [34]. At a high level, the example captures a process where a customer books flights and/or makes hotel reservations.

2.1 Syntax of HAS

We begin with the underlying database schema. We assume familiarity with the notions of *key* and *foreign key* (e.g., see [56]), as well as first-order formula (FO), existential FO (∃FO), and quantifier-free FO (e.g., see [45]).

Definition 1. *A **database schema*** DB *is a finite set of relation symbols, where each relation R of* DB *has an associated sequence of distinct attributes containing the following:*

- *a key attribute* ID *(present in all relations),*
- *a set of foreign key attributes* $\{F_1, \ldots, F_m\}$*, and*
- *a set of non-key attributes* $\{A_1, \ldots, A_n\}$ *disjoint from* $\{\text{ID}, F_1, \ldots, F_m\}$.

To each foreign key attribute F_i *of* R *is associated a relation* R_i *of* DB *and the inclusion dependency* $R[F_i] \subseteq R_i[\text{ID}]$ *(stating that the projection of* R *on* F_i *is included in the projection of* R_i *on* ID*). It is said that* F_i *references* R_i*. We denote by* $attr(R)$ *the set of attributes of* R*.*

The domain $Dom(A)$ of each attribute A depends on its type. The domain of all non-key attributes is numeric, specifically $\mathbb{R}$. The domain of each key attribute is a countable infinite domain disjoint from $\mathbb{R}$. For distinct relations R and R', $Dom(R.ID) \cap Dom(R'.ID) = \emptyset$. The domain of a foreign key attribute F referencing R is $Dom(R.ID)$. We denote by $DOM_{id} = \cup_{R \in \text{DB}} Dom(R.ID)$. Intuitively, in such a database schema, each tuple is an object with a *globally* unique id. This id does not appear anywhere else in the database except in foreign keys referencing it. An *instance* of a database schema DB is a mapping D associating to each relation symbol R a finite relation $D(R)$ of the same arity as R, whose tuples provide, for each attribute, a value from its domain. In addition, D satisfies all key and inclusion dependencies associated with the keys and foreign keys of the schema. The active domain of D, denoted $\texttt{adom}(D)$, consists of all elements of D (id's and reals). A database schema DB is *acyclic* if there are no cycles in the references induced by foreign keys. More precisely, consider the labeled graph FK whose nodes are the relations of the schema and in which there is an edge from R_i to R_j labeled with F if R_i has a foreign key attribute F referencing R_j. The schema DB is *acyclic* [1] if the graph FK is acyclic, and it is *linearly-cyclic* if each relation R is contained in at most one simple cycle. A main reason for considering these special schemas is that they lead to significantly improved complexity of verification. The most restricted, acyclic schemas, still capture Star and Snowflake schemas [42, 64], widely used in storing business process data. As mentioned earlier, the verifier described in [2] exhibits excellent perfomance on HAS with acyclic schemas.

Example 2. *The HAS of the travel booking business process has the following database schema:*

- `FLIGHTS(ID, price, comp_hotel_id)`
  `HOTELS(ID, unit_price, discount_price)`

In the schema, the `ID`*'s are key attributes,* `price`*,* `unit_price`*,* `discount_price` *are non-key attributes, and* `comp_hotel_id` *is a foreign key attribute satisfying the inclusion dependency:*

$$\texttt{FLIGHTS}[\textit{comp\_hotel\_id}] \subseteq \texttt{HOTELS}[\texttt{ID}].$$

Intuitively, each flight stored in the `FLIGHTS` *table has a hotel compatible for discount. If a flight is purchased together with a compatible hotel reservation, a discount is applied on the hotel reservation. Otherwise, the full price needs to be paid. The schema is acyclic, since* $\texttt{FLIGHTS}[\textit{comp\_hotel\_id}] \subseteq \texttt{HOTELS}[\texttt{ID}]$ *is the only inclusion dependency in the schema.*

The assumption that the ID of each relation is a single attribute is made for simplicity, and multiple-attribute IDs can be easily handled. The fact that the domain of all non-key attributes is

[1] Here a cycle is a sequence of relations $\{R_i\}_{1 \le i \le k}$ where $k \ge 2$, $R_1 = R_k$ and there is a foreign key reference from R_{i-1} to R_i for every $i > 1$.

numeric is also harmless. Indeed, a uninterpreted domain on which only equality can be used can be easily simulated. Note that the keys and foreign keys used on our schemas are special cases of the dependencies used in [21]. The limitation to keys and foreign keys is one of the factors leading to improved complexity of verification and still captures most schemas of practical interest.

We next proceed with the definition of tasks and services, described informally in the introduction. The definition imposes various restrictions needed for decidability of verification. These are discussed and motivated in Section 4.

Similarly to the database schema, we consider two infinite, disjoint sets VAR_{id} of ID variables and $VAR_{\mathbb{R}}$ of numeric variables. We associate to each variable x its *domain* $Dom(x)$. If $x \in VAR_{id}$, then $Dom(x) = \{\texttt{null}\} \cup DOM_{id}$, where $\texttt{null} \notin DOM_{id} \cup \mathbb{R}$ (`null` plays a special role that will become clear shortly). If $x \in VAR_{\mathbb{R}}$, then $Dom(x) = \mathbb{R}$. An *artifact variable* is a variable in $VAR_{id} \cup VAR_{\mathbb{R}}$. If $\bar{x}$ is a sequence of artifact variables, a *valuation* of $\bar{x}$ is a mapping ν associating to each variable in $\bar{x}$ an element of its domain $Dom(x)$.

Definition 3. *A **task schema** over database schema* DB *is a triple* $T = \langle \bar{x}^T, S^T, \bar{s}^T, \bar{x}^T_{\mathsf{in}}, \bar{x}^T_{\mathsf{out}} \rangle$ *where* $\bar{x}^T$ *is a sequence of distinct artifact variables,* S^T *is a relation symbol not in* DB *with associated arity* k, $\bar{s}^T$ *is a sequence of* k *distinct ID variables in* $\bar{x}^T$ *and* $\bar{x}^T_{\mathsf{in}}$ *and* $\bar{x}^T_{\mathsf{out}}$ *are subsequences of* $\bar{x}^T$ *called the input and output variables of* T.

We denote by $\bar{x}^T_{id} = \bar{x}^T \cap VAR_{id}$ and $\bar{x}^T_{\mathbb{R}} = \bar{x}^T \cap VAR_{\mathbb{R}}$. We refer to S^T as the *artifact relation* or *set* of T. Intuitively, an artifact relation is an updatable set where a task can insert/retrieve tuples. As we shall see, tuples of artifact relations are restricted to contain values from DOM_{id}. The data stored in the artifact variables and relations together represent the current state of a task.

Example 4. ***ManageTrips*** *is a task in the travel booking artifact system. This task models the process whereby the customer creates, stores, and retrieves candidate trips. A trip consists of a flight and/or hotel reservation. The customer can also choose one of the candidate trips and finalize the booking. The task has the following artifact variables:*

- *ID variables:* `flight_id`, `hotel_id`,
- *numeric variables:* `amount_paid`, `status`.

It also has an artifact relation `TRIPS` *storing candidate trips* (`flight_id`, `hotel_id`). ***ManageTrips*** *has no input/output variables.*

Definition 5. *An **artifact schema** is a tuple* $\mathcal{A} = \langle \mathcal{H}, \text{DB} \rangle$ *where* DB *is a database schema and* $\mathcal{H}$ *is a rooted tree of task schemas over* DB *with pairwise disjoint sets of artifact variables*[2] *and distinct artifact relation symbols.*

The rooted tree $\mathcal{H}$ defines the *task hierarchy*. Suppose the set of tasks is $\{T_1, \ldots, T_k\}$. For uniformity, we always take task T_1 to be the root of $\mathcal{H}$. We denote by $\preceq_{\mathcal{H}}$ (or simply $\preceq$ when $\mathcal{H}$ is understood) the partial order on $\{T_1, \ldots, T_k\}$ induced by $\mathcal{H}$ (with T_1 the minimum). For a node T of $\mathcal{H}$, we denote by *tree(T)* the subtree of $\mathcal{H}$ rooted at T, *child*(T) the set of children of T (also called *subtasks* of T), *desc*(T) the set of descendants of T (excluding T). Finally, *desc*$^*(T)$ denotes *desc*$(T) \cup \{T\}$. We denote by $\mathcal{S}_{\mathcal{H}}$ (or simply $\mathcal{S}$ when $\mathcal{H}$ is understood) the relational schema $\{S^{T_i} \mid 1 \leq i \leq k\}$. An instance of $\mathcal{S}$ is a mapping associating to each $S^{T_i} \in \mathcal{S}$ a finite relation over DOM_{id} of the same arity.

Example 6. *The travel booking artifact system has the following 4 tasks:* T_1*:**ManageTrips***,* T_2*:**AddHotel***,* T_3*:**AddFlight*** *and* T_4*:**BookTrip***, which form the hierarchy represented in Fig. 1.*

[2]In examples we sometimes use for convenience the same artifact variable names in several tasks, with the understanding that they represent distinct variables.

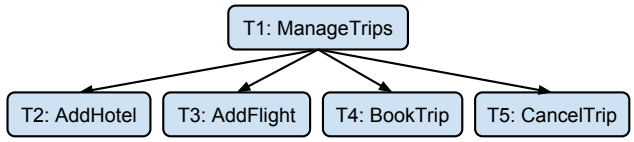

Fig. 1. Tasks hierarchy.

The process implemented by the above tasks can be described informally as follows. At the root task ***ManageTrips****, the customer can add a flight and/or hotel to the trip by calling the* ***AddHotel*** *or the* ***AddFlight*** *tasks. The customer can also store candidate trips in the artifact relation* TRIPS *and retrieve previously stored trips. After the customer has made a decision, the* ***BookTrip*** *task is called to book the trip and the payment is processed. After the payment, the customer can decide to cancel the flight and/or the hotel reservation using the* ***CancelTrip*** *task and receive a refund.*

Definition 7. *An* ***instance*** *of an artifact schema* $\mathcal{A} = \langle \mathcal{H}, \mathrm{DB} \rangle$ *is a tuple* $\bar{I} = \langle \bar{\nu}, stg, D, \bar{S} \rangle$ *where* D *is an instance of* DB, $\bar{S}$ *an instance of* $\boldsymbol{\mathcal{S}}$, $\bar{\nu}$ *a valuation of* $\bigcup_{i=1}^{k} \bar{x}^{T_i}$, *and* stg *(standing for "stage") a mapping of* $\{T_1, \ldots, T_k\}$ *to* $\{\texttt{init}, \texttt{active}, \texttt{closed}\}$.

The stage $stg(T_i)$ of a task T_i has the following intuitive meaning in the context of a run of its parent: `init` indicates that T_i is inactive and available to be called, `active` says that T_i has been called and has not yet returned its answer, and `closed` indicates that T_i has returned its answer. As we shall see, T_i cannot be called while its stage is `closed`, but the model provides a way to reset the stage to `init`. Thus, T_i can be called multiple times during a run of its parent. However, only one instance of T_i can be active at any given time.

Example 8. *Fig. 2 shows an example of an instance of the travel booking business process specified in HAS. The only active task is* ***ManageTrip****.*

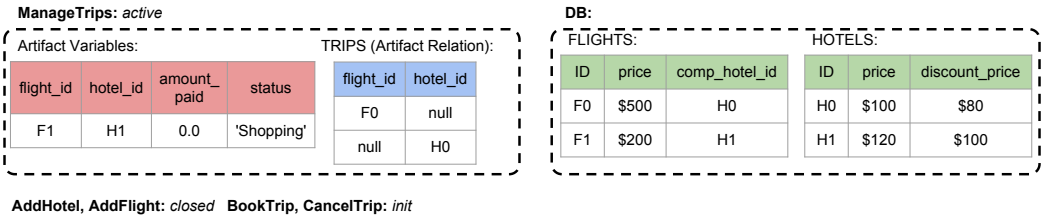

Fig. 2. An instance of the travel booking schema.

We proceed with the definition of *conditions*, used to specify task services. We denote by $\mathcal{C}$ an infinite set of relation symbols, each of which has a fixed interpretation as the set of real solutions of a finite set of polynomial inequalities with integer coefficients. By slight abuse, we sometimes use the same notation for a relation symbol in $\mathcal{C}$ and its fixed interpretation. For a given artifact schema $\mathcal{A} = \langle \mathcal{H}, \mathrm{DB} \rangle$ and a sequence $\bar{x}$ of variables, a *condition* on $\bar{x}$ is a quantifier-free FO formula over $\mathrm{DB} \cup \mathcal{C} \cup \{=\}$ whose variables are included in $\bar{x}$. The special constant `null` can be used in equalities with ID variables. For each atom $R(x, y_1, \ldots, y_m, z_1, \ldots, z_n)$ of relation $R(ID, A_1, \ldots, A_m, F_1, \ldots, F_n) \in \mathrm{DB}$, $\{x, z_1, \ldots, z_n\} \subseteq VAR_{id}$ and $\{y_1, \ldots, y_m\} \subseteq VAR_{\mathbb{R}}$. Atoms over $\mathcal{C}$ use only numeric variables. If α is a condition on $\bar{x}$, D is an instance of DB and ν a valuation of $\bar{x}$, we denote by $D \cup \mathcal{C} \models \alpha(\nu)$ the fact that $D \cup \mathcal{C}$ satisfies α with valuation ν, with standard semantics. For an atom $R(\bar{y})$ in α where $R \in \mathrm{DB}$ and $\bar{y} \subseteq \bar{x}$, if $\nu(y) = \texttt{null}$ for any $y \in \bar{y}$, then $R(\bar{y})$ is false. As will become apparent, although conditions used in HAS are quantifier-free, $\exists$FO conditions can be simulated by adding variables to $\bar{x}^T$, so we use them as shorthand whenever convenient.

Example 9. *The following ∃FO formula indicates that the customer in the travel booking process has chosen a pair of compatible flight and hotel and paid the discounted amount:*

$$\exists q \exists p_1 \exists p_2 \texttt{FLIGHTS}(\texttt{flight\_id}, q, \texttt{hotel\_id}) \wedge \texttt{HOTELS}(\texttt{hotel\_id}, p_1, p_2) \wedge \texttt{amount\_paid} = q + p_2.$$

We next define services of tasks. We start with internal services, which update the artifact variables and artifact relation of the task.

Definition 10. *Let $T = \langle \bar{x}^T, S^T, \bar{s}^T, \bar{x}^T_{\text{in}}, \bar{x}^T_{\text{out}} \rangle$ be a task schema of an artifact schema $\mathcal{A}$. An **internal service** σ of T is a tuple $\langle \pi, \psi, \delta \rangle$ where:*

- *π and ψ, called* pre-condition *and* post-condition*, respectively, are conditions over $\bar{x}^T$*
- *$\delta \subseteq \{+S^T(\bar{s}^T), -S^T(\bar{s}^T)\}$ is a set of* artifact relation updates*; $+S^T(\bar{s}^T)$ and $-S^T(\bar{s}^T)$ are called an **insertion** and **retrieval** of $\bar{s}^T$, respectively.*

Intuitively, an internal service of T can be called only when the current instance satisfies the pre-condition. The update of variables $\bar{x}^T$ is valid if the next instance satisfies the post-condition. Variables can be changed arbitrarily during a service activation, as long as the post condition holds. This feature allows services to also model actions by external actors who provide input into the workflow by setting the value of non-propagated variables. Such actors may even include humans or other parties whose behavior is not deterministic. For example, a bank manager carrying out a "loan decision" action can be modeled by a service whose result is stored in a variable and whose value is restricted by the post-condition to either "Approve" or "Deny". Note that deterministic actors can be modeled by simply using tighter post-conditions.

As will be seen in the formal definition, $+S^T(\bar{s}^T)$ causes an insertion of the *current* value of $\bar{s}^T$ into S^T, while $-S^T(\bar{s}^T)$ causes the removal of some non-deterministically chosen tuple of S^T and its assignment as the *next* value of $\bar{s}^T$. In particular, if $\delta = \{+S^T(\bar{s}^T), -S^T(\bar{s}^T)\}$, the tuple inserted by $+S^T(\bar{s}^T)$ and the one retrieved by $-S^T(\bar{s}^T)$ are generally distinct, but may be the same as a degenerate case.

Example 11. *The **ManageTrips** task has 3 internal services:* Initialize*,* StoreTrip *and* RetrieveTrip*.* Initialize *creates a new trip with* $\texttt{flight\_id} = \texttt{hotel\_id} = \texttt{null}$*. When* RetrieveTrip *is called, a previously stored trip is chosen non-deterministically and removed from* TRIPS *for processing, and* (flight_id, hotel_id) *is set to be the chosen tuple. When* StoreTrip *is called, the current tuple* (flight_id, hotel_id) *is inserted into* TRIPS*. The latter two services are specified as follows.*

RetrieveTrip*:*
Pre: $\texttt{flight\_id} = \texttt{null} \wedge \texttt{hotel\_id} = \texttt{null}$
Post: $\texttt{status} =$ *"Shopping"*
Update: $\{-\texttt{TRIPS}(\texttt{flight\_id}, \texttt{hotel\_id})\}$

StoreTrip*:*
Pre: $\texttt{flight\_id} \neq \texttt{null} \vee \texttt{hotel\_id} \neq \texttt{null}$
Post: $\texttt{flight\_id} = \texttt{null} \wedge \texttt{hotel\_id} = \texttt{null} \wedge \texttt{status} =$ *"Shopping"*
Update: $\{+\texttt{TRIPS}(\texttt{flight\_id}, \texttt{hotel\_id})\}$

Fig. 3. Examples of two services.

As seen above, internal services of a task cause transitions on the data local to the task. Interactions among tasks are specified using two kinds of special services called the *opening*-services and *closing*-services. Specifically, each task T is equipped with an opening service σ^o_T and a closing service σ^c_T. Each non-root task T can be activated by its parent task via a call to σ^o_T which includes passing parameters to T that initialize its input variables $\bar{x}^T_{\text{in}}$. When T terminates (if ever), it returns to the parent the contents of its output variables $\bar{x}^T_{\text{out}}$ via a call to σ^c_T. Moreover, calls to σ^o_T are guarded by a condition on the parent's artifact variables, and closing calls to σ^c_T are guarded by a condition on the artifact variables of T.

Definition 12. *Let T_c be a child of a task T in $\mathcal{A}$.*
*(i) The **opening-service** $\sigma^o_{T_c}$ of T_c is a tuple $\langle \pi, f_{in} \rangle$, where π is a (pre-)condition over $\bar{x}^T$, and f_{in} is a 1-1 mapping from $\bar{x}^{T_c}_{\mathsf{in}}$ to $\bar{x}^T$ (called the input variable mapping). We denote $range(f_{in})$ by $\bar{x}^T_{T_c\downarrow}$ (the variables of T passed as input to T_c).*
*(ii) The **closing-service** $\sigma^c_{T_c}$ of T_c is a tuple $\langle \pi, f_{out} \rangle$, where π is a (pre-)condition over $\bar{x}^{T_c}$, and f_{out} is a 1-1 mapping from $\bar{x}^{T_c}_{\mathsf{out}}$ to $\bar{x}^T$ (called the output variable mapping). We denote $range(f_{out})$ by $\bar{x}^T_{T_c\uparrow}$, referred to as the **returned variables** from T_c. It is required that $\bar{x}^T_{T_c\uparrow} \cap \bar{x}^T_{\mathsf{in}} = \emptyset$.*

Requiring $\bar{x}^T_{T_c\uparrow} \cap \bar{x}^T_{\mathsf{in}} = \emptyset$ means that a returning child task cannot overwrite the input variables of the parent task, so that the values of the input variables stay unchanged throughout an execution of the task. While the definition allows the return of numeric variables, it turns out that for the purpose of verification one can assume that only ID variables are returned. One can additionally assume that the sets of variables returned by different subtasks are disjoint. The discussion of the simplifications is postponed to Section 3, since they must be considered in the context of the property language HLTL-FO.

For uniformity of notation, we also equip the root task T_1 with a service $\sigma^o_{T_1}$ with pre-condition *true* that initiates the computation by providing a valuation to a designated subset $\bar{x}^{T_1}_{in}$ of $\bar{x}^{T_1}$ (the input variables of T_1), and a service $\sigma^c_{T_1}$ whose pre-condition is *false* (so it never occurs in a run). For a task T we denote by Σ_T the set of its internal services, $\Sigma^{oc}_T = \Sigma_T \cup \{\sigma^o_T, \sigma^c_T\}$, $\Sigma^{obs}_T = \Sigma^{oc}_T \cup \{\sigma^o_{T_c}, \sigma^c_{T_c} \mid T_c \in child(T)\}$, and $\Sigma^\delta_T = \Sigma_T \cup \{\sigma^o_T\} \cup \{\sigma^c_{T_c} \mid T_c \in child(T)\}$. Intuitively, Σ^{obs}_T consists of the services observable in runs of task T and Σ^δ_T consists of services whose application can modify the variables $\bar{x}^T$.

Example 13. *The opening-service $\sigma^o_{T_4}$ of the **BookTrip** task has pre-condition* `flight_id` $\neq$ `null` $\wedge$ `hotel_id` $\neq$ `null` $\wedge$ `status` $=$ *"Shopping", meaning that both the hotel and flight have been chosen by the customer but are not yet paid. The input variables $\bar{x}^{T_4}_{in}$ of **BookTrip** are* {`flight_id`, `hotel_id`}*, which are mapped by the mapping f_{in} to variables* {`flight_id`, `hotel_id`} *of **ManageTrips**. The closing-service $\sigma^c_{T_4}$ of the **BookTrip** task has pre-condition* `status` $=$ *"Paid", meaning that the trip was successfully paid. The output variables $\bar{x}^{T_4}_{out}$ of **BookTrip** are* {`status`, `amount_paid`}*, which are mapped by f_{out} to the identically named variables* {`status`, `amount_paid`} *of **ManageTrips** (the returned variables $\bar{x}^{T_1}_{T_4\uparrow}$ from **BookTrip**).*

Definition 14. *A* Hierarchical Artifact System *(HAS) is a triple $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$, where $\mathcal{A}$ is an artifact schema, Σ is a set of services of tasks in $\mathcal{A}$ including σ^o_T and σ^c_T for each task T of $\mathcal{A}$, and Π is a condition over $\bar{x}^{T_1}_{in}$ (where T_1 is the root task).*

2.2 Semantics of HAS

We next define the semantics of HAS. Intuitively, a run of a HAS on a database D consists of an infinite sequence of transitions among HAS instances (also referred to as configurations, or snapshots), starting from an initial artifact tuple satisfying pre-condition Π, and empty artifact relations. At each snapshot, each active task T can open a subtask T_c if the pre-condition of the opening service of T_c holds, and the values of a subset of $\bar{x}^T$ are passed to T_c as its input variables. T_c can be closed if the pre-condition of its closing service is satisfied. When T_c is closed, the values of the return variables of T_c are sent to T. An internal service of T can only be applied after all active subtasks of T have returned their answer.

Tree of Local Runs. Because of the hierarchical structure, and the locality of task specifications, the actions of concurrently active children of a given task are independent of each other and can be arbitrarily interleaved. To capture just the essential information, factoring out the arbitrary interleavings, we first define the notion of *local run* and *tree of local runs*. Intuitively, a local run of a task consists of a sequence of services of the task, together with the transitions they cause on the task's local artifact variables and relation. The tasks' input and output are also specified. A tree of local runs captures the relationship between the local runs of tasks and those of their subtasks, including the passing of inputs and results. Then the runs of the full artifact system simply consist of all legal interleavings of transitions represented in the tree of local runs, lifted to full HAS instances (we refer to these as *global runs*). We begin by defining instances of tasks and local transitions. For a mapping M, we denote by $M[a \mapsto b]$ the mapping that sends a to b and agrees with M everywhere else.

Definition 15. *Let $T = \langle \bar{x}^T, S^T, \bar{s}^T, \bar{x}^T_{\text{in}}, \bar{x}^T_{\text{out}} \rangle$ be a task in Γ and D a database instance over* DB. *An* instance *of T is a pair (ν, S) where ν is a valuation of $\bar{x}^T$ and S an instance of S^T. For instances $I = (\nu, S)$ and $I' = (\nu', S')$ of T and a service $\sigma \in \Sigma^{obs}_T$, there is a local transition $I \xrightarrow{\sigma} I'$ if the following holds. If σ is an internal service $\langle \pi, \psi \rangle$, then:*

- $D \cup C \models \pi(\nu)$ *and* $D \cup C \models \psi(\nu')$
- $\nu'(y) = \nu(y)$ *for each* y *in* $\bar{x}^T_{\text{in}}$
- *if* $\delta = \{+S^T(\bar{s}^T)\}$, *then* $S' = S \cup \{\nu(\bar{s}^T)\}$, [3]
- *if* $\delta = \{-S^T(\bar{s}^T)\}$, *then* $\nu'(\bar{s}^T) \in S$ *and* $S' = S - \{\nu'(\bar{s}^T)\}$,
- *if* $\delta = \{+S^T(\bar{s}^T), -S^T(\bar{s}^T)\}$, *then* $\nu'(\bar{s}^T) \in S \cup \{\nu(\bar{s}^T)\}$ *and* $S' = (S \cup \{\nu(\bar{s}^T)\}) - \{\nu'(\bar{s}^T)\}$,
- *if* $\delta = \emptyset$ *then* $S' = S$.

If $\sigma = \sigma^o_{T_c} = \langle \pi, f_{in} \rangle$ is the opening-service for a child T_c of T then $D \cup C \models \pi(\nu)$, $\nu' = \nu$ and $S' = S$. If $\sigma = \sigma^c_{T_c}$ then $S = S'$, $\nu'|(\bar{x}^T - \bar{x}^T_{T_c\uparrow}) = \nu|(\bar{x}^T - \bar{x}^T_{T_c\uparrow})$ and $\nu'(z) = \nu(z)$ for every $z \in \bar{x}^T_{T_c\uparrow} \cap \text{VAR}_{id}$ for which $\nu(z) \neq$ `null`. Finally, if $\sigma = \sigma^c_T$ then $I' = I$.

Example 16. *Figure 4 shows two local transitions obtained by calling internal services* StoreTrip *and* RetrieveTrip *of* ***ManageTrips****. Figure 5 illustrates a transition caused by closing a sub-task.*

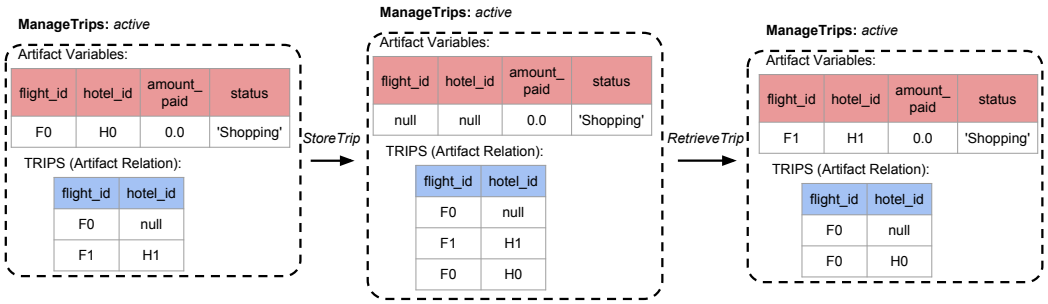

Fig. 4. Two transitions caused by the *StoreTrip* and the *RetrieveTrip* services.

Remark 17. *Recall that tuples retrieved from artifact relations are selected non-deterministically. For example, the* RetrieveTrip *service above extracts an arbitrary trip from the* TRIPS *relation. However, it may be useful to be able to select a particular trip for retrieval. While this capability is not explicitly provided, it can be simulated. Extracting the trip with a specified* `flight_id` *can be done as follows:*

[3] All artifact relation operations preserve the order of variables/attributes.

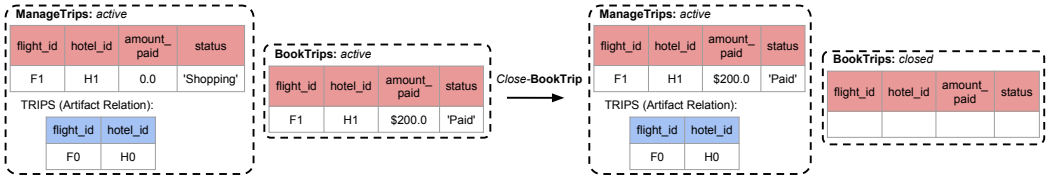

Fig. 5. Transition caused by a closing service.

(1) *an internal service retrieves a trip non-deterministically*
(2) *a subtask T is called and returns the desired* `flight_id`
(3) *the retrieved trip and the chosen* `flight_id` *are passed to another subtask T' which checks whether the trip has the chosen* `flight_id`*. The run blocks (so is invalidated) if this is not the case.*

We now define local runs.

Definition 18. *Let $T = \langle \bar{x}^T, S^T, \bar{s}^T, \bar{x}^T_{\text{in}}, \bar{x}^T_{\text{out}} \rangle$ be a non-root task in Γ and D a database instance over* DB*. A* local run *of T over D is a triple $\rho_T = (\nu_{in}, \nu_{out}, \{(I_i, \sigma_i)\}_{0 \le i < \gamma})$, where:*

- $\gamma \in \mathbb{N} \cup \{\omega\}$
- *for each $i \ge 0$, $I_i = (\nu_i, S_i)$ is an instance of T and $\sigma_i \in \Sigma^{obs}_T$*
- *ν_{in} is a valuation of $\bar{x}^T_{in}$*
- *$\sigma_0 = \sigma^o_T$ and $S_0 = \emptyset$,*
- *$\nu_0 | \bar{x}^T_{in} = \nu_{in}$, $\nu_0(z) = \texttt{null}$ for $z \in \text{VAR}_{id} - \bar{x}^T_{\text{in}}$ and $\nu_0(z) = 0$ for $z \in \text{VAR}_{\mathbb{R}} - \bar{x}^T_{\text{in}}$*
- *if for some i, $\sigma_i = \sigma^c_T$ then $\gamma \in \mathbb{N}$ and $i = \gamma - 1$ (and ρ_T is called a* returning *local run)*
- *$\nu_{out} = \nu_{\gamma-1} | \bar{x}^T_{\text{out}}$ if ρ_T is a returning run and $\perp$ otherwise*
- *a* segment *of ρ_T is a subsequence $\{(I_i, \sigma_i)\}_{i \in J}$, where J is a maximal interval $[a, b] \subseteq \{i \mid 0 \le i < \gamma\}$ such that no σ_j is an internal service of T for $j \in [a+1, b]$. A segment J is* terminal *if $\gamma \in \mathbb{N}$ and $b = \gamma - 1$ (and is called returning if $\sigma_{\gamma-1} = \sigma^c_T$ and blocking otherwise). Segments of ρ_T must satisfy the following properties. For each child T_c of T there is at most one $i \in J$ such that $\sigma_i = \sigma^o_{T_c}$. If J is not blocking and such an i exists, there is exactly one $j \in J$ for which $\sigma_j = \sigma^c_{T_c}$, and $j > i$. If J is blocking, there is* at most *one such j.*
- *for every $0 < i < \gamma$, $I_{i-1} \xrightarrow{\sigma_i} I_i$.*

Local runs of the root task T_1 are defined as above, except that ν_{in} is a valuation of $\bar{x}^{T_1}_{in}$ such that $D \cup C \models \Pi$, and $\nu_{out} = \perp$ (the root task never returns).

For a local run as above, we denote $\gamma(\rho_T) = \gamma$. Note that by definition of segment, a task can call each of its children tasks at most once between two consecutive services in Σ^{oc}_T and all of the called children tasks must complete within the segment, unless it is blocking. These restrictions are essential for decidability and are discussed in Section 4.

Observe that local runs take arbitrary inputs and allow for arbitrary return values from its children tasks. The valid interactions between the local runs of a tasks and those of its children is captured by the notion of *tree of local runs*.

Definition 19. *A* tree of local runs *is a directed labeled tree **Tree** where each node is an occurence of a local run ρ_T for some task T and every edge connects a local run of a task T with a local run of a child task T_c and is labeled with a non-negative integer i (denoted $i(\rho_{T_c})$). In addition, the following properties are satisfied. Let $\rho_T = (\nu^T_{in}, \nu^T_{out}, \{(I_i, \sigma_i)\}_{0 \le i < \gamma})$ be a node of **Tree**, where $I_i = (\nu_i, S_i)$, $i \ge 0$.*

*Let i be such that $\sigma_i = \sigma^o_{T_c}$ the opening service of some child T_c of T. There exists a unique edge labeled i from ρ_T to a node $\rho_{T_c} = (\nu_{in}, \nu_{out}, \{(I'_i, \sigma'_i)\}_{0 \le i < \gamma'})$ of **Tree**, and the following hold:*

- *$\nu_i(f_{in}(z)) = \nu_{in}(z)$ for every $z \in \bar{x}^{T_c}_{\text{in}}$ where f_{in} is the input variable mapping of $\sigma^o_{T_c}$*
- *ρ_{T_c} is a returning run iff there exists $j > i$ such that $\sigma_j = \sigma^c_{T_c}$; let k be the minimum such j. Then for every $z \in \bar{x}^{T_c}_{\text{out}}$ if either (1) $\nu_{k-1}(f_{out}(z)) = \texttt{null}$[4] or (2) variable z is numeric, then $\nu_k(f_{out}(z)) = \nu_{out}(z)$, where f_{out} is the output variable mapping of $\sigma^c_{T_c}$.*

*Finally, for every node ρ_T of **Tree**, if ρ_T is blocking then there exists a child of ρ_T that is not returning (so is infinite or blocking).*

The above definition is illustrated in Fig. 6. Note that a tree of local runs may generally be rooted at a local run of any task of Γ. We say that ***Tree*** is *full* if it is rooted at a local run of T_1.

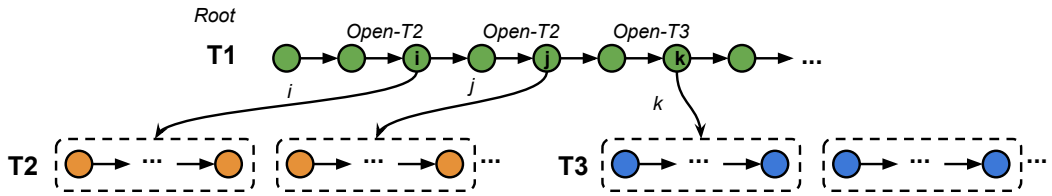

Fig. 6. A tree of local runs.

Global runs. Intuitively, a global run of Γ on database instance D over $\mathcal{DB}$ is an infinite sequence $\rho = \{(I_i, \sigma_i)\}_{i \ge 0}$, where each I_i is an instance (ν_i, stg_i, D, S_i) of $\mathcal{A}$ and $\sigma_i \in \Sigma$, resulting from a tree of local runs by interleaving its transitions, lifted to full HAS instances. Let D be a database and ***Tree*** a full tree of local runs over D. For a local run $\rho = (\nu_{in}, \nu_{out}, \{(I_m, \sigma_m)\}_{m<\gamma})$ (where $I_m = (\nu_m, S_m)$) and $i < \gamma$, we denote by $\sigma(\rho, i) = \sigma_i$, $\nu(\rho, i) = \nu_i$, and $S(\rho, i) = S_i$. Let $\preceq$ be the pre-order on the set $\{(\rho, i) \mid \rho \in \textbf{\textit{Tree}}, 0 \le i < \gamma(\rho)\}$ defined as the smallest reflexive-transitive relation containing the following:

(1) for each node ρ and $0 \le i \le j < \gamma(\rho)$, $(\rho, i) \preceq (\rho, j)$
(2) for each edge in ***Tree*** from ρ_T to ρ_{T_c} labeled i, $(\rho_T, i) \preceq (\rho_{T_c}, 0)$ and $(\rho_{T_c}, 0) \preceq (\rho_T, i)$. Additionally, if ρ_{T_c} is returning and m is the smallest $j > i$ for which $\sigma(\rho_T, j) = \sigma^c_{T_c}$, then $(\rho_{T_c}, \gamma(\rho_{T_c})) \preceq (\rho_T, m)$ and $(\rho_T, m) \preceq (\rho_{T_c}, \gamma(\rho_{T_c}))$.

Let $\approx$ be the equivalence relation induced by $\preceq$ (i.e., $a \approx b$ iff $a \preceq b$ and $b \preceq a$). Note that all classes of $\approx$ are singletons except for the ones induced by (2), which are of the form $\{(\rho_1, i), (\rho_2, j)\}$ where $\sigma(\rho_1, i) = \sigma(\rho_2, j) \in \{\sigma^o_T, \sigma^c_T\}$ for some task T. For an equivalence class ε of $\approx$ we denote by $\sigma(\varepsilon)$ the unique service of elements in ε. A *linearization* of $\preceq$ is an enumeration of the equivalence classes of $\approx$ consistent with $\preceq$. Consider a linearization $\{\varepsilon_i\}_{i \ge 0}$ of $\preceq$. Note that $\varepsilon_0 = (\rho_{T_1}, 0)$ and let $\nu(\rho_{T_1}, 0) = \nu_0$. A global run induced by $\{\varepsilon_i\}_{i \ge 0}$ is a sequence $\rho = \{(\bar{I}_i, \sigma_i)\}_{i \ge 0}$ such that $\sigma_i = \sigma(\varepsilon_i)$ and each $\bar{I}_i$ is an instance $(\bar{\nu}_i, stg_i, D, \bar{S}_i)$ of $\mathcal{A}$, defined inductively as follows. For $i = 0$,

- $\bar{\nu}_0(\bar{x}^{T_1}) = \nu_0(\bar{x}^{T_1})$ (and arbitrary on other variables)
- $stg_0 = \{T_1 \mapsto \texttt{active}, T_i \mapsto \texttt{init} \mid 2 \le i \le k\}$
- $\bar{S}_0 = \{S^{T_i} \mapsto \emptyset \mid 1 \le i \le k\}$.

[4]Although an ID variable with non-null values cannot be overwritten by a returning child task, it can be reset to `null` later by an internal transition.

For $i > 0$, $\bar{I}_i$ is defined as follows. Suppose first that $\varepsilon_i = \{(\rho, j)\}$ where ρ is a local run of task T and $\sigma(\rho, j)$ is an internal service of T. Then $\bar{\nu}_i = \bar{\nu}_{i-1}[\bar{x}^T \mapsto \nu(\rho, j)(\bar{x}^T)]$, $\bar{S}_i = \bar{S}_{i-1}[S^T \mapsto S(\rho, j)]$, and $stg_i = stg_{i-1}[\bar{T} \mapsto \texttt{init} \mid \bar{T} \in desc(T)]$. Now suppose $\varepsilon = \{(\rho_T, j), (\rho_{T_c}, 0)\}$, where T_c is a child of T, ρ_T and ρ_{T_c} are local runs of T and T_c, and $\sigma(\varepsilon) = \sigma^o_{T_c}$. Then $\bar{\nu}_i = \bar{\nu}_{i-1}[\bar{x}^{T_c} \mapsto \nu(\rho_{T_c}, 0)(\bar{x}^{T_c})]$, $\bar{S}_i = \bar{S}_{i-1}[S^{T_c} \mapsto \emptyset]$, and $stg_i = stg_{i-1}[T_c \mapsto \texttt{active}]$. Finally, suppose $\varepsilon = \{(\rho_T, j), (\rho_{T_c}, \gamma - 1)\}$ where $\sigma(\varepsilon) = \sigma^c_{T_c}$. Then $\bar{\nu}_i = \bar{\nu}_{i-1}[\bar{x}^T \mapsto \nu(\rho_T, j)(\bar{x}^T)]$, $stg_i = stg_{i-1}[T_c \mapsto \texttt{closed}]$, and $\bar{S}_i = S_{i-1}[S^{T_c} \mapsto \emptyset]$.

We denote by $\mathcal{L}(\boldsymbol{Tree})$ the set of global runs induced by linearizations of $\preceq$. The set of global runs of Γ on a database D is $Runs_D(\Gamma) = \bigcup\{\mathcal{L}(\boldsymbol{Tree}) \mid \boldsymbol{Tree}$ is a full tree of local runs of Γ on $D\}$ and the set of global runs of Γ is $Runs(\Gamma) = \bigcup_D Runs_D(\Gamma)$.

3 HIERARCHICAL LTL-FO

In order to specify temporal properties of HAS's we use an extension of LTL (linear-time temporal logic). Recall that LTL is propositional logic augmented with temporal operators **X** (next), **U** (until), **G** (always) and **F** (eventually) (e.g., see [33]). Their semantics is reviewed in Appendix A.1. An extension of LTL in which propositions are interpreted as FO sentences has previously been defined to specify properties of sequences of structures [60], and in particular of runs of artifact systems [21, 26]. The extension is denoted by LTL-FO. In order to specify properties of HAS's, we shall use a variant of LTL-FO, called *hierarchical* LTL-FO, denoted HLTL-FO. Intuitively, an HLTL-FO formula uses as building blocks LTL-FO formulas acting on local runs of individual tasks, referring only to the database and local data, and can recursively state HLTL-FO properties on runs resulting from calls to children tasks. This closely mirrors the hierarchical execution of tasks, and is a natural fit for this computation model. In addition to its naturalness, the choice of HLTL-FO has several technical justifications. First, verification of LTL-FO (and even LTL) properties is not possible for HAS's.

Theorem 20. *It is undecidable, given an LTL-FO formula φ and a HAS $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$, whether $\Gamma \models \varphi$. Moreover, this holds even for LTL formulas over Σ (restricting the sequence of services in a global run).*

The proof, provided in Appendix A.2, is by reduction from repeated state reachability in VASS with resets and bounded lossiness, whose undecidability follows from [48]. Essentially, when defined on global runs, LTL is expressive enough to encode the transitions of a VASS using the interleavings of multiple tasks. When combined with resets of counters, which can be simulated by opening/closing of tasks, verification becomes undecidable.

Another technical argument in favor of HLTL-FO is that it only expresses properties that are invariant under interleavings of independent tasks. Interleaving invariance is not only a natural soundness condition, but also allows more efficient model checking by *partial-order reduction* [53]. Moreover, HLTL-FO enjoys a pleasing completeness property: it expresses, in a reasonable sense, *all* interleaving-invariant LTL-FO properties of HAS's. This is discussed at the end of the section.

To illustrate the difference between LTL-FO and HLTL-FO, we exhibit a simple LTL property that is not expressible in HLTL.

Example 21. *Referring to our travel booking example, suppose that the opening services of* ***AddFlight*** *and* ***AddHotel*** *have preconditions* `flight_id = null` *and* `hotel_id = null`*, respectively. Thus, the two tasks may be active at the same time. Suppose that* ***AddFlight*** *has an internal service* ChooseFlight

and ***AddHotel*** *has an internal service* ChooseHotel. *Consider the LTL property*

$$\mathbf{G}\left((\sigma^o_{\textbf{\textit{AddFlight}}} \wedge \mathbf{F}\ \sigma^o_{\textbf{\textit{AddHotel}}}) \rightarrow (\ \neg\text{ChooseHotel}\ \mathbf{U}\ \text{ChooseFlight}\ )\right)$$

stating that whenever the ***AddFlight*** *task is called before the* ***AddHotel*** *task, the hotel is not chosen before the flight. Clearly, this property is violated by the example, because it is not invariant with respect to legal interleavings of services. Indeed, when* ***AddFlight*** *and* ***AddHotel*** *are simulteneously active, their internal services may interleave arbitrarily. Such properties are conveniently filtered out by HLTL, which only expresses interleaving-invariant properties.*

We next define HLTL-FO. We first define the propositional version of the language, HLTL. Similarly to LTL-FO, HLTL-FO formulas are obtained by interpreting the propositions as statements about instances of tasks in a run.

Definition 22. *Let* $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ *be an artifact system where* $\mathcal{A} = \langle \mathcal{H}, \text{DB} \rangle$. *An HLTL formula* φ *over a task* T *of* $\mathcal{H}$ *is an expression defined as follows:*

$$\varphi ::= \Sigma_T^{obs} \mid P_T^{\varphi} \mid [\psi]_{T_c} \mid \mathbf{X}\,\varphi \mid \varphi\,\mathbf{U}\,\varphi \mid \mathbf{F}\,\varphi \mid \mathbf{G}\,\varphi \mid (\varphi \wedge \varphi) \mid (\neg\varphi)$$

where P_T^{φ} *is a finite set of propositions and* ψ *is an HLTL formula over task* $T_c \in \text{child}(T)$. *Additionally,* $P_{T'}^{\varphi} \cap P_{T''}^{\varphi} = \emptyset$ *for all distinct* T', T'' *in* $\mathcal{H}$. *The set of HLTL formulas over* T *is denoted* $HLTL(T)$.

Intuitively, a formula $[\psi]_{T_c}$ holds in a given configuration if T makes a call to T_c and the run of T_c resulting from the call satisfies ψ.

For an HLTL formula φ, we denote $P_{\mathcal{H}}^{\varphi} = \bigcup_{T \in \mathcal{H}} P_T^{\varphi}$. An HLTL-FO formula over task T is obtained from an HLTL formula over T by interpreting each proposition in $P_{\mathcal{H}}^{\varphi}$ as a quantifier-free FO formula referring to the variables and artifact relations of the tasks, and a fixed specified set of global variables. Informally, a proposition of P_T^{φ} mapped by f to a quantifier-free FO formula holds in a given configuration of T if the formula is true in that configuration. We next formally define HLTL-FO formulas.

Definition 23. *Let* $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ *be an artifact system where* $\mathcal{A} = \langle \mathcal{H}, \text{DB} \rangle$. *Let* $\bar{y}$ *be a finite sequence of variables in* $\text{VAR}_{id} \cup \text{VAR}_{\mathbb{R}}$ *disjoint from* $\bigcup_{T \in \mathcal{H}} \bar{x}^T$, *called* global variables. *Let* C_T *be the set of conditions on* $\bar{x}^T \cup \bar{y}$ *extended by allowing atoms of the form* $S^T(\bar{z})$ *in which all variables in* $\bar{z}$ *are in* $\bar{y} \cap \text{VAR}_{id}$. *An HLTL-FO formula over task* T *using global variables* $\bar{y}$ *is a pair* (φ, f) *(denoted for conciseness* φ_f*) where* φ *is an HLTL formula over* T *and* f *is a mapping on* $P_{\mathcal{H}}^{\varphi}$ *such that* $f(p) \in C_{T'}$ *for every* $p \in P_{T'}^{\varphi}$. *An HLTL-FO formula over* Γ *is an expression* $\forall \bar{y} \varphi_f$, *where* φ_f *is an HLTL-FO formula over task* T_1 *using global variables* $\bar{y}$.

Since HLTL-FO properties depend on local runs of tasks and their relationship to local runs of their descendants, their semantics is naturally defined using the full trees of local runs. We first define satisfaction by a local run in the tree, of HLTL-FO formulas with no global variables. This is done recursively. Let ***Tree*** be a full tree of local runs of Γ over some database D. Let φ_f be an HLTL-FO formula for task T, with no global variables. If we associate to each expression $[\psi]_{T_c}$ in φ a distinct proposition $[\psi]_{T_c}^{prop}$, φ can be viewed as an LTL formula using propositions in $P_T^{\varphi} \cup \Sigma_T^{obs} \cup \{[\psi]_{T_c}^{prop} \mid \psi \in HLTL(T_c), T_c \in child(T)\}$. Let $\rho_T = (\nu_{in}, \nu_{out}, \{(I_i, \sigma_i)\}_{i<\gamma})$ be a local run of T in ***Tree***. For each configuration (I_j, σ_j), we define the truth assignment induced on the propositions of φ by the function f. A proposition $\sigma \in \Sigma_T^{obs}$ holds in (I_j, σ_j) if $\sigma = \sigma_j$. For $p \in P_T^{\varphi}$, its induced truth value is that of the FO formula $f(p)$ in I_j. Finally, the induced truth value of $[\psi]_{T_c}^{prop}$ in (I_j, σ_j) is true iff $\sigma_j = \sigma_{T_c}^o$ and the local run of T_c connected to ρ_T in ***Tree*** by an edge labeled

j satisfies the HLTL-FO formula ψ_f. The formula φ_f is satisfied if the sequence of induced truth values of its propositions via f satisfies φ. Note that ρ_T may be finite, in which case a finite variant of the LTL semantics is used [24] (see Appendix A.1).

A full tree of local runs satisfies an HLTL-FO formula φ_f over T_1 if its root (a local run of T_1) satisfies φ_f. Finally, let $\varphi_f(\bar{y})$ be an HLTL-FO over T_1 with global variables $\bar{y}$. Then $\forall\bar{y}\varphi_f(\bar{y})$ is satisfied by ***Tree***, denoted ***Tree*** $\models \forall\bar{y}\varphi_f(\bar{y})$, if for every valuation ν of $\bar{y}$, ***Tree*** satisfies φ_{f^ν} where f^ν is obtained from f by replacing each y in $f(p)$ by $\nu(y)$ for every $p \in P$. Finally, Γ satisfies $\forall\bar{y}\varphi_f(\bar{y})$, denoted $\Gamma \models \forall\bar{y}\varphi_f(\bar{y})$, if ***Tree*** $\models \forall\bar{y}\varphi_f(\bar{y})$ for every database instance D and tree of local runs ***Tree*** of Γ on D.

The semantics of HLTL-FO on trees of local runs of a HAS also induces a semantics on the global runs of the HAS. Let $\forall\bar{y}\varphi_f(\bar{y})$ be an HLTL-FO formula and $\rho \in \mathcal{L}(\textbf{\textit{Tree}})$, where ***Tree*** is a full tree of local runs of Γ. We say that ρ satisfies $\forall\bar{y}\varphi_f(\bar{y})$ if ***Tree*** satisfies $\forall\bar{y}\varphi_f(\bar{y})$. This is well defined in view of the following easily shown fact: if $\rho \in \mathcal{L}(\textbf{\textit{Tree}}_1) \cap \mathcal{L}(\textbf{\textit{Tree}}_2)$ then $\textbf{\textit{Tree}}_1 = \textbf{\textit{Tree}}_2$.

Example 24. *The following property of the travel booking workflow can be specified in HLTL-FO:* if a discount is applied to the hotel reservation, then a compatible flight must be purchased without cancellation. *One typical way to defeat the policy would be for a user to first book the flight and the hotel with the discount price, but next cancel the flight trying to avoid paying a penalty. Detecting such bugs can be subtle, especially when they involve multiple tasks. The following HLTL-FO property of task* ***ManageTrips*** *says "if* ***BookTrip*** *is called and the discount is applied, then if* ***CancelTrip*** *is called next and the customer cancels the flight, then the hotel discount must also be canceled and deducted from the flight refund". The property is specified as the formula φ_f, where*

$$\varphi = \mathbf{G}\left(\texttt{Discounted} \rightarrow \mathbf{X}\left(\sigma^o_{\textbf{T5:CancelTrip}} \rightarrow [\mathbf{G}(\textit{CancelFlight} \rightarrow \texttt{Refund})]_{\textbf{T5:CancelTrip}}\right)\right),$$

CancelFlight is the name of the service for canceling only the flight in ***CancelTrip****, and f interprets the proposition* `Discounted` *as the subformula defined in Example 9, and* `Refund` *as the formula*

$$\exists q \exists p_1 \exists p_2\ \texttt{FLIGHTS}(\texttt{flight\_id}, q, \texttt{hotel\_id}) \wedge \texttt{HOTELS}(\texttt{hotel\_id}, p_1, p_2) \wedge \texttt{amount\_refunded} = q - (p_1 - p_2).$$

Simplifications. Before proceeding, we note that several simplifications to HLTL-FO formulas and HAS specifications can be made without impact on verification. First, although useful at the surface syntax, the global variables, as well as set atoms, can be easily eliminated from the HLTL-FO formula to be verified (Lemma 67 in Appendix A.3). It is also useful to note that one can assume, without loss of generality, two simplifications on artifact systems regarding the interaction of tasks with their subtasks: (i) for every task T, the set of variables passed to subtasks is disjoint with the set of variables returned by subtasks, and (ii) all variables returned by subtasks are non-numeric (Lemma 68 in Appendix A.3). In view of the above, we henceforth consider only properties with no global variables or set atoms, and artifact systems simplified as described.

Checking HLTL-FO properties using automata. We next show how to check HLTL-FO properties of trees of local runs of artifact systems. Before we do so, recall the standard construction of a Büchi automaton B_φ corresponding to an LTL formula φ [58, 63]. The automaton B_φ has exponentially many states and accepts precisely the set of ω-words that satisfy φ. Recall that we are interested in evaluating LTL formulas φ on both infinite *and* finite runs. It is easily seen that for the B_φ obtained by the standard construction there is a subset Q^{fin} of its states such that B_φ viewed as a finite-state automaton with final states Q^{fin} accepts precisely the finite words that satisfy φ (details omitted).

Consider now an artifact system Γ and let φ_f be an HLTL-FO formula over Γ. Consider a full tree ***Tree*** of local runs. For task T, denote by Φ_T the set of sub-formulas $[\psi]_T$ occurring in φ and by 2^{Φ_T} the set of truth assignments to these formulas. For each T and $\eta \in 2^{\Phi_T}$, let $B(T, \eta)$ be the Büchi automaton constructed from the formula

$$\left(\wedge_{\psi \in \Phi_T, \eta(\psi)=1}\ \psi\right) \wedge \left(\wedge_{\psi \in \Phi_T, \eta(\psi)=0}\ \neg\psi\right)$$

and define the collection of automata $\mathcal{B}_\varphi = \{B(T, \eta) \mid T \in \mathcal{H}, \eta \in 2^{\Phi_T}\}$.

We now define acceptance of ***Tree*** by $\mathcal{B}_\varphi$. An *adornment* of ***Tree*** is a mapping α associating to each edge from ρ_T to ρ_{T_c} a truth assignment in $2^{\Phi_{T_c}}$. ***Tree*** is accepted by $\mathcal{B}_\varphi$ if there exists an adornment α such that:

- for each local run ρ_T of T with no outgoing edge and incoming edge with adornment η, ρ_T is accepted by $B(T, \eta)$
- for each local run ρ_T of T with incoming edge labeled by η, $\alpha(\rho_T)$ is accepted by $B(T, \eta)$, where $\alpha(\rho_T)$ extends ρ_T by assigning to each configuration $(\rho_j, \sigma^o_{T_c})$ the truth assignment in $2^{\Phi_{T_c}}$ adorning its outgoing edge labeled j. (Recall that in configurations (I_j, σ_j) for which $\sigma_j \neq \sigma^o_{T_c}$, all formulas in Φ_{T_c} are *false* by definition.)
- $\alpha(\rho_{T_1})$ is accepted by the Büchi automaton B_φ where $\alpha(\rho_{T_1})$ is defined as above.

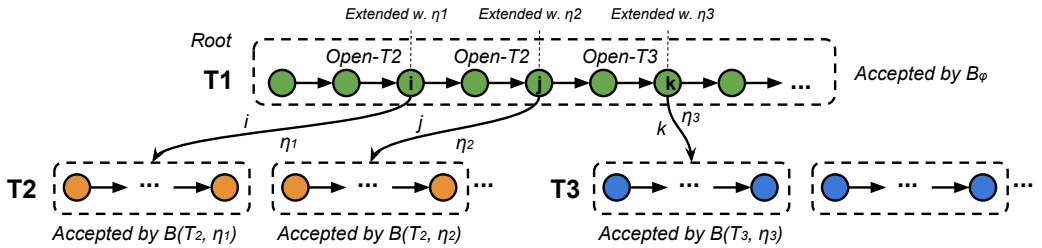

Fig. 7. A tree of local runs accepted by $\mathcal{B}_\varphi$.

The definition of acceptance is illustrated in Figure 7. The following can be shown.

Lemma 25. *A full tree of local runs* ***Tree*** *satisfies φ_f iff* ***Tree*** *is accepted by $\mathcal{B}_\varphi$.*

HLTL-FO vs. interleaving-invariant LTL-FO. We next show that HLTL-FO expresses, in a reasonable sense, all interleaving-invariant LTL-FO properties. We consider a notion of interleaving-invariance of LTL-FO formulas based on their propositional structure, rather than the specifics of the propositions' interpretation (which may lead to "accidental" invariance). In view of Lemma 67, we consider only formulas with no global variables or set atoms. We first recall the logic LTL-FO, slightly adapted to our context. Let $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ be a HAS where $\mathcal{A} = \langle \mathcal{H}, \text{DB} \rangle$. An LTL-FO formula φ_f over Γ consists of an LTL formula φ with propositions $P \cup \Sigma$ together with a mapping f associating to each $p \in P$ a condition over $\bar{x}^T$ for some $T \in \mathcal{H}$ (and we say that $f(p)$ is over T) . Satisfaction of φ_f on a global run $\rho = \{(I_i, \sigma_i)\}_{i \geq 0}$ of Γ on database D, where $I_i = (v_i, stg_i, D, S_i)$, is defined as usual, modulo the following:

- $f(p)$ over T holds in (I_i, σ_i) iff $stg_i(T) = \texttt{active}$ and the condition $f(p)$ on $v_i(\bar{x}^T)$ holds;
- proposition σ in Σ holds in (I_i, σ_i) if $\sigma = \sigma_i$.

Thus, the information about (I_i, σ_i) relevant to satisfaction of φ_f consists of σ_i, the stage of each task (active or not), and the truth values in I_i of $f(p)$ for $p \in P$.

We now make more precise the notion of (propositional) invariance under interleavings. Consider an LTL-FO formula φ_f over Γ. Invariance under interleavings is a property of the propositional formula φ (so independent on the interpretation of propositions provided by f). Let $P \cup \Sigma$ be the set of propositions of φ and let P_T denote the subset of P for which $f(p)$ is a condition over $\bar{x}^T$. Thus, $\{P_T \mid T \in \mathcal{H}\}$ is a partition of P. We define the set $\mathcal{L}(\Gamma)$ of ω-words associated to Γ, on which φ operates. The alphabet, denoted $\mathbf{A}(\Gamma)$, consists of all triples (κ, stg, σ) where $\sigma \in \Sigma$, κ is a truth assignment to the propositions in P, and stg is a mapping associating to each $T \in \mathcal{H}$ its stage (`active`, `init` or `closed`). An ω-word $\{(\kappa_i, stg_i, \sigma_i)\}_{i\geq 0}$ over $\mathbf{A}(\Gamma)$ is in $\mathcal{L}(\Gamma)$ if the following hold:

(1) for each $i > 0$, if $\sigma_i \in \Sigma_T^\delta$, then κ_i and κ_{i-1} agree on all $P_{\bar{T}}$ where $\bar{T} \neq T$;
(2) the sequence of calls, returns, and internal services obeys the conditions on service sequences in global runs of Γ;
(3) for each $i > 0$ and $T \in \mathcal{H}$, $stg_i(T)$ is the stage of T as determined by the sequence of calls and returns in $\{\sigma_j\}_{j<i}$.

The formal definition of (2) and (3) mimics closely the analogous definition of global runs of HAS's (omitted). Consider an ω-word $u = \{(\kappa_i, stg_i, \sigma_i)\}_{i\geq 0}$ in $\mathcal{L}(\Gamma)$. We define the partial order $\preceq_u$ on $\{i \mid i \geq 0\}$ as the reflexive-transitive closure of the relation consisting of all pairs (i, j) such that $i < j$ and for some T, $\sigma_i, \sigma_j \in \Sigma_T^{obs}$. Observe that 0 is always the minimum element in $\preceq_u$. A linearization of $\preceq_u$ is a total order on $\{i \mid i \geq 0\}$ containing $\preceq_u$. One can represent a linearization of $\preceq_u$ as a sequence $\{i_j \mid j \geq 0\}$ such that $i_n \preceq_u i_m$ implies that $n \leq m$. For each such linearization α, we define the ω-word $u_\alpha = \{(\bar{\kappa}_j, \overline{stg}_j, \sigma_{i_j})\}_{j\geq 0}$ in $\mathcal{L}(\Gamma)$ as follows. The stage function is the one determined by the sequence of services. The functions $\bar{\kappa}_j$ are defined by induction as follows:

- $\bar{\kappa}_0 = \kappa_0$;
- if $j > 0$ and $\sigma_{i_j} \in \Sigma_T^\delta$ then $\bar{\kappa}_j = \bar{\kappa}_{j-1}[P_T \mapsto \kappa_{i_j}(P_T)]$

Intuitively, u_α is obtained from u by commuting actions that are incomparable with respect to $\preceq_u$, yielding the linearization α. We note that the relation $\preceq_u$ is the analog to our setting of Mazurkiewicz traces, used in concurrent systems to capture dependencies among process actions [31, 32, 49].

Definition 26. *An LTL-FO formula φ_f over Γ is propositionally invariant with respect to interleavings if for every $u \in \mathcal{L}(\Gamma)$ and linearization α of $\preceq_u$, $u \models \varphi$ iff $u_\alpha \models \varphi$.*

We can show the following (see Appendix A.4).

Theorem 27. *HLTL-FO expresses precisely the LTL-FO properties of HAS's that are propositionally invariant with respect to interleavings.*

4 RESTRICTIONS AND UNDECIDABILITY

We briefly review the main restrictions imposed on the HAS model and motivate them by showing that they are needed to ensure decidability of verification. Specifically, recall that the following restrictions are placed:

(1) in an internal transition of a given task (caused by an internal service), only the input parameters of the task are explicitly propagated from one artifact tuple to the next
(2) each task may overwrite upon return only `null` variables in the parent task
(3) the artifact variables of a task storing the values returned by its subtasks are disjoint from the task's input variables
(4) an internal transition can take place only if all active subtasks have returned
(5) each task has just one artifact relation

(6) the artifact relation of a task is reset to empty every time the task closes
(7) the tuple of artifact variables whose value is inserted or retrieved from a task's artifact relation is fixed
(8) each subtask may be called at most once between internal transitions of its parent

These restrictions are placed in order to control the data flow and recursive computation in the system. Lifting any of them leads to undecidability of verification, as stated informally next.

Theorem 28. *For each i, $1 \leq i \leq 8$, let $HAS^{(i)}$ be defined identically to HAS but without restriction (i) above. It is undecidable, given a $HAS^{(i)}$ Γ and an HLTL-FO formula φ_f over Γ, whether $\Gamma \models \varphi_f$.*

The proofs of undecidability for (1)-(7) are by reduction from the Post Correspondence Problem (PCP) [54, 57]. They make no use of arithmetic, so undecidability holds even without arithmetic constraints. The only undecidability result relying on arithmetic is (8). Indeed, restriction (8) can be lifted in the absence of numeric variables, with no impact on decidability or complexity of verification. This is because restriction (2) ensures that even if a subtask is called repeatedly, only a bounded number of calls have a non-vacuous effect.

The proofs using a reduction from the PCP rely on the same main idea: removal of the restriction allows to extract from the database a path of unbounded length in a labeled graph, and check that its labels spell a solution to the PCP. For illustration, the proof of undecidability for (2) using this technique is sketched in Appendix B.

We claim that the above restrictions remain sufficiently permissive to capture a wide class of applications of practical interest. This is confirmed by numerous examples of practical business processes modeled as artifact systems, that we encountered in our collaboration with IBM The restrictions limit the recursion and data flow among tasks and services. In practical workflows, the required recursion is rarely powerful enough to allow unbounded propagation of data among services. Instead, as also discussed in [21], recursion is often due to two scenarios:

- allowing 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 (its input parameters). A typical example is repeatedly providing credit card information until the payment goes through, while the order details remain unchanged.
- allowing a task to batch-process an unbounded collection of records, each processed independently, with unchanged input parameters (e.g. sending invitations to an event to all attendants on the list, for the same event details).

Moreover, we recently showed in [2] that HAS and HLTL-FO can express a realistic benchmark of workflows obtained from existing sets of business process specifications and properties by extending them with data-aware features. The benchmark was then used to evaluate the performance of a verifier we implemented using the techniques developed in the present paper.

5 VERIFICATION WITHOUT ARITHMETIC

In this section we consider verification for the case when the artifact system and the HLTL-FO property have no arithmetic constraints. We show in Section 6 how our approach can be extended when arithmetic is present.

The roadmap to verification is the following. Let Γ be a HAS and φ_f an HLTL-FO formula over Γ. To verify that every tree of local runs of Γ satisfies φ_f, we check that there is no tree of local

runs satisfying $\neg\varphi_f$, or equivalently, accepted by $\mathcal{B}_{\neg\varphi}$. Since there are infinitely many trees of local runs of Γ due to the unbounded data domain, and each tree can be infinite, an exhaustive search is impossible. We address this problem by developing a symbolic representation of trees of local runs, called *symbolic tree of runs*. The symbolic representation is subtle for several reasons. First, unlike the representations in [21, 26], it is not finite state. This is because summarizing the relevant information about artifact relations requires keeping track of the number of tuples of various isomorphism types. Second, the symbolic representation does not capture the full information about the actual runs, but just enough for verification. Specifically, we show that for every HLTL-FO formula φ_f, there exists a tree of local runs accepted by $\mathcal{B}_\varphi$ iff there exists a symbolic tree of runs accepted by $\mathcal{B}_\varphi$. We then develop an algorithm to check the latter. The algorithm relies on reductions to state reachability problems in Vector Addition Systems with States (VASS) [16].

One might wonder whether there is a simpler approach to verification of HAS, that reduces it to verification of a flat system (consisting of a single task). This could indeed be done in the absence of artifact relations, by essentially concatenating the artifact tuples of the tasks along the hierarchy that are active at any given time, and simulating all transitions by internal services. However, there is strong evidence that this is no longer possible when tasks are equipped with artifact relations. First, a naive simulation using a single artifact relation would require more powerful updating capabilities (e.g. resetting artifact relations to be empty) than available in the model. Adding these capabilities would result in a model expressive enough to simulate vector addition systems with resets where verification is undecidable [48]. Moreover, Theorem 20 shows that LTL is undecidable for hierarchical systems, whereas the results in this section imply that it is decidable for flat ones (as it coincides with HLTL for single tasks). While this does not rule out a simulation, it shows that there can be no effective simulation natural enough to be extensible to LTL properties. A reduction to the model of [21] is even less plausible, because of the lack of artifact relations. Note that, even if a reduction were possible, the results of [21] would be of no help in obtaining our lower complexities for verification, since the algorithm provided there is non-elementary in all cases.

We next embark upon the development outlined above.

5.1 Symbolic Representation

We begin by defining the symbolic analog of a local run, called *local symbolic run*. The symbolic tree of runs is obtained by connecting the local symbolic runs similarly to the way local runs are connected in trees of local runs.

Each local symbolic run is a sequence of symbolic representations of an actual instance within a local run of a task T. The representation has the following ingredients:

(1) an equality type of the artifact variables of T and the elements in the database reachable from them by navigating foreign keys up to a specified depth $h(T)$. This is called the *T-isomorphism type* of the variables.
(2) the T-isomorphism type of the input and return variables (if representing a returning local run)
(3) for each T-isomorphism type of the set variables of T together with the input variables, the net number of insertions of tuples of that type in S^T.

Intuitively, (1) and (2) are needed in order to ensure that the assumptions made about the database while navigating via foreign keys in tasks and their subtasks are consistent. The depth $h(T)$ is chosen to be sufficiently large to ensure the consistency. (3) is required in order to make sure that a

retrieval from S^T of a tuple with a given T-isomorphism type is allowed only when sufficiently many tuples of that type have been inserted in S^T.

We now formally define the symbolic representation, starting with T-isomorphism type. Let $\bar{x}^T$ be the variables of T. We define $h(T)$ as follows. Let FK be the foreign key graph of the schema DB and $F(n)$ be the maximum number of distinct paths of length at most n starting from any relation R in FK. Let $h(T) = 1 + |\bar{x}^T| \cdot F(\delta)$ where $\delta = 1$ if T is a leaf task and $\delta = \max_{T_c \in child(T)} h(T_c)$ otherwise.

Note that when the schema DB is acyclic, the maximum depth $h(T)$ is trivially bounded since starting from any arbitrary entry in the database, the longest path obtained by navigations with the keys/foreign keys has length bounded by the number of relations in DB. However, this is not the case when DB is cyclic, as the paths can be infinite. The maximum depth $h(T)$ of navigations is now determined by the number of variables in each task and the height of the hierarchy. Intuitively, when T is a leaf task, the maximum depth is bounded by the number of variables in T because the longest navigation path is obtained when all variables are used to form the path. When T is a non-leaf task, a path can be obtained by chaining the navigation paths in multiple child tasks by passing input and return variables, which gives the above recursive definition of $h(T)$. We explain this in more detail in the proof of Lemma 47.

We next define expressions that denote navigation via foreign keys starting from the set of id variables $\bar{x}^T_{id}$ of T. For each $x \in \bar{x}^T_{id}$ and $R \in$ DB, let x_R be a new symbol. An expression is a sequence $\xi_1.\xi_2.\ldots.\xi_m$, where $\xi_1 = x_R$ for some $x \in \bar{x}^T_{id}$ and $R \in$ DB, ξ_2 is an attribute of R, and for each i, $2 \leq i < m$, ξ_i is a foreign key and ξ_{i+1} is an attribute in the relation referenced by ξ_i. We define the length of $\xi_1.\xi_2.\ldots.\xi_m$ as m. A *navigation set* $\mathcal{E}_T$ is a set of expressions such that:

- for each $x \in \bar{x}^T_{id}$, $\mathcal{E}_T$ contains at most one expression x_R ($R \in$ DB)
- $\mathcal{E}_T$ consists of all expressions $x_R.w$ where $x_R \in \mathcal{E}_T$ and the length of $x_R.w$ is at most $h(T)$.

In other words, the expressions in $\mathcal{E}_T$ denote all possible ways of navigating via foreign keys from a given subset of $\bar{x}^T_{id}$, by paths of length at most $h(T)$. In particular, note that $\mathcal{E}_T$ is closed under prefix. We can now define T-isomorphism type. Let $\mathcal{E}^+_T = \mathcal{E}_T \cup \bar{x}^T \cup \{\texttt{null}, 0\}$. The *sort* of $e \in \mathcal{E}^+_T$ is numeric if $e \in \bar{x}^T_{\mathbb{R}} \cup \{0\}$ or $e = w.a$ where a is a numeric attribute; its sort is `null` if $e =$ `null` or $e = x \in \bar{x}^T_{id}$ and $x_R \notin \mathcal{E}_T$ for all $R \in$ DB; and its sort is ID(R) for $R \in$ DB if $e = x_R$, or $e = x \in \bar{x}^T_{id}$ and $x_R \in \mathcal{E}_T$, or $e = w.f$ where f is a foreign key referencing R.

Definition 29. *A T-isomorphism type τ consists of a navigation set $\mathcal{E}_T$ together with an equivalence relation $\sim_\tau$ over $\mathcal{E}^+_T$ such that:*

- *if $e \sim_\tau e'$ then e and e' are of the same sort;*
- *for every $\{x, x_R\} \subseteq \mathcal{E}^+_T$, $x \sim_\tau x_R$;*
- *for every e of sort* `null`*, $e \sim_\tau$* `null`*;*
- *if $u \sim_\tau v$ and $u.f, v.f \in \mathcal{E}_T$ then $u.f \sim_\tau v.f$.*

We call an equivalence relation $\sim_\tau$ as above an *equality type* for τ. The relation $\sim_\tau$ is extended to tuples componentwise.

The intuition underlying the above definition is the following. First, the relation $\sim_\tau$ is an equivalence relation over the navigation set $\mathcal{E}_T$ extended with the variables $\bar{x}^T$ and the constants. Two expressions can be equal in $\sim_\tau$ only when they are of the same sort, meaning that they are both numeric, nulls or navigations ending with foreign key attributes referencing the ID of the same relation. Second, for an ID variable x, if an expression x_R appears in $\mathcal{E}^+_T$, this means that x contains

a tuple id of relation R, and x and x_R are essentially the same. Finally, if two expressions u and v are equal, then the key and foreign key dependencies require that expressions $u.f$ and $v.f$ extending u and v with the same expression f must also be equal.

Note that τ provides enough information to evaluate conditions over $\bar{x}^T$. Satisfaction of a condition φ by an isomorphism type τ, denoted $\tau \models \varphi$, is defined as follows:

- $x = y$ holds in τ iff $x \sim_\tau y$,
- $R(x, y_1, \ldots, y_n, z_1, \ldots, z_m)$ holds in τ for relation $R(id, a_1, \ldots, a_n, f_1, \ldots, f_m)$ where the a_i's and f_i's are numeric and foreign key attributes respectively, iff $\{x_R.a_1, \ldots, x_R.a_n, x_R.f_1, \ldots, x_R.f_m\} \subseteq \mathcal{E}_T$, and $(y_1, \ldots, y_n, z_1, \ldots, z_m) \sim_\tau (x_R.a_1, \ldots, x_R.a_n, x_R.f_1, \ldots, x_R.f_m)$.
- Boolean combinations of conditions are standard.

Example 30. *Figure 8 shows an example of a T-isomorphism type. The database schema contains two relations $R(\mathrm{ID}, A)$ and $S(\mathrm{ID}, B, C)$ where A is a foreign key attribute referencing ID of S and $\{B, C\}$ are numeric attributes. The task T contains 3 ID variables $\{x, y, z\}$. The T-isomorphism type has the following expressions: variables $\{x, y, z\}$, constants $\{0, \mathtt{null}\}$ and a set of navigations $\{x_R.A, y_R.A, x_R.A.B, y_R.A.B, x_R.A.C, y_R.A.C\}$. Since the schema is acyclic, the navigation depth $h(T)$ is bounded by the depth of the foreign key graph. The edges in Fig. 8 represent the equality type $\sim_\tau$ where two expressions e and e' are connected if $e \sim_\tau e'$. Note that since $x_R.A \sim y_R.A$, to ensure that the FDs are satisfied, we must also have $x_R.A.B \sim_\tau y_R.A.B$ and $x_R.A.C \sim_\tau y_R.A.C$ in $\sim_\tau$.*

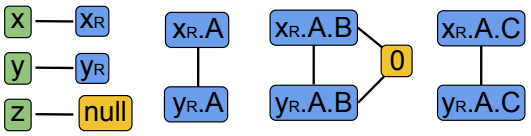

Fig. 8. A T-isomorphism type.

Let τ be a T-isomorphism type with navigation set $\mathcal{E}_T$ and equality type $\sim_\tau$. The projection of τ onto a subset of variables $\bar{z}$ of $\bar{x}^T$ is defined as follows. Let $\mathcal{E}_T|\bar{z} = \{x_R.e \in \mathcal{E}_T | x \in \bar{z}\}$ and $\sim_\tau |\bar{z}$ be the projection of $\sim_\tau$ onto $\bar{z} \cup \mathcal{E}_T|\bar{z} \cup \{\mathtt{null}, 0\}$. The projection of τ onto $\bar{z}$, denoted as $\tau|\bar{z}$, is a T-isomorphism type with navigation set $\mathcal{E}_T|\bar{z}$ and equality type $\sim_\tau |\bar{z}$. Furthermore, the projection of T-isomorphism onto $\bar{z}$ up to length k, denoted as $\tau|(\bar{z}, k)$, is defined as $\tau|\bar{z}$ with all expressions in $\mathcal{E}_T|\bar{z}$ with length more than k removed.

We apply variable renaming to isomorphism types as follows. Let f be a 1-1 partial mapping from $\bar{x}^T$ to $VAR_{id} \cup VAR_{\mathbb{R}}$ such that $f(\bar{x}^T_{id}) \subseteq VAR_{id}$, $f(\bar{x}^T_{\mathbb{R}}) \subseteq VAR_{\mathbb{R}}$ and $f(\bar{x}^T) \cap \bar{x}^T = \emptyset$. For a T-isomorphism type τ with navigation set $\mathcal{E}_T$, $f(\tau)$ is the isomorphism type obtained as follows. Its navigation set is obtained by replacing in $\mathcal{E}_T$ each variable x and x_R in $\mathcal{E}_T$ with $f(x)$ and $f(x)_R$, for $x \in dom(f)$. The relation $\sim_{f(\tau)}$ is the image of $\sim_\tau$ under the same substitution. As we shall see, variable renaming and projection are applied to an isomorphism type when a subset of the variables of a task are passed as input variables to a child task. Projection is also used when a tuple is inserted in an artifact relation.

As noted earlier, a T-isomorphism type captures all information needed to evaluate a condition on $\bar{x}_T$. However, the set S^T can contain unboundedly many tuples, which cannot be represented by a finite equality type. This is handled by keeping a set of counters for projections of T-isomorphism types on the variables relevant to S^T, that is, $(\bar{x}^T_{\mathrm{in}} \cup \bar{s}^T)$. We refer to the projection of a T-isomorphism type onto $(\bar{x}^T_{\mathrm{in}} \cup \bar{s}^T)$ as a TS-isomorphism type, and denote by $TS(T)$ the set of TS-isomorphism types of T. We will use counters to record the number of tuples in S^T of each TS-isomorphism type.

We can now define symbolic instances.

Definition 31. *A symbolic instance I of task T is a tuple $(\tau, \bar{c})$ where τ is a T-isomorphism type and $\bar{c}$ is a vector of integers where each dimension of $\bar{c}$ corresponds to a TS-isomorphism type.*

We denote by $\bar{c}(\hat{\tau})$ the value of the dimension of $\bar{c}$ corresponding to the TS-isomorphism type $\hat{\tau}$ and by $\bar{c}[\hat{\tau} \mapsto a]$ the vector obtained from $\bar{c}$ by replacing $\bar{c}(\hat{\tau})$ with a.

Example 32. *Examples of symbolic instances can be found in Fig. 9, where the task schema is the same as the one in Example 30 and the database schema is a single relation $R(\mathrm{ID}, A)$ with a single numeric attribute A. The symbolic instances consist of the T-isomorphism types and collections of counters of TS-isomorphism types.*

Definition 33. *A* local symbolic run *$\tilde{\rho}_T$ of task T is a tuple $(\tau_{in}, \tau_{out}, \{(I_i, \sigma_i)\}_{0 \leq i < \gamma})$, where:*

- *each I_i is a symbolic instance $(\tau_i, \bar{c}_i)$ of T*
- *each σ_i is a service in Σ_T^{obs}*
- *$\gamma \in \mathbb{N} \cup \{\omega\}$ (if $\gamma = \omega$ then $\tilde{\rho}_T$ is infinite, otherwise it is finite)*
- *τ_{in}, called the input isomorphism type, is a T-isomorphism type projected to $\bar{x}_{\mathsf{in}}^T$. And $\tau_{in} \models \Pi$ if $T = T_1$.*
- *at the first instance I_0, $\tau_0 | \bar{x}_{\mathsf{in}}^T = \tau_{in}$, for every $x \in \bar{x}_{id}^T - \bar{x}_{\mathsf{in}}^T$, $x \sim_{\tau_0} \mathtt{null}$, and for every $x \in \bar{x}_{\mathbb{R}}^T - \bar{x}_{\mathsf{in}}^T$, $x \sim_{\tau_0} 0$. Also $\bar{c}_0 = \bar{0}$ and $\sigma_0 = \sigma_T^o$.*
- *if for some i, $\sigma_i = \sigma_T^c$ then $\tilde{\rho}_T$ is finite and $i = \gamma - 1$ (and $\tilde{\rho}_T$ is called a* returning *run)*
- *τ_{out} is $\perp$ if $\tilde{\rho}_T$ is infinite or finite but $\sigma_{\gamma-1} \neq \sigma_T^c$, and it is $\tau_{\gamma-1} | (\bar{x}_{\mathsf{in}}^T \cup \bar{x}_{\mathsf{out}}^T)$ otherwise*
- *a* segment *of $\tilde{\rho}_T$ is a subsequence $\{(I_i, \sigma_i)\}_{i \in J}$, where J is a maximal interval $[a, b] \subseteq \{i \mid 0 \leq i < \gamma\}$ such that no σ_j is an internal service of T for $j \in [a+1, b]$. A segment J is* terminal *if $\gamma \in \mathbb{N}$ and $b = \gamma - 1$. Segments of $\tilde{\rho}_T$ must satisfy the following properties. For each child T_c of T there is at most one $i \in J$ such that $\sigma_i = \sigma_{T_c}^o$. If J is not terminal and such i exists, there is exactly one $j \in J$ for which $\sigma_j = \sigma_{T_c}^c$, and $j > i$. If J is terminal, there is* at most *one such j.*
- *for every $0 < i < \gamma$, I_i is a* **successor** *of I_{i-1} under σ_i (see below).*

The successor relation is defined next. We begin with some preliminary definitions.

A TS-isomorphism type $\hat{\tau}$ is *input-bound* if for every $s \in \bar{s}^T$, $s \nsim_{\hat{\tau}} \mathtt{null}$ implies that there exists an expression $x_R.w$ in $\hat{\tau}$ such that $x \in \bar{x}_{\mathsf{in}}^T$ and $x_R.w \sim_{\hat{\tau}} s$. We denote by $TS_{ib}(T)$ the set of input-bound types in $TS(T)$. Informally, a TS-isomorphism type is input-bound if the values of all the variables in $\bar{s}^T$ are uniquely determined by the values of the input variables of T. Since the values of the input variables are fixed in a local run of T, tuples $\bar{s}^T$ reachable from them in the same run by given navigations are unique. Therefore, the counter values for these TS-isomorphism types cannot exceed 1, and are treated as special cases when updated, as shown below.

For $\hat{\tau}, \hat{\tau}' \in TS(T)$, update δ of the form $\{+S^T(\bar{s}^T)\}$ or $\{-S^T(\bar{s}^T)\}$ and mapping $\bar{c}_{ib}$ from $TS_{ib}(T)$ to $\{0, 1\}$, we define the mapping $\bar{a}(\delta, \hat{\tau}, \hat{\tau}', \bar{c}_{ib})$ from $TS(T)$ to $\{-1, 0, 1\}$ as follows. Informally, the vector $\bar{a}(\delta, \hat{\tau}, \hat{\tau}', \bar{c}_{ib})$ specifies how the current counters need to be modified to reflect the update δ. Note that $\bar{a}_0$ is the mapping sending $TS(T)$ to 0.

- if $\delta = \{+S^T(\bar{s}^T)\}$, then $\bar{a}(\delta, \hat{\tau}, \hat{\tau}', \bar{c}_{ib})$ is $\bar{a}_0[\hat{\tau} \mapsto 1]$ if $\hat{\tau}$ is not input-bound, and $\bar{a}_0[\hat{\tau} \mapsto (1 - \bar{c}_{ib}(\hat{\tau}))]$ otherwise
- if $\delta = \{-S^T(\bar{s}^T)\}$, then $\bar{a}(\delta, \hat{\tau}, \hat{\tau}', \bar{c}_{ib}) = \bar{a}_0[\hat{\tau}' \mapsto -1]$
- if δ is $\{+S^T(\bar{s}^T), -S^T(\bar{s}^T)\}$ then $\bar{a}(\delta, \hat{\tau}, \hat{\tau}', \bar{c}_{ib}) = \bar{a}(\delta^+, \hat{\tau}, \hat{\tau}', \bar{c}_{ib}) + \bar{a}(\delta^-, \hat{\tau}, \hat{\tau}', \bar{c}_{ib})$ where $\delta^+ = \{+S^T(\bar{s}^T)\}$ and $\delta^- = \{-S^T(\bar{s}^T)\}$.

Next, we define the **successor** relation of symbolic instances. For symbolic instances $I = (\tau, \bar{c})$ and $I' = (\tau', \bar{c}')$, I' is a successor of I by applying service σ' iff:

- If σ' is an internal service $\langle \pi, \psi, \delta \rangle$, then for $\hat{\tau} = \tau|(\bar{x}^T_{\text{in}} \cup \bar{s}^T)$ and $\hat{\tau}' = \tau'|(\bar{x}^T_{\text{in}} \cup \bar{s}^T)$,
  - $\tau|\bar{x}^T_{\text{in}} = \tau'|\bar{x}^T_{\text{in}}$,
  - $\tau \models \pi$ and $\tau' \models \psi$,
  - $\bar{c}' \geq \bar{0}$ and $\bar{c}' = \bar{c} + \bar{a}(\delta, \hat{\tau}, \hat{\tau}', \bar{c}_{ib})$, where $\bar{c}_{ib}$ the restriction of $\bar{c}$ to $TS_{ib}(T)$.
- If σ' is an opening service $\langle \pi, f_{in} \rangle$ of subtask T_c, then $\tau = \tau' \models \pi$ and $\bar{c}' = \bar{c}$.
- If σ' is a closing service of subtask T_c, then for $\bar{x}^T_{const} = \bar{x}^T - \{x \in \bar{x}^T_{T_c\uparrow} | x \sim_\tau \texttt{null}\}$, $\tau'|\bar{x}^T_{const} = \tau|\bar{x}^T_{const}$ and $\bar{c}' = \bar{c}$.
- If σ' is the closing service $\sigma^c_T = \langle \pi, f_{out} \rangle$ of T, then $\tau \models \pi$ and $(\tau, \bar{c}) = (\tau', \bar{c}')$.

Example 34. *Figure 9 shows an example of two symbolic transitions. There is a single database relation* $R(\text{ID}, A)$*, the task* T *has 3 variables* $\{x, y, z\}$ *and* $\bar{s}^T = \{y, z\}$*. There is no input variable. The two applied services are* "insert_yz" *and* "retrieve_yz"*:*

- *The pre-condition of* insert_yz *is* $x = y$*, the post-condition is* $x = \texttt{null} \wedge y = \texttt{null} \wedge z = \texttt{null}$*, and the set update is* $\{+S^T(y, z)\}$*. So when applying* insert_yz*, the current tuple* (y, z) *is inserted to* S^T *and the values of all variables are set to* null*.*
- *The pre-condition of* retrieve_yz *is True, the post-condition is* $x = \texttt{null}$*, and the set update is* $\{-S^T(y, z)\}$*. So when applying* retrieve_yz*, a tuple* (y, z) *will be retrieved from* S^T*, the variables* $\{y, z\}$ *are set to the retrieved tuple, and* x *is set to* null*.*

Denote by $(\tau_1, \bar{c}_1)$*,* $(\tau_2, \bar{c}_2)$ *and* $(\tau_3, \bar{c}_3)$ *the 3 symbolic instances. In order for* $(\tau_2, \bar{c}_2)$ *to be a valid successor of* $(\tau_1, \bar{c}_1)$ *by applying* insert_yz*, the* T*-isomorphism types* τ_1 *and* τ_2 *must satisfy the pre-condition and post-condition of* insert_yz *respectively, and the counter vector* $\bar{c}_2$ *must be obtained from* $\bar{c}_1$ *by incrementing the counter for the projection* $\tau_1|\{y, z\}$ *by 1. Similarly, in order to apply* retrieve_yz*,* τ_2 *and* τ_3 *must satisfy the pre-condition and post-condition of* retrieve_yz*, and the counter for the projection* $\tau_3|\{y, z\}$ *must be decremented by 1 in* $\bar{c}_3$*.*

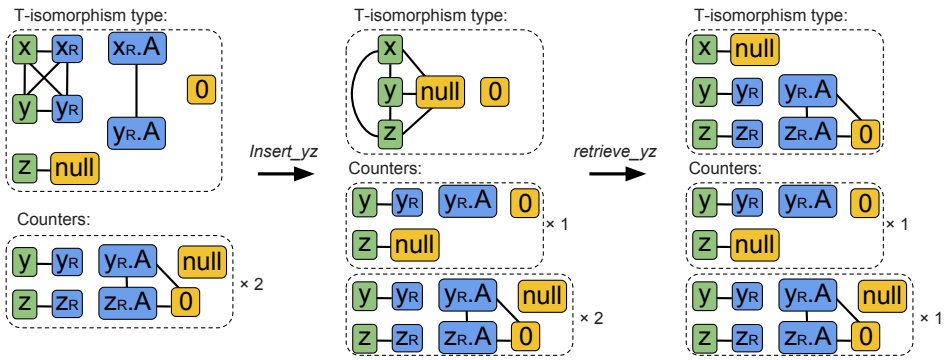

Fig. 9. Two local symbolic transitions.

Note that there is a subtle mismatch between transitions in actual local runs and in symbolic runs. In the symbolic transitions defined above, a service inserting a tuple in S^T *always* causes the correspoding counter to increase (except for the input-bound case). However, in actual runs, an inserted tuple may collide with an already existing tuple in the set, in which case the number of tuples does *not* increase. Symbolic runs do not account for such collisions (beyond the input-bound case), which raises the danger that they might overestimate the number of available tuples and allow impossible retrievals. Fortunately, the proof of Theorem 37 shows that collisions can be ignored at no peril. More specifically, it follows from the proof that for every actual local run with collisions satisfying an HLTL-FO property there exists an actual local run without collisions that satisfies the same property. The intuition is the following. First, given an actual run with collisions, one

can modify it so that only new tuples are inserted in the artifact relation, thus avoiding collisions. However, this raises a challenge, since it may require augmenting the database with new tuples. If done naively, this could result in an infinite database. The more subtle observation, detailed in the proof of Theorem 37, is that only a bounded number of new tuples must be created, thus keeping the database finite.

Definition 35. *A* symbolic tree of runs *is a directed labeled tree* ***Sym*** *in which each node is a local symbolic run $\tilde{\rho}_T$ for some task T, and every edge connects a local symbolic run of a task T with a local symbolic run of a child task T_c and is labeled with a non-negative integer i (denoted $i(\tilde{\rho}_{T_c})$). In addition, the following properties are satisfied. Let $\tilde{\rho}_T = (\tau_{in}, \tau_{out}, \{(I_i, \sigma_i)\}_{0 \leq i < \gamma})$ be a node of* ***Sym****. Let i be such that $\sigma_i = \sigma^o_{T_c}$ for some child T_c of T. There exists a unique edge labeled i from $\tilde{\rho}_T$ to a node $\tilde{\rho}_{T_c} = (\tau'_{in}, \tau'_{out}, \{(I'_i, \sigma'_i)\}_{0 \leq i < \gamma'})$ of* ***Sym****, and the following hold:*

- $\tau'_{in} = f^{-1}_{in}(\tau_i)|(\bar{x}^{T_c}_{\mathsf{in}}, h(T_c))$ *where f_{in} is the input variable mapping of $\sigma^o_{T_c}$*
- $\tilde{\rho}_{T_c}$ *is a returning run iff there exists $j > i$ such that $\sigma_j = \sigma^c_{T_c}$; let k be the minimum such j. Let $\bar{x}_r = \bar{x}^T_{T_c\downarrow}$ and $\bar{x}_w = \{x | x \in \bar{x}^T_{T_c\uparrow}, x \sim_{\tau_{k-1}} \mathtt{null}\}$. Then $\tau_k|(\bar{x}_r \cup \bar{x}_w, h(T_c)) = ((f_{in} \circ f_{out})(\tau_{out}))|(\bar{x}_r \cup \bar{x}_w)$ where f_{out} is the output variable mapping of $\sigma^c_{T_c}$.*

For every local symbolic run $\tilde{\rho}_T$ where $\gamma \neq \omega$ and $\tau_{out} = \perp$, there exists a child of $\tilde{\rho}_T$ which is not returning.

Now consider an HLTL-FO formula φ_f over Γ. Satisfaction of φ_f by a symbolic tree of runs is defined analogously to satisfaction by local runs, keeping in mind that as previously noted, isomorphism types of symbolic instances of T provide enough information to evaluate conditions over $\bar{x}^T$. The definition of acceptance by the automaton $\mathcal{B}_\varphi$, and Lemma 25, are also immediately extended to symbolic trees of runs. We state the following.

Lemma 36. *A symbolic tree of runs* ***Sym*** *over Γ satisfies φ_f iff* ***Sym*** *is accepted by $\mathcal{B}_\varphi$.*

The key result enabling the use of symbolic trees of runs is the following.

Theorem 37. *For an artifact system Γ and HLTL-FO property φ_f, there exists a tree of local runs* ***Tree*** *accepted by $\mathcal{B}_\varphi$, iff there exists a symbolic tree of runs* ***Sym*** *accepted by $\mathcal{B}_\varphi$.*

The *only-if* part is relatively straightforward and we outline the proof in Section 5.2. The *if* part is non-trivial. We prove it by showing a construction of an actual database and an accepted tree of local runs from any accepted symbolic tree of runs ***Sym***. The construction has 3 major components.

- First, for each *finite* local symbolic run, we construct an actual accepted local run over a local database (Lemma 42), using a global equality type that extends the local equality types by taking into account connections across instances resulting from the propagation of input variables and insertions/retrievals of tuples from S^T, and subject to satisfaction of the key constraints. The key challenge in this step is to show that our choice of $h(T)$, the maximal navigation depth in the symbolic representations, is sufficiently large to guarantee satisfaction of all key constraints (Lemma 47).
- Next, we apply the same construction to each *infinite* local symbolic run, resulting in an accepted infinite local run over an infinite database. The infinite database is then turned into a finite one by carefully merging data values, while avoiding any inconsistencies. One of the subtleties is showing that the mismatch between symbolic and actual transitions discussed above, leading to the possible overestimation by the counters in symbolic runs of the number of tuples available in artifact relations, is not dangerous (Lemma 60).
- Finally, all finite and infinite local runs are recursively combined into a tree of local runs by renaming and merging data values stored in the variables and the local databases.

5.2 Only-if: from actual runs to symbolic runs

Let ***Tree*** be a tree of local runs accepted by $\mathcal{B}_\varphi$ (with database D). The construction of ***Sym*** from ***Tree*** is simple. This can be done by replacing each local run $\rho_T \in$ ***Tree*** with a local symbolic run $\tilde{\rho}_T$. More precisely, let

$$\rho_T = (\nu_{in}, \nu_{out}, \{(J_i, \sigma_i)\}_{0 \le i < \gamma})$$

be a local run in ***Tree***, where $J_i = (\nu_i, S_i)$, We construct a corresponding local symbolic run

$$\tilde{\rho}_T = (\tau_{in}, \tau_{out}, \{(I_i, \sigma_i)\}_{0 \le i < \gamma})$$

For $0 \le i < \gamma$, $I_i = (\tau_i, \bar{c}_i)$ is constructed from (ν_i, S_i) as follows. The navigation set $\mathcal{E}_T$ of τ_i contains every x_R for every $x \in \bar{x}^T$ and R such that $\nu(x)$ is an ID of relation R in D. Then we define ν_i^* to be a mapping from $\mathcal{E}_T^+ = \mathcal{E}_T \cup \{0, \mathtt{null}\} \cup \bar{x}^T$ to actual values, where:

- $\nu_i^*(e) = e$ if $e \in \{0, \mathtt{null}\}$,
- $\nu_i^*(e) = \nu_i(x)$ for $e = x$ or $e = x_R$, and
- $\nu_i^*(e.\xi) = t.\xi$ if $\nu_i^*(e)$ is an ID of a tuple $t \in D$.

We construct the equality type $\sim_{\tau_i}$ such that for every e and e' in $\mathcal{E}_T^+$, $e \sim_{\tau_i} e'$ iff $\nu_i^*(e) = \nu_i^*(e')$. Also we let $\tau_{in} = \tau_0 | \bar{x}_{\text{in}}^T$ and $\tau_{out} = \tau_{\gamma-1} | \bar{x}_{\text{in}}^T \cup \bar{x}_{\text{out}}^T$ if $\nu_{out} \neq \perp$ and $\tau_{out} = \perp$ otherwise. Since D satisfies the functional dependencies, for every τ_i and expressions e and e', $e \sim_{\tau_i} e'$ implies that $\nu_i^*(e) = \nu_i^*(e')$, so for every attribute a, if $e.a$ and $e'.a$ are in the navigation set of τ_i, then $e.a \sim_{\tau_i} e'.a$ because $\nu_i^*(e.a) = \nu_i^*(e'.a)$.

To illustrate the construction of the local symbolic runs, consider the two actual transitions shown in Figure 10. The above construction yields the symbolic instances shown in Figure 9.

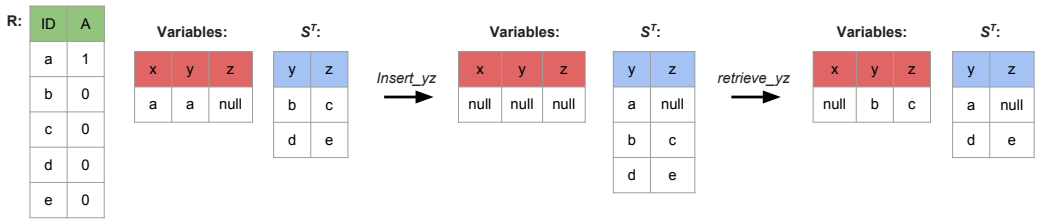

Fig. 10. Illustration of the construction of the local symbolic runs.

By construction of the τ_i's, the following facts hold:

Fact 38. *For every condition ψ over $\bar{x}^T$, $D \models \psi(\nu_i)$ iff $\tau_i \models \psi$.*

Fact 39. *For all i, i' and $\bar{x} \subseteq \bar{x}^T$, if $\nu_i(\bar{x}) = \nu_{i'}(\bar{x})$ then $\tau_i|\bar{x} = \tau_{i'}|\bar{x}$.*

Given $\{(\tau_i, \sigma_i)\}_{0 \le i < \gamma}$, the sequence of vectors of TS-isomorphism type counters $\{\bar{c}_i\}_{0 \le i < \gamma}$ is uniquely defined. Let $\tilde{\rho}_T = (\tau_{in}, \tau_{out}, \{(I_i, \sigma_i)\}_{0 \le i < \gamma})$. In view of Fact 38, it is easy to see that $\tilde{\rho}_T$ satisfies all items in the definition of local symbolic run that do not involve the counters. To show that $\tilde{\rho}_T$ is a local symbolic run, it remains to show that $\bar{c}_i \ge \bar{0}$ for $0 \le i < \gamma$. To see that this holds, we associate a sequence of counter vectors $\{\tilde{c}_i\}_{0 \le i < \gamma}$ to the local run ρ_T, where each $\tilde{c}_i$ provides, for each TS-isomorphism type $\hat{\tau}$, the number of tuples in S_i of TS-isomorphism type $\hat{\tau}$ (the TS-isomorphism type of a tuple $t \in S_i$ is defined analogously to the T-isomorphism type for each local instance). By definition, $\tilde{c}_i \ge \bar{0}$ for each $i \ge 0$. Thus it is sufficient to show that $\tilde{c}_i \le \bar{c}_i$ for each i. We show this by induction. For $i = 0$, $\tilde{c}_0 = \bar{c}_0 = 0$. Suppose $\tilde{c}_{i-1} \le \bar{c}_{i-1}$ and consider the transition under service σ_i in ρ_T and $\tilde{\rho}_T$. It is easily seen that $\tilde{c}_{i-1}$ and $\bar{c}_{i-1}$ are modified in the same way *except* in the case when $+S^T(\bar{s}^T) \in \delta$, $\hat{\tau}_{i-1}$ is not input-bound, and $\nu_{i-1}(\bar{s}^T) \in S_{i-1}$. In this case, if $\hat{\tau}$ is the TS-isomorphism type of $\nu_{i-1}(\bar{s}^T)$, $\tilde{c}_i(\hat{\tau}) = \tilde{c}_{i-1}(\hat{\tau})$ whereas $\bar{c}_i(\hat{\tau}) = \bar{c}_{i-1}(\hat{\tau}) + 1$. In all cases,

$\tilde{c}_i \leq \bar{c}_i$. Thus, $\tilde{\rho}_T$ is a local symbolic run. The fact that ***Sym*** is a tree of symbolic local runs follows from Fact 39, which ensures the consistency of the isomorphism types passed to and from subtasks. Finally, the fact that ***Sym*** is accepted by $\mathcal{B}_\varphi$ follows from acceptance of ***Tree*** by $\mathcal{B}_\varphi$ and Fact 38.

5.3 If part: from symbolic runs to actual runs

We denote by **FD** the set of key dependencies in the database schema DB and **IND** the set of foreign key dependencies. We show the following.

Lemma 40. *For every symbolic tree of runs **Sym** accepted by $\mathcal{B}_\beta$, there exists a tree **Tree** of local runs accepted by $\mathcal{B}_\beta$ with a finite database instance D where $D \models$ **FD**.*

Note that the above does not require that D satisfy **IND**. This is justified by the following.

Lemma 41. *For every tree of local runs **Tree** with database $D \models$ **FD** if **Tree** is accepted by $\mathcal{B}_\beta$ then there exists a finite database $D' \models$ **FD** $\cup$ **IND** such that **Tree** with database D' is also a tree of local runs accepted by $\mathcal{B}_\beta$.*

Proof. We can construct D' by adding tuples to D as follows. First, for each relation R such that R is empty in D, we add an arbitrary tuple t to R. Next, for each foreign key dependency $R_i[F] \subseteq R_j[ID]$, for each tuple t of R_i such that there is no tuple in R_j with id $t[F]$, we add to R_j a tuple t' where

- $t'[ID] = t[F]$, and
- $t'[attr(R_j) - \{ID\}] = t''[attr(R_j) - \{ID\}]$ where t'' is an existing tuple in R_j.

Tree with database D' is accepted by $\mathcal{B}_\beta$ since D' is an extension of D. Also D' is finite since the number of added tuples is at most linear in the sum of number of empty relations in D and the number of tuples in D that violate **IND**. □

To show Lemma 40, we begin with a construction of a local run ρ_T on a finite database D_T for each local symbolic run $\tilde{\rho}_T \in$ ***Sym***. The local runs are constructed so that they can be merged consistently into a tree of local runs ***Tree*** with a single finite database D. The major challenge in the construction of each ρ_T and D_T is that if $\tilde{\rho}_T$ is infinite, the size of S^T can grow infinitely, and a naive construction of ρ_T would require infinitely many distinct values in D_T. Our construction needs to ensure that D_T is always finite. For ease of exposition, we first consider the case where $\tilde{\rho}_T$ is finite and then extend the result to infinite $\tilde{\rho}_T$.

5.4 Handling finite local symbolic runs

Recall from the previous section that $\nu^*(e)$ denotes the value of expression e in database D_T with valuation ν of $\bar{x}^T$. By abuse of notation, we extend $\nu^*(e)$ to $e \in \{x_R.w | x \in \bar{x}^T, R \in \mathsf{DB}\} \cup \bar{x}^T \cup \{0, \mathtt{null}\}$ where there is no restriction on the length of w. So for expression $e = x_R.w$, $\nu^*(e)$ is the value in D_T obtained by foreign key navigation starting from the value $\nu^*(x)$ at relation R and by the sequence of attributes w, if such a value exists. Note that ν^* may be only partially defined since D_T may not satisfy all foreign key constraints. Analogously, we define $\nu^*_{in}(e)$ to be the value of e in D_T at valuation ν_{in} and $\nu^*_{out}(e)$ to be the value of e in D_T at valuation ν_{out}.

We prove the following, showing the existence of an actual local run corresponding to a finite local symbolic run. The lemma provides some additional information used when merging local runs into a final tree of runs.

Lemma 42. *For every finite local symbolic run $\tilde{\rho}_T = (\tau_{in}, \tau_{out}, \{(I_i, \sigma_i)\}_{0 \le i < \gamma})$ ($\gamma \neq \omega$), there exists a local run $\rho_T = (\nu_{in}, \nu_{out}, \{(\rho_i, \sigma_i)\}_{0 \le i < \gamma})$ on finite database $D_T \models$ **FD** such that for every $0 \le i < \gamma$,*

(i) *for every expression $e = x_R.w$ where $\nu_i^*(e)$ is defined, there exists expression $e' = x_R.w'$ where $|w'| \le h(T)$ such that $\nu_i^*(e) = \nu_i^*(e')$,*
(ii) *for all expressions $e, e' \in \mathcal{E}_T^+$ of τ_i, if $\nu_i^*(e)$ and $\nu_i^*(e')$ are defined, then $e \sim_{\tau_i} e'$ iff $\nu_i^*(e) = \nu_i^*(e')$, and*
(iii) *for $\delta = h(T_c)$ if $\sigma_i \in \{\sigma_{T_c}^o, \sigma_{T_c}^c\}$ for some $T_c \in child(T)$ and $\delta = 1$ otherwise, for every expression $e \in \mathcal{E}_T^+ - \{x_R.w | x \in \bar{x}^T, |w| > \delta\}$, $\nu_i^*(e)$ is defined.*

Part (i), needed for technical reasons, says that for all values v in D_T, if v is the value of expression $x_R.w$, then v is also the value of an expression $x_R.w'$ where the length of w' is within $h(T)$. Part (ii) says, intuitively, that the equality types in the symbolic local run and the constructed local run are the same. Part (iii) states that for every $0 \le i < \gamma$, at valuation ν_i, every expression e within δ steps of foreign key navigation from any variable x is defined in D_T. Since $\delta \ge 1$, this together with (ii) implies that for every condition π, $\tau_i \models \pi$ iff $D_T \models \pi(\nu_i)$. So if $\tilde{\rho}_T$ is accepted by some computation of a Büchi automaton $B(T, \eta)$ then ρ_T is also accepted by the same computation of $B(T, \eta)$.

We provide the proof of Lemma 42 in the remainder of the section. We first show that from each finite local symbolic run $\tilde{\rho}_T$, we can construct a *global isomorphism type* of $\tilde{\rho}_T$, which is essentially an equality type over the entire set of expressions in the symbolic instances of $\tilde{\rho}_T$. Then we show that the local run ρ_T and database D_T whose domain values are the equivalence classes of the global isomorphism type, satisfy the properties in Lemma 42.

Global isomorphism types. We prove Lemma 42 by constructing ρ_T and D_T from $\tilde{\rho}_T = (\tau_{in}, \tau_{out}, \{(I_i, \sigma_i)\}_{0 \le i < \gamma})$ ($\gamma \neq \omega$). We first introduce some additional notation.

Let $\mathcal{I}^+$ be the set of symbolic instances I_i of $\tilde{\rho}_T$ ($i < \gamma - 1$) such that $+S^T(\bar{s}^T) \in \delta_{i+1}$ and $\hat{\tau}_i$ is not input-bound. Similarly let $\mathcal{I}^-$ be the set of symbolic instances I_j ($j < \gamma$) such that $-S^T(\bar{s}^T) \in \delta_j$ and $\hat{\tau}_j$ is not input-bound. We define a one-to-one function `Retrieve` from $\mathcal{I}^-$ to $\mathcal{I}^+$ such that for every $I_i = \texttt{Retrieve}(I_j)$, $i < j$ and $\hat{\tau}_i = \hat{\tau}_j$. We say that I_j retrieves from I_i. As $\bar{c}_i \ge 0$ for every i, at least one mapping `Retrieve` always exists. Intuitively, `Retrieve` connects symbolic instance I_j to I_i such that I_j retrieves a tuple from S^T which has the same isomorphism type as a tuple inserted at I_i. For each $I_i = \texttt{Retrieve}(I_j)$, in the local run ρ_T we construct, valuations ν_i and ν_j have same values on variables $\bar{s}^T$. Here we ignore input-bound isomorphism types since these can be seen as part of the input isomorphism type: in ρ_T, instances having the same input-bound TS-isomorphism type have the same values on $\bar{s}^T$.

Fig. 11. An illustration of a life cycle.

Recall that a *segment* $S = \{(I_i, \sigma_i)\}_{a \le i \le b}$ is a maximum consecutive subsequence of $\{(I_i, \sigma_i)\}_{0 \le i < \gamma}$ such that σ_a is an internal service and for $a < i \le b$, σ_i is opening service or closing service of child tasks of T. For our choice of the `Retrieve` relation, we define a *life cycle* $L = \{(I_i, \sigma_i)\}_{i \in J}$ as a maximum subsequence of $\{(I_i, \sigma_i)\}_{0 \le i < \gamma}$ for $J \subseteq [0, \gamma)$ where for each pair of consecutive (I_a, σ_a) and (I_b, σ_b) in L where $a < b$, (I_a, σ_a) and (I_b, σ_b) are either in the same segment or $I_a = \texttt{Retrieve}(I_b)$ (illustrated in Figure 11). Note that a life cycle L is also a sequence of segments.

From the definition of local symbolic runs, we can show the following properties for segments and life cycles:

Lemma 43. (*i*) *For every segment* $S = \{(I_i, \sigma_i)\}_{a \le i \le b}$, *for every* $i, j \in [a, b]$ *where* $i < j$, *for* $\bar{x} = \{x | x \in \bar{x}^T, x \nprec_{\tau_i} \texttt{null}\}$, $\tau_i|\bar{x} = \tau_j|\bar{x}$. (*ii*) *For every life cycle* $L = \{(I_i, \sigma_i)\}_{i \in J}$, *for every* $i, j \in J$ *where* $i < j$, *for* $\bar{x} = \{x | x \in \bar{x}^T_{\text{in}} \cup \bar{s}^T, x \nprec_{\tau_i} \texttt{null}\}$, $\tau_i|\bar{x} = \tau_j|\bar{x}$.

Next, for each symbolic instance I_i, we define the *pruned* isomorphism type $\lambda_i = (\mathcal{E}_i, \sim_i)$ of I_i as follows. Intuitively, each λ_i is obtained from τ_i by removing expressions with "long" navigation from variables. Formally, let $\mathcal{E}^+_T$ be the extended navigation set of τ_i and $\mathcal{E}^-_T = \mathcal{E}^+_T - \{x_R.w | x \in \bar{x}^T, |w| > \delta\}$, where $\delta = 1$ if T is a leaf task, otherwise $\delta = \max_{T_c \in child(T)} h(T_c)$. The choice of δ ensures that the remaining information in each λ_i is sufficient for the consistency of keys and foreign keys within one single transition (i.e. an internal transition or a child task return). A *local expression* of I_i is a pair (i, e) where $e \in \mathcal{E}^-_T$, and we define $\mathcal{E}_i = \{(i, e) | e \in \mathcal{E}^-_T\}$ as the *local navigation set* of λ_i. We also define the *local equality type* $\sim_i$ of λ_i to be an equality type over $\mathcal{E}_i$ where $(i, e) \sim_i (i, e')$ iff $e \sim_{\tau_i} e'$, for every $e, e' \in \mathcal{E}^-_T$. Intuitively, the set $\mathcal{E}_i$ contains all expressions that are assigned with fresh values when the actual local run is constructed.

Then we define the global isomorphism type as follows. A global isomorphism type is a pair $\Lambda = (\mathcal{E}, \sim)$, where $\mathcal{E} = \bigcup_{0 \le i < \gamma} \mathcal{E}_i$ is called the *global navigation set* and $\sim$ is an equality type over $\mathcal{E}$ called *global equality type*. For each expression $e \in \mathcal{E}$, let $[e]$ denote its equivalence class with respect to $\sim$. The global equality type $\sim$ is constructed as follows:

(1) **Initialization:** $\sim \leftarrow \bigcup_{0 \le i < \gamma} \sim_i$
(2) **Chase:** Until convergence, merge two equivalence classes E and E' of $\sim$ if E and E' satisfy one of the following conditions:
- **Segment-Condition:** For some segment $S = \{(I_i, \sigma_i)\}_{a \le i \le b}$, variable $x \in \bar{x}^T$ and $i, i' \in [a, b]$ where $x \nprec_{\tau_i} \texttt{null}$ and $x \nprec_{\tau_{i'}} \texttt{null}$, $E = [(i, x)]$ and $E' = [(i', x)]$.
- **Life-Cycle-Condition:** For some life cycle $L = \{(I_i, \sigma_i)\}_{i \in J}$, variable $x \in \bar{x}^T_{\text{in}} \cup \bar{s}^T$ and $i, i' \in J$ where $x \nprec_{\tau_i} \texttt{null}$ and $x \nprec_{\tau_{i'}} \texttt{null}$, $E = [(i, x)]$ and $E' = [(i', x)]$.
- **Input-Condition:** For some variable $x \in \bar{x}^T_{\text{in}}$ and $i, i' \in [0, \gamma)$, $E = [(i, x)]$ and $E' = [(i', x)]$.
- **FD-Condition:** For some local expressions $(i, e), (i', e')$ and attribute a where $(i, e) \sim (i', e')$, $E = [(i, e.a)]$ and $E' = [(i', e'.a)]$.

From the global isomorphism type Λ defined above, we construct ρ_T and D_T as follows. The domain of D_T is the set of equivalence classes of $\sim$. Each relation $R(id, a_1, \ldots, a_k)$ in D_T consists of all tuples $([(i, e)], [(i, e.a_1)], \ldots [(i, e.a_k)])$ for which $(i, e), (i, e.a_1), \ldots, (i, e.a_k) \in \mathcal{E}$. Note that the chase step guarantees that for all local expressions $(i, e), (i', e')$, if $(i, e.a), (i', e'.a) \in \mathcal{E}$ and $(i, e) \sim (i', e')$, then $(i, e.a) \sim (i', e'.a)$. It follows that $D_T \models \textbf{FD}$. We next define $\rho_T = (\nu_{in}, \nu_{out}, \{(\rho_i, \sigma_i)\}_{0 \le i < \gamma})$, where $\rho_i = (\nu_i, S_i)$. First, let $\nu_i(x) = [(i, x)]$ for $0 \le i < \gamma$, $\nu_{in} = \nu_0|\bar{x}^T_{in}$, and $\nu_{out} = \perp$ if $\tau_{out} = \perp$ and $\nu_{out} = \nu_{\gamma-1}|\bar{x}^T_{\text{out}}$ otherwise. Suppose that, as will be shown below, properties (i)-(iii) of Lemma 42 hold for D_T and the sequence $\{\nu_i\}_{0 \le i < \gamma}$ so defined. Note that (ii) and (iii) imply that the pre- and-post conditions of all services σ_i hold. Also, by construction, for every variable $x \in \bar{x}^T$ where $\nu_{i-1}(x) = \nu_i(x)$ is required by the transition under σ_i we always have $(i, x) \sim (i + 1, x)$. Consider the sets $\{S_i\}_{0 \le i < \gamma}$. Recall the constraints imposed on sets by the definition of local run: $S_0 = \emptyset$, and for $0 < i < \gamma$ where δ_i is the set update of σ_i,

(1) $S_i = S_{i-1} \cup \nu_{i-1}(\bar{s}^T)$ if $\delta_i = \{+S^T(\bar{s}^T)\}$,
(2) $S_i = S_{i-1} - \nu_i(\bar{s}^T)$ if $\delta_i = \{-S^T(\bar{s}^T)\}$,
(3) $S_i = (S_{i-1} \cup \{\nu_{i-1}(\bar{s}^T)\}) - \{\nu_i(\bar{s}^T)\}$ if $\delta_i = \{+S^T(\bar{s}^T), -S^T(\bar{s}^T)\}$, and
(4) $S_i = S_{i-1}$ if $\delta_i = \emptyset$.

Note that the only cases that can make ρ_T invalid are those for which δ_i contains $-S^T(\bar{s}^T)$. Indeed, while a tuple can always be inserted, a tuple can be retrieved only if it belongs to S^T (or is simultaneously inserted as in case (3)). Thus, in order to show that the specified retrievals are possible, it is sufficient to prove the following.

Lemma 44. *Let $0 < i < \gamma$ be such that (1)-(4) hold for $\{S_j\}_{0 \le j < i}$. If $\delta_i = \{-S^T(\bar{s}^T)\}$ then $\nu_i(\bar{s}^T) \in S_{i-1}$. If $\delta_i = \{+S^T(\bar{s}^T), -S^T(\bar{s}^T)\}$ then either $\nu_i(\bar{s}^T) \in S_{i-1}$ or $\nu_i(\bar{s}^T) = \nu_{i-1}(\bar{s}^T)$.*

Proof. The key observations, which are easily checked by the construction of Λ, are the following:

(†) for every $k, k' \in [0, \gamma)$, if $\hat{\tau}_k$, $\hat{\tau}_{k'}$ are not input-bound and I_k and $I_{k'}$ are not in the same life cycle, then $\nu_k(\bar{s}^T) \neq \nu_{k'}(\bar{s}^T)$.
(‡) for every $k, k' \in [0, \gamma)$, if $\hat{\tau}_k$, $\hat{\tau}_{k'}$ are input-bound, $\nu_k(\bar{s}^T) = \nu_{k'}(\bar{s}^T)$ iff $\hat{\tau}_k = \hat{\tau}_{k'}$.

Now suppose that $0 < i < \gamma$, (1)-(4) hold for $\{S_j\}_{0 \le j < i}$, and $\delta_i = \{-S^T(\bar{s}^T)\}$. Suppose first that $\hat{\tau}_i$ is not input-bound. Let L be the life cycle to which I_i belongs, and $n < i$ be such that $I_n = \texttt{Retrieve}(I_i)$. By (†), $\nu_k(\bar{s}^T) \neq \nu_i(\bar{s}^T)$ for every $n < k < i$. Since (1)-(4) hold for all $j < i$, $\nu_n(\bar{s}^T) \in S_{i-1}$. By construction of Λ (specifically the Life-Cycle chase condition), $\nu_n(\bar{s}^T) = \nu_i(\bar{s}^T)$. Thus, $\nu_i(\bar{s}^T) \in S_{i-1}$. The case when $\delta_i = \{+S^T(\bar{s}^T), -S^T(\bar{s}^T)\}$ is similar.

Now suppose $\hat{\tau}_i$ is input-bound and $\delta_i = \{-S^T(\bar{s}^T)\}$. By definition of symbolic local run, $\bar{c}_{i-1}(\hat{\tau}_i) = 1$. Thus, there must exist a maximum $n < i$ such that $\hat{\tau}_n = \hat{\tau}_i$ and for which the transition under σ_n sets $\bar{c}_n(\hat{\tau}_i) = 1$. Since $\bar{c}_{i-1}(\hat{\tau}_i) = 1$ and n is maximal, there is no j, $n < j < i$ for which δ_j contains $-S^T(\bar{s}^T)$ and $\hat{\tau}_j = \hat{\tau}_i$. From the above and (‡) it easily follows that $\nu_n(\bar{s}^T) = \nu_i(\bar{s}^T)$ and $\nu_i(\bar{s}^T) \in S_{i-1}$. The case when $\delta_i = \{+S^T(\bar{s}^T), -S^T(\bar{s}^T)\}$ is similar. □

It remains to prove properties (i)-(iii) of Lemma 42. First, as $\delta \ge 1$ and $\delta \ge h(T_c)$ for every $T_c \in child(T)$, property (iii) is immediately satisfied. We next prove (i) and (ii).

Proof of property (i). We first introduce some additional notation. For each i and $(i, e) \in \mathcal{E}_i$, we denote by $[(i, e)]_i$ the equivalence class of (i, e) wrt $\sim_i$. And for $x \in \bar{x}^T$ we denote by $\mathsf{Reach}_i(x, w)$ the unique equivalence class of $\sim_i$ reachable from $[(i, x_R)]_i$ by some navigation w (if such class exists). More precisely:

Definition 45. *For each $0 \le i < \gamma$, we define the* navigation graph $G(\sim_i)$ *of the local equality type $\sim_i$ to be the labeled directed graph whose nodes are the equivalence classes of $\sim_i$ and where for each attribute a, there is an edge labeled a from E to F if there exist $e \in E$ and $f \in F$ such that $(i, e.a) \in \mathcal{E}_i$ and $e.a \sim_{\tau_i} f$. Note that for each E there is at most one outgoing edge labeled a. For $x \in \bar{x}^T, x \nsim_i \texttt{null}$ and sequence of attributes w, we denote by $\mathsf{Reach}_i(x, w)$ the unique equivalence class F of $\sim_i$ reachable from $[(i, x)]_i$ by a path in $G(\sim_i)$ whose sequence of edge labels spells w, if such exists, and the empty set otherwise.*

Example 46. *The graph $G(\sim_i)$ of the equality type in Example 30 is shown in Figure 12. In this $G(\sim_i)$, the node $\mathsf{Reach}_i(y, AB)$ is the equivalence class $\{(i, x_R.A.B), (i, y_R.A.B), (i, 0)\}$.*

By our choice of $h(T)$ and our construction of the λ_i's, we can show that

Lemma 47. *For every $0 \le i < \gamma$ and expression $x_R.w$, if $\mathsf{Reach}_i(x, w)$ is non-empty, then there exists an expression $x_R.\tilde{w}$ where $|\tilde{w}| < h(T)$ such that $\mathsf{Reach}_i(x, w) = \mathsf{Reach}_i(x, \tilde{w})$.*

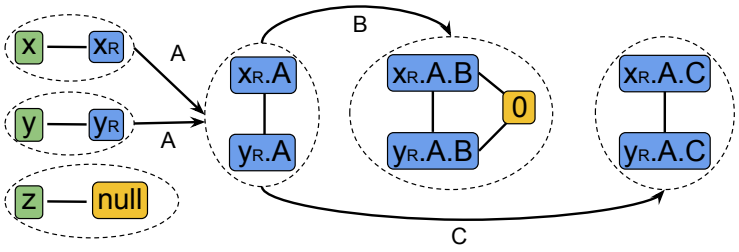

Fig. 12. An illustration of the navigation graph $G(\sim_i)$.

Proof. It is sufficient to show that for each i, $|G(\sim_i)| < h(T)$, where $|G(\sim_i)|$ is the number of nodes in $G(\sim_i)$. Indeed, since there is a path from $[(i, x_R)]_i$ to $\mathsf{Reach}_i(x, w)$ in $G(\sim_i)$, there must exist a simple such path, of length at most $|G(\sim_i)| < h(T)$.

To show that $|G(\sim_i)| < h(T)$, recall that $|G(\sim_i)|$ is bounded by the number of isomorphism types of $\sim_i$. Recall that $h(T) = 1 + |\bar{x}^T| \cdot F(\delta)$ where $F(n)$ is the maximum number of distinct paths of length at most n starting from any relation in the foreign key graph FK. By definition, for each variable x, the number of expressions $\{e | e = x_R.w, (i, e) \in \mathcal{E}_i\}$ is bounded by $F(\delta)$. Thus the number of equivalence classes of $\sim_i$ is at most $|\bar{x}^T| \cdot F(\delta) < h(T)$. So $|G(\sim_i)| < h(T)$. □

Property (i) now follows from Lemma 47. Let $e = x_R.w$ be an expression for which $v_i^*(e)$ is defined. By construction, $\mathsf{Reach}_i(x, w) \subseteq v_i^*(e)$. By Lemma 47, there exists $e' = x_R.w'$ where $|w'| < h(T)$ and $\mathsf{Reach}_i(x, w') = \mathsf{Reach}(x, w)$. It follows that $v_i^*(e')$ is defined and $v_i^*(e) \cap v_i^*(e') \neq \emptyset$. As $v_i^*(e)$ and $v_i^*(e')$ are equivalence classes of $\sim$, we have $v_i^*(e) = v_i^*(e')$, proving (i).

Proof of property (ii). To show property (ii), it is sufficient to show an invariant which implies property (ii) and is satisfied throughout the construction of Λ. For simplicity, we assume that the chase step in the construction of $\sim$ is divided into the following 3 phases.

- The *Segment Phase*. In this phase, we merge equivalence classes E and E' that satisfies either the Segment-Condition or the FD-condition.
- The *Life Cycle Phase*. In this phase, we merge equivalence classes E and E' that satisfies either the Life-Cycle-Condition or the FD-condition.
- The *Input Phase*. In this phase, we merge equivalence classes E and E' that satisfies either the Input-condition or the FD-condition.

It is easily seen that no chase step applies after the input phase. Thus, the above steps compute the complete chase.

For each equivalence class E of $\sim$, we let $i(E)$ be the set of indices $\{i | (i, e) \in E\}$ and for each $i \in i(E)$, we denote by $E|_i$ the projection of E on the navigation set $\mathcal{E}_i$. One can show that during the segment phase, for every E of $\sim$, $i(E)$ are indices within the same segment. During the life cycle phase, for every E of $\sim$, $i(E)$ are indices within the same life cycle. And during the input phase, $i(E)$ can be arbitrary indices.

The invariant is defined as follows.

Lemma 48. *(Invariant of Λ) Throughout the construction of Λ, for every equivalence class E of $\sim$, there exists variable $x \in \bar{x}^T$ and navigation w where $|w| \leq h(T)$, such that for every $i \in i(E)$, $E|_i = \mathsf{Reach}_i(x, w)$.*

Lemma 48 implies that for each equivalence class E of $\sim$ and for each λ_i, E is a superset of at most one equivalence class of λ_i. So $(i, e) \sim (i, e')$ implies $(i, e) \sim_i (i, e')$ thus $\Lambda|\mathcal{E}_i = \lambda_i$ for every $0 \leq i < \gamma$, which implies property (ii) of Lemma 42.

Proof. We consider each step of the construction of the global equality type $\sim$. For the initialization step, the invariant holds by Lemma 47.

For the Chase steps, assume that the invariant is satisfied before merging two equivalence classes E and E'. For each equivalence class E of $\sim$, we denote by $x(E)$ and $w(E)$ the variable and the navigation for E as stated in Lemma 48. To show the invariant is satisfied after merging E and E', it is sufficient to show that there exists variable y and navigation u where $|u| \leq h(T)$ such that for every $i \in i(E)$, $E|_i = \mathsf{Reach}_i(y, u)$ and for every $i \in i(E')$, $E'|_i = \mathsf{Reach}_i(y, u)$.

Consider the segment phase. Suppose first that E and E' are merged due to the Segment-Condition. For simplicity, we let $x = x(E), x' = x(E'), w = w(E)$ and $w' = w(E')$. If $E = [(i, y)]$ and $E' = [(i', y)]$ where i, i' are indices within the same segment S, then by the assumption, we have $(i, y) \in \mathsf{Reach}_i(x, w)$, so $y \sim_{\tau_i} x_R.w$. As $i(E)$ are indices of a segment S, and by Lemma 43, we have that for every $j \in i(E)$, $y \sim_{\tau_j} x_R.w$, so $E|_j = \mathsf{Reach}_j(x, w) = \mathsf{Reach}_j(y, \epsilon)$. Similarly, we can show that for every $j \in i(E')$, $E'|_j = \mathsf{Reach}_j(y, \epsilon)$.

Next suppose E and E' are merged due to the FD-condition. Thus, $E = [(i, e.a)]$ and $E' = [(i', e'.a)]$ where $(i, e) \sim (i', e')$. Let E^* be the equivalence class of $\sim$ that contains (i, e) and (i', e'). By the assumption, for $y = x(E^*)$ and $u = w(E^*)$, we have that $E^*|_i = \mathsf{Reach}_i(y, u)$ so $(i, e) \in \mathsf{Reach}_i(y, u)$. By Lemma 47, there exists navigation $\tilde{u}$ where $|\tilde{u}| < h(T)$ such that $\mathsf{Reach}_i(y, u) = \mathsf{Reach}_i(y, \tilde{u})$. So $(i, e.a) \in \mathsf{Reach}_i(y, \tilde{u}.a)$. Then in E, by the hypothesis, we have $(i, e.a) \in \mathsf{Reach}_i(x, w)$ so $\mathsf{Reach}_i(y, \tilde{u}.a) = \mathsf{Reach}_i(x, w)$. As $i(E)$ are indices of a segment S, and by Lemma 43, we have that for every $j \in i(E)$, for some relation R_1 and R_2, $y_{R_1}.\tilde{u}.a \sim_{\tau_j} x_{R_2}.w$ so $E|_j = \mathsf{Reach}_j(x, w) = \mathsf{Reach}_j(y, \tilde{u}.a)$. Similarly, we can show that for every $j \in i(E')$, $E'|_j = \mathsf{Reach}_j(y, \tilde{u}.a)$. Therefore, the invariant is preserved during the segment phase.

Consider the life cycle phase. We can show that the invariant is again preserved, together with the following additional property: for each equivalence class E of $\sim$ produced in this phase, $x(E) \in \bar{x}^T_{\mathsf{in}} \cup \bar{s}^T$. Suppose E and E' are merged due to the Life-Cycle Condition, where $E = [(i, y)]$, $E' = [(i', y)]$ and $y \in \bar{x}^T_{\mathsf{in}} \cup \bar{s}^T$. We have that $E|_j = \mathsf{Reach}_j(x, w) = \mathsf{Reach}_j(y, \epsilon)$ for every $j \in i(E)$. Indeed, by Lemma 43 and because $i(E)$ are indices of some life cycle L, $x_R.w \sim_{\tau_i} y$ implies that $x_R.w \sim_{\tau_j} y$ for every index j of L. Similarly, $E'|_j = \mathsf{Reach}_j(y, \epsilon)$ for every $j \in i(E')$. The case when E and E' are merged in this stage due to the FD-condition is similar to the above. Following similar analysis, we can show that the input phase also preserves the invariant together with the property that for every E produced at the input phase, $x(E) \in \bar{x}^T_{\mathsf{in}}$. This uses the fact that $\tau_i|\bar{x}^T_{\mathsf{in}} = \tau_{in}$ for every $0 \leq i < \gamma$. □

This completes the proof of Lemma 42.

5.5 Handling infinite local symbolic runs

Next we show that Lemma 42 can be extended to infinite periodic local symbolic runs, which together with finite runs are sufficient to represent accepted symbolic trees of runs by our VASS construction (see Lemma 63). Specifically, we show that we can extend the construction of the global isomorphism type to infinite periodic $\tilde{\rho}_T$, while producing only *finitely* many equivalence classes. This is sufficient to show that the corresponding database D_T is finite. We define periodic local symbolic runs next.

Definition 49. *A local symbolic run $\tilde{\rho}_T = (\tau_{in}, \tau_{out}, \{(I_i, \sigma_i)\}_{0 \leq i < \gamma})$ is periodic if $\gamma = \omega$ and there exists $n > 0$ and $0 < t \leq n$, such that for every $i \geq n$, the symbolic instances $I_i = (\tau_i, \bar{c}_i)$ and*

$I_{i-t} = (\tau_{i-t}, \bar{c}_{i-t})$ satisfy that $(\tau_i, \sigma_i) = (\tau_{i-t}, \sigma_{i-t})$ and $\bar{c}_i \geq \bar{c}_{i-t}$. The integer n and t are called the offset *and* period *of $\tilde{\rho}_T$ respectively.*

The following is a consequence of Lemma 63, proven later in the section.

Corollary 50. *If there exists a symbolic tree of runs **Sym** accepted by $\mathcal{B}_\beta$, then there exists a symbolic tree of runs **Sym′** accepted by $\mathcal{B}_\beta$ such that for every $\tilde{\rho}_T \in$ **Sym**, $\tilde{\rho}_T$ is finite or periodic.*

The above corollary indicates that for verification, it is sufficient to consider only finite and periodic $\tilde{\rho}_T$. So what we need to prove is:

Lemma 51. *For every periodic local symbolic run $\tilde{\rho}_T = (\tau_{in}, \tau_{out}, \{(I_i, \sigma_i)\}_{0 \leq i < \omega})$, there exists a local run $\rho_T = (\nu_{in}, \nu_{out}, \{(\rho_i, \sigma_i)\}_{0 \leq i < \omega})$ on finite database $D_T \models$ **FD** such that for every $i \geq 0$,*

(i) *for every expression $e = x_R.w$ where $\nu_i^*(e)$ is defined, there exists expression $e' = x_R.w'$ where $|w'| \leq h(T)$ such that $\nu_i^*(e) = \nu_i^*(e')$,*
(ii) *for all expressions $e, e' \in \mathcal{E}_T^+$ of τ_i, if $\nu_i^*(e)$ and $\nu_i^*(e')$ are defined, then $e \sim_{\tau_i} e'$ iff $\nu_i^*(e) = \nu_i^*(e')$, and*
(iii) *for $\delta = h(T_c)$ if $\sigma_i \in \{\sigma^o_{T_c}, \sigma^c_{T_c}\}$ for some $T_c \in child(T)$ and $\delta = 1$ otherwise, for every expression $e \in \mathcal{E}_T^+ - \{x_R.w | x \in \bar{x}^T, |w| > \delta\}$, $\nu_i^*(e)$ is defined.*

Intuitively, if we directly apply the construction of ρ_T and D_T from Lemma 42 in the case of finite $\tilde{\rho}_T$, then each life cycle with non-input-bound TS-isomorphism types would be assigned with distinct sets of values, which could lead to an infinite D_T. However, for any two disjoint life cycles L_1 and L_2, reusing the same values in L_1 and L_2 does not cause any conflict. And in particular, if L_1 and L_2 are identical on the sequence of τ_i's and σ_i's, they can share exactly the same set of values.

Thus at a high level, our goal is to show that any periodic local symbolic run $\tilde{\rho}_T$ can be partitioned into finitely many subsets of identical life cycles with disjoint timespans. Unfortunately, this is generally not true if we pick the `Retrieve` function arbitrarily (recall that `Retrieve` defines the set of life cycles). This is because an arbitrary `Retrieve` may yield life cycles whose timespans have unbounded length. If the timespans overlap, it is impossible to separate the life cycles into finitely many subsets of life cycles with disjoint timespans. So instead of picking an arbitrary `Retrieve` as in the finite case, we show that for periodic $\tilde{\rho}_T$ we can construct `Retrieve` such that the timespan of each life cycle has bounded length. This implies that we can partition the life cycles into finitely many subsets of identical life cycles with disjoint timespans, as desired. Finally we show that given the partition, we can construct the local run ρ_T together with a finite D_T.

We first define the equivalence relation between life cycles.

Definition 52. *Segments $S_1 = \{(I_i, \sigma_i)\}_{a_1 \leq i \leq b_1}$ and $S_2 = \{(I_i, \sigma_i)\}_{a_2 \leq i \leq b_2}$ are equivalent, denoted as $S_1 \equiv S_2$, if $\{(\tau_i, \sigma_i)\}_{a_1 \leq i \leq b_1} = \{(\tau_i, \sigma_i)\}_{a_2 \leq i \leq b_2}$.*

Recall that $\mathcal{I}^+$ (and $\mathcal{I}^-$) are the sets of symbolic instances inserting (and retrieving) non-input-bound TS-isomorphism types respectively. We define that

Definition 53. *A segment $S = \{(I_i, \sigma_i)\}_{a \leq i \leq b}$ is* static *if $I_a \in \mathcal{I}^-$, $I_b \in \mathcal{I}^+$ and $\tau_a|\bar{s}^T = \tau_b|\bar{s}^T$. A segment S is called* dynamic *if it is not static.*

When we compare two life cycles L_1 and L_2, we can ignore their static segments since they do not change the content of S^T. We define equivalence of two life cycles as follows.

Definition 54. *For life cycle L, let $dyn(L) = \{S_i\}_{1 \leq i \leq k}$ be the sequence of dynamic segments of L. Two life cycles L_1 and L_2 are equivalent, denoted as $L_1 \equiv L_2$, if $|dyn(L_1)| = |dyn(L_2)|$ and for $dyn(L_1) = \{S_i^1\}_{1 \leq i \leq k}$ and $dyn(L_2) = \{S_i^2\}_{1 \leq i \leq k}$, for every $1 \leq i \leq k$, $S_i^1 \equiv S_i^2$.*

Note that for each life cycle L, the number of dynamic segments within L is bounded by $|\bar{s}^T|$ since within L, each variable in $\bar{s}^T$ is written at most once by returns of child tasks of T. For a task T, as the number of T-isomorphism types is bounded, the number of services is bounded and the length of a segment is bounded because each subtask can be called at most once, the number of equivalence classes of segments is bounded. And since the number of dynamic segments is bounded within the same life cycle, the number of equivalence classes of life cycles is also bounded. Thus,

Lemma 55. *The equivalence relation $\equiv$ on life cycles has finite index.*

Our next step is to show that one can define a `Retrieve` function so that all life cycles have bounded timespans. The timespan of a life cycle is defined as follows:

Definition 56. *The timespan of a life cycle L, denoted by $sp(L)$, is an interval $[a, b]$ where a is the index of the first symbolic instance of the first dynamic segment of L and b is the index of the last symbolic instance of the last dynamic segment.*

Consider an equivalence class $\mathcal{L}$ of life cycles. Suppose that for each $L \in \mathcal{L}$, the length of $sp(L)$ is bounded by some constant m. Then we can further partition $\mathcal{L}$ into m subsets $\mathcal{L}_0, \ldots, \mathcal{L}_{m-1}$ of life cycles with disjoint timespan by assigning each $L \in \mathcal{L}$ where $sp(L) = [a, b]$ to the subset $\mathcal{L}_k$ where $k = a \bmod m$.

We next show how to construct the function `Retrieve`. In particular, we construct a periodic `Retrieve` such that there is a short gap between each pair of inserting and retrieving instances. This is done in several steps, illustrated in Figure 13. Recall that n and t are the offset and the period of $\tilde{\rho}_T$.

(1) Initialize `Retrieve` to be an arbitrary one-to-one mapping with domain $\{I_j | I_j \in \mathcal{I}^-, 0 \le j \le n\}$ such that for every $I_i = \texttt{Retrieve}(I_j)$, $i < j$ and $\hat{\tau}_i = \hat{\tau}_j$ (recall that $\hat{\tau}_i = \tau_i | \bar{x}^T_{\text{in}} \cup \bar{s}^T$).
(2) For every $j \in [n+1, n+t]$, for $j' = j - t$ and for i' being the index where $I_{i'} = \texttt{Retrieve}(I_{j'})$,
   (i) if $i' \in [n-t+1, n]$, then for $i = i' + t$, let $\texttt{Retrieve} \leftarrow \texttt{Retrieve}[I_{j+k\cdot t} \mapsto I_{i+k\cdot t} | k \ge 0]$, otherwise
   (ii) if $i' \in [0, n-t]$, then we pick $i \in [n-t+1, n]$ satisfying that $I_i \in \mathcal{I}^+$, $\hat{\tau}_i = \hat{\tau}_j$ and I_i is currently not in the range of `Retrieve`. Then we let $\texttt{Retrieve} \leftarrow \texttt{Retrieve}[I_{j+k\cdot t} \mapsto I_{i+k\cdot t} | k \ge 0]$.

At step 2 for the case $i' \in [0, n-t]$, the i that we picked always exists for the following reason. For every TS-isomorphism type $\hat{\tau}$, let

- $M^-_{\hat{\tau}}$ be the number of symbolic instances in $\mathcal{I}^-$ with TS-isomorphism type $\hat{\tau}$ and indices in $[n-t+1, n]$ that retrieves from symbolic instances with indices in $[0, n-t]$, and
- $M^+_{\hat{\tau}}$ be the number of symbolic instances in $\mathcal{I}^+$ with TS-isomorphism type $\hat{\tau}$ and indices in $[n-t+1, n]$ that is NOT retrieved by symbolic instances with indices in $[n-t+1, n]$.

We have $M^+_{\hat{\tau}} - M^-_{\hat{\tau}} = \bar{c}_n(\hat{\tau}) - \bar{c}_{n-t}(\hat{\tau}) \ge 0$. So for every $I_{i'} = \texttt{Retrieve}(I_{j'})$ where $j' \in [n-t+1, n]$ and $i' \in [0, n-t]$, we can always find a unique $i \in [n-t+1, n]$ such that $I_i \in \mathcal{E}^+$, $\hat{\tau}_i = \hat{\tau}_{j'} = \hat{\tau}_{i'}$ and I_i is not retrieved by any retrieving instances with indices in $[n-t+1, n]$.

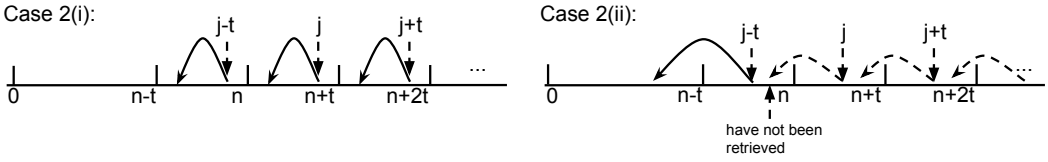

Fig. 13. Construction of `Retrieve`

Let us fix the function `Retrieve` constructed above. We first show the following:

Lemma 57. *For every periodic $\tilde{\rho}_T$, and $j > n$, $I_i = \texttt{Retrieve}(I_j)$ implies that $j - i \leq 2t$ and $I_{i+t} = \texttt{Retrieve}(I_{j+t})$.*

Proof. By construction, for every $I_i = \texttt{Retrieve}(I_j)$ where $j > i > n$, $I_{i+t} = \texttt{Retrieve}(I_{j+t})$. And it is also guaranteed that for the indices i and j, either (1) i and j are both in the same range $[n+tk+1, n+t(k+1)]$ for some $k \geq 0$, or (2) $i \in [n+tk+1, n+t(k+1)]$ and $j \in [n+t(k+1)+1, n+t(k+2)]$ for some $k \geq 0$. In both cases, $j - i \leq 2t$. □

For every life cycle L, for every pair of consecutive dynamic segments S and S', we denote by $gap(S, S')$ the number of static segments in between S and S'. To show that $sp(L)$ is bounded, it is sufficient to show that $gap(S, S')$ is bounded for every pair of consecutive dynamic segments S and S'. For every segment S, we denote by $a(S)$ the index of the first symbolic instance of S. For every segment S where $a(S) > n$, we let $p(S) = (a(S) - n - 1) \bmod t$.

For every pair of consecutive dynamic segments S and S' and by periodicity of $\texttt{Retrieve}$, there are no two static segments T and T' in L in between S and S' such that $a(S) < a(T) < a(T') < a(S')$ and $p(T) = p(T')$. Thus in L, the number of static segments in between S and S' is at most $n + t$. Then by Lemma 57, the number of symbolic instances in between any pair of consecutive segments is bounded by $\max(2t, n)$ so $gap(S, S') \leq (n + t) \cdot \max(2t, n + t)$. And as the number of dynamic segments in L is bounded by $|\bar{s}^T|$ and the length of each segment is at most $2|child(T)|$, it follows that:

Lemma 58. *For every periodic local symbolic run $\tilde{\rho}_T$ and life cycle L of $\tilde{\rho}_T$, $|sp(L)|$ is bounded by $m = (n + t) \cdot \max(2t, n + t) \cdot (|\bar{s}^T| + 1) \cdot 2|child(T)|$.*

So for a possibly infinite set of life cycles $\mathcal{L}$ where $|sp(L)| \leq m$ for each $L \in \mathcal{L}$, $\mathcal{L}$ can be partitioned into sets $\mathcal{L}_0, \ldots, \mathcal{L}_{m-1}$ by assigning each life cycle $L \in \mathcal{L}$ where $sp(L) = [a, b]$ to the set $\mathcal{L}_{a \bmod m}$. So for every $\mathcal{L}_i$ and two distinct L_1, L_2 in $\mathcal{L}_i$ where $sp(L_1) = [a_1, b_1]$ and $sp(L_2) = [a_2, b_2]$, we have $a_1 \neq a_2$. Assume $a_1 < a_2$. Then as $a_1 \equiv a_2 \pmod m$, $a_2 - a_1 \geq m$. And since $b_1 - a_1 + 1 < m$, L_1 and L_2 are disjoint. Thus, given Lemma 55 and Lemma 58, we have

Lemma 59. *Every local symbolic run $\tilde{\rho}_T$ can be partitioned into finitely many subsets of life cycles such that for each subset $\mathcal{L}$, if $L_1 \in \mathcal{L}$, $L_2 \in \mathcal{L}$ and $L_1 \neq L_2$ then $L_1 \equiv L_2$ and $sp(L_1) \cap sp(L_2) = \emptyset$.*

Next, we show how we can construct the local run ρ_T and finite database D_T from $\tilde{\rho}_T$ using the partition. We first construct a global isomorphism type $\Lambda = (\mathcal{E}, \sim)$ of $\tilde{\rho}_T$ using the approach for the finite case. Then we merge equivalent segments in Λ as follows to obtain a new global isomorphism type with finitely many equivalence classes. To merge two equivalent segments $S_1 = \{(I_i, \sigma_i)\}_{a_1 \leq i \leq a_1 + l}$ and $S_2 = \{(I_i, \sigma_i)\}_{a_2 \leq i \leq a_2 + l}$, first for every $0 \leq i \leq l$ and for every $x \in \bar{x}^T$, we merge the equivalence classes $[(a_1 + i, x)]$ and $[(a_2 + i, x)]$ of $\sim$. Then we apply the chase step (i.e. the FD-condition) to make sure the resulting database satisfies **FD**.

The new Λ is constructed as follows. For every two segments $S_1 = \{(I_i, \sigma_i)\}_{a \leq i \leq b}$ and $S_2 = \{(I_i, \sigma_i)\}_{c \leq i \leq d}$, we define that S_1 precedes S_2, denote by $S_1 \prec S_2$, if $b < c$. For each subset $\mathcal{L}$ and for each pair of life cycles $L_1, L_2 \in \mathcal{L}$ where $dyn(L_1) = \{S_i^1\}_{1 \leq i \leq k}$ and $dyn(L_2) = \{S_i^2\}_{1 \leq i \leq k}$,

- for $1 \leq i \leq k$, merge S_i^1 and S_i^2,
- for $1 \leq i < k$, for every static segments $S_1 \subseteq L_1$ and $S_2 \subseteq L_2$ where $S_i^1 \prec S_1 \prec S_{i+1}^1$, $S_i^2 \prec S_2 \prec S_{i+1}^2$ and $S_1 \equiv S_2$, merge S_1 and S_2, and
- for every pair of static segments $S_1 \subseteq L_1$ and $S_2 \subseteq L_2$ where $S_k^1 \prec S_1$, $S_k^2 \prec S_2$ and $S_1 \equiv S_2$, merge S_1 and S_2.

Finally, ρ_T and D_T are constructed following the same approach as in the finite case. In the above construction, as the number of subsets of life cycles is finite, and for each $\mathcal{L}$, the number of dynamic segments is bounded and the number of equivalence classes of static segments is bounded, the number of equivalence classes of Λ is also finite so D_T is finite.

By an analysis similar to the finite case, we can show that ρ_T and D_T satisfy property (i)-(iii) in Lemma 51 and $D_T \models \mathbf{FD}$. In particular, to show property (ii), we can show the same invariant as in Lemma 48, the invariant holds because every pair of merged segments are equivalent.

Finally, to show Lemma 51, it remains to show that ρ_T is a valid local run. Similar to the finite case, it is sufficient to show that

Lemma 60. *For every $i \geq 0$, if $\delta_i = \{-S^T(\bar{s}^T)\}$ then $\nu_i(\bar{s}^T) \in S_{i-1}$. If $\delta_i = \{+S^T(\bar{s}^T), -S^T(\bar{s}^T)\}$ then either $\nu_i(\bar{s}^T) \in S_{i-1}$ or $\nu_i(\bar{s}^T) = \nu_{i-1}(\bar{s}^T)$.*

Proof. The following can be easily checked by the construction of Λ:

(i) for every pair of distinct life cycles L and L' where $sp(L) \cap sp(L') \neq \emptyset$, for every $I_k \in L$ and $I_{k'} \in L'$, if $\hat{\tau}_k$, $\hat{\tau}_{k'}$ are not input-bound then $\nu_k(\bar{s}^T) \neq \nu_{k'}(\bar{s}^T)$, and
(ii) for every pair of life cycles L and L' where $sp(L) \cap sp(L') = \emptyset$, if $I_i, I_j \in L$, $I_j = \texttt{Retrieve}(I_i)$, $\hat{\tau}_i$ is not input-bound, $I_k \in L'$ for $j < k < i$ and $\nu_k(\bar{s}^T) = \nu_i(\bar{s}^T) = \nu_j(\bar{s}^T)$, then I_k is contained in a static segment of L'.
(iii) for every $k, k' \geq 0$, if $\hat{\tau}_k$, $\hat{\tau}_{k'}$ are input-bound, $\nu_k(\bar{s}^T) = \nu_{k'}(\bar{s}^T)$ iff $\hat{\tau}_k = \hat{\tau}_{k'}$.

Consider the case when $\delta_i = \{-S^T(\bar{s}^T)\}$ and $\hat{\tau}_i$ is not input-bound. Let $I_j = \texttt{Retrieve}(I_i)$ and L be the life cycle that contains I_i. Consider I_k where $j < k < i$ and let L' be the life cycle containing I_k. If $sp(L) \cap sp(L') \neq \emptyset$, by (i), $\nu_i(\bar{s}^T) \neq \nu_k(\bar{s}^T)$. If $sp(L) \cap sp(L') = \emptyset$, by (ii), the segment containing I_k is static, so it does not change S^T. Thus, for every segment S between I_j and I_i, the tuple $\nu_i(\bar{s}^T)$ remains in S^T after S. So $\nu_i(\bar{s}^T) \in S_{i-1}$. The case when $\delta_i = \{-S^T(\bar{s}^T), +S^T(\bar{s}^T)\}$ is similar.

The proof for the case when $\hat{\tau}_i$ is input-bound is the same as the proof for Lemma 44. □

This completes the proof of Lemma 51.

5.6 Handling symbolic trees of runs

Finally, we show Lemma 40 by providing a recursive construction of a tree of runs ***Tree*** and database D from any symbolic tree of runs ***Sym*** where all local symbolic runs are either finite or periodic, using Lemmas 42 and 51. Intuitively, the construction simply applies the two lemmas to each node $\tilde{\rho}_T$ of ***Sym*** to obtain a local run ρ_T with a local database D_T. Then the local runs and databases are combined into a tree of local runs recursively by renaming the values in each ρ_T and D_T in a bottom-up manner, reflecting the communication among local runs via input and return variables.

Formally, we first define recursively the construction function F where $F(\boldsymbol{Sym}_T) = (\boldsymbol{Tree}_T, D_T)$ where $\boldsymbol{Sym}_T$ is a subtree of ***Sym*** and $(\boldsymbol{Tree}_T, D_T)$ are the resulting subtree of local runs and database instance. F is defined as follows.

If T is a leaf task, then $\boldsymbol{Sym}_T$ contains a single local symbolic run $\tilde{\rho}_T$. We define that $F(\boldsymbol{Sym}_T) = F(\tilde{\rho}_T) = (\rho_T, D_T)$ where ρ_T and D_T are the local run and database instance shown to exist in Lemmas 42 and 51 corresponding to $\tilde{\rho}_T$.

If T is a non-leaf task where the root of $\boldsymbol{Sym}_T$ is $\tilde{\rho}_T = (\tau_{in}, \tau_{out}, \{(I_i, \sigma_i)\}_{0 \leq i < \gamma})$, then we first let $(\rho_T, D_{\texttt{root}}) = F(\tilde{\rho}_T)$. Next, let $J = \{i | \sigma_i = \sigma^o_{T_c}, T_c \in child(T)\}$. For every $i \in J$, we denote by $\boldsymbol{Sym}_i$

the subtree rooted at the child of $\tilde{\rho}_T$ where the edge connecting it with $\tilde{\rho}_T$ is labeled i and let $\tilde{\rho}_i$ be the root of $\boldsymbol{Sym}_i$. We denote by $(\boldsymbol{Tree}_i, D_i) = F(\boldsymbol{Sym}_i)$ and by ρ_i the local run at the root of $\boldsymbol{Tree}_i$. From the construction in Lemmas 42 and 51, we assume that the domains of $D_{\texttt{root}}$ and the D_i's are equivalence classes of local expressions. We first define the renaming function r whose domain is $\bigcup_{i\in J} \texttt{adom}(D_i)$ as follows.

(1) Initialize r to be the identity function.
(2) For every $i \in J$, for every expression $x_R.w$ where $x \in \bar{x}_{\text{in}}^{T_c}$ and $v^*_{in}(x_R.w)$ is defined, for $y = f_{in}(x)$, let $r \leftarrow r[v^*_{in}(x_R.w) \mapsto v^*_i(y_R.w)]$. Note that v^*_{in} is defined wrt v_{in} of ρ_i and D_i and v^*_i is defined wrt v_i of ρ_T and $D_{\texttt{root}}$. And we shall see next that for every such $x_R.w$, if $v^*_{in}(x_R.w)$ is defined, then $v^*_i(y_R.w)$ is also defined.
(3) For every $i \in J$ where $\tilde{\rho}_i$ is a returning local symbolic run where the index of the corresponding $\sigma^c_{T_c}$ in $\tilde{\rho}_T$ is j, for every expression $x_R.w$ where $x \in \bar{x}_{\text{out}}^{T_c}$ and $v^*_{out}(x_R.w)$ is defined, for $y = f_{out}(x)$, let $r \leftarrow r[v^*_{out}(x_R.w) \mapsto v^*_j(y_R.w)]$.

We denote by $r(D)$ the database instance obtained by replacing each value $v \in dom(r)$ in D with $r(v)$ and denote by $r(\boldsymbol{Tree})$ the tree of runs obtained by replacing each value $v \in dom(r)$ in $\boldsymbol{Tree}$ with $r(v)$.

Then if $\tilde{\rho}_T$ is finite, we define $F(\boldsymbol{Sym}_T) = (\boldsymbol{Tree}_T, D_T)$ where $D_T = D_{\texttt{root}} \cup \bigcup_{i\in J} r(D_i)$ and $\boldsymbol{Tree}_T$ is obtained from $\boldsymbol{Sym}_T$ by replacing the root of $\boldsymbol{Sym}_T$ with ρ_T and each subtree $\boldsymbol{Sym}_i$ with $r(\boldsymbol{Tree}_i)$.

If $\tilde{\rho}_T$ is periodic where the period is t and the loop starts with index n, we define $F(\boldsymbol{Sym}_T) = (\boldsymbol{Tree}_T, D_T)$ where $D_T = D_{\texttt{root}} \cup \bigcup_{i\in J, i<n} r(D_i)$ and $\boldsymbol{Tree}_T$ is obtained from $\boldsymbol{Sym}_T$ by replacing the root of $\boldsymbol{Sym}_T$ with ρ_T and each subtree $\boldsymbol{Sym}_i$ with $r(\boldsymbol{Tree}_{i'})$, where $i' = i$ if $i < n$ otherwise $i' = n + (i - n) \bmod t$.

To prove the correctness of the construction, we first need to show that for every $\boldsymbol{Sym}_T$ and $(\boldsymbol{Tree}_T, D_T) = F(\boldsymbol{Sym}_T)$, D_T is a finite database satisfying **FD** and $\boldsymbol{Tree}_T$ is a valid tree of runs over D_T. Let $\tilde{\rho}_T$ and ρ_T be the root of $\boldsymbol{Sym}_T$ and $\boldsymbol{Tree}_T$ respectively. We show the following:

Lemma 61. *For every symbolic tree of runs $\boldsymbol{Sym}_T$ where $(\boldsymbol{Tree}_T, D_T) = F(\boldsymbol{Sym}_T)$, D_T is a finite database satisfying* **FD**, *$\boldsymbol{Tree}_T$ is a valid tree of runs over D_T, and (ρ_T, D_T) satisfies properties (i)-(iii) in Lemma 42 and 51.*

Proof. We use a simple induction. For the base case, where T is a leaf task, the lemma holds trivially. For the induction step, assume that for each $i \in J$, D_i is finite and satisfies **FD**, $\boldsymbol{Tree}_i$ is a valid tree of runs over D_i, and (ρ_i, D_i) satisfies property (i)-(iii).

For each $i \in J$, where $\tilde{\rho}_i$ is a local symbolic run of task $T_c \in child(T)$, we first consider the connection between $\tilde{\rho}_i$ and $\tilde{\rho}_T$ via input variables. As ρ_i satisfies properties (i) and (ii), for every expressions $x_R.w$ and $x'_{R'}.w'$ in the input isomorphism type τ_{in} of $\tilde{\rho}_i$, if $v^*_{in}(x_R.w)$ and $v^*_{in}(x'_{R'}.w')$ are defined, then $v^*_{in}(x_R.w) = v^*_{in}(x'_{R'}.w')$ iff $x_R.w \sim_{\tau_{in}} x'_{R'}.w'$. And by definition of symbolic tree of runs, we have that $\tau_{in} = f^{-1}_{in}(\tau_i)|(\bar{x}_{\text{in}}^{T_c}, h(T_c))$. So for $y = f_{in}(x)$ and $y' = f_{in}(x')$, $v^*_{in}(x_R.w) = v^*_{in}(x'_{R'}.w')$ iff $y_R.w \sim_{\tau_i} y'_{R'}.w'$. Then as ρ_T satisfies (ii) and (iii), $v^*_i(y_R.w)$ and $v^*_i(y'_{R'}.w')$ are defined and $v^*_i(y_R.w) = v^*_i(y'_{R'}.w')$ iff $y_R.w \sim_{\tau_i} y'_{R'}.w'$ so $v^*_i(y_R.w) = v^*_i(y'_{R'}.w')$ iff $v^*_{in}(x_R.w) = v^*_{in}(x'_{R'}.w')$.

If $\tilde{\rho}_i$ is returning, using the same argument as above, we can show the following. Let j be the index of the corresponding returning service $\sigma^c_{T_c}$. Let f be the function where $f(x) = \begin{cases} f_{in}(x), x \in \bar{x}_{\text{in}}^{T_c} \\ f_{out}(x), x \in \bar{x}_{\text{out}}^{T_c} \end{cases}$

and let ν be the valuation where $\nu(x) = \begin{cases} \nu_{in}(x), x \in \bar{x}_{\text{in}}^{T_c} \\ \nu_{out}(x), x \in \bar{x}_{\text{out}}^{T_c} \end{cases}$, where ν_{in} and ν_{out} are the input and output valuation of ρ_i. For all expressions $x_R.w$ and $x'_{R'}.w'$ where $x, x' \in \bar{x}_{\text{out}}^{T_c} \cup \bar{x}_{\text{in}}^{T_c}$, if $\nu^*(x_R.w)$ and $\nu^*(x'_{R'}.w')$ are defined, then for $y = f(x)$ and $y' = f(x')$, $\nu_j^*(y_R.w)$ and $\nu_j^*(y'_{R'}.w')$ are also defined and $\nu_j^*(y_R.w) = \nu_j^*(y'_{R'}.w')$ iff $\nu^*(x_R.w) = \nu^*(x'_{R'}.w')$.

Given this, after renaming, $D_{\texttt{root}}$ and $r(D_i)$ can be combined consistently. Also, one can easily check that $\boldsymbol{Tree}_T$ is a valid tree of runs where (ρ_T, D_T) satisfies properties (i)-(iii) and $D_T \models \mathbf{FD}$. And D_T is a finite database because it is the union of $D_{\texttt{root}}$ and finitely many $r(D_i)$'s and by the hypothesis, $D_{\texttt{root}}$ and the D_i's are finite. □

Finally, to complete the proof of correctness of the construction, we note:

Lemma 62. *For every full symbolic tree of runs **Sym** where all local symbolic runs in **Sym** are either finite or periodic, for $(\boldsymbol{Tree}, D) = F(\boldsymbol{Sym})$ and every HLTL-FO property φ_f, **Sym** is accepted by $\mathcal{B}_\varphi$ iff **Tree** is accepted by $\mathcal{B}_\varphi$ on D.*

The above follows immediately from the fact that by construction, for every task T and local symbolic run $\tilde{\rho}_T = (\tau_{in}, \tau_{out}, \{(I_i, \sigma_i)\}_{0 \le i < \gamma})$ in $\boldsymbol{Sym}$ where the corresponding local run in $\boldsymbol{Tree}$ is $\rho_T = (\nu_{in}, \nu_{out}, \{(\rho_i, \sigma_i)\}_{0 \le i < \gamma})$, for every condition π over $\bar{x}^T$ and $0 \le i < \gamma$, $\tau_i \models \pi$ iff $D \models \pi(\nu_i)$.

This completes the proof of Lemma 40, and the only-if part of Theorem 37.

5.7 Symbolic Verification

In view of Theorem 37, we can now focus on the problem of checking the existence of a symbolic tree of runs satisfying a given HLTL-FO property. To begin, we define a notion that captures the functionality of each task and allows a modular approach to the verification algorithm. Let φ_f be an HLTL-FO formula over Γ, and recall the automaton $\mathcal{B}_\varphi$ and associated notation from Section 3. We consider the relation $\mathcal{R}_T$ between input and outputs of each task, defined by its symbolic runs that satisfy a given truth assignment β to the formulas in Φ_T. More specifically, we denote by $\mathcal{H}_T$ the restriction of $\mathcal{H}$ to T and its descendants, and Γ_T the corresponding HAS, with precondition *true*. The relation $\mathcal{R}_T$ consists of the set of triples $(\tau_{in}, \tau_{out}, \beta)$ for which there exists a symbolic tree of runs $\boldsymbol{Sym}_T$ of $\mathcal{H}_T$ such that:

- β is a truth assignment to Φ_T
- $\boldsymbol{Sym}_T$ is accepted by $\mathcal{B}_\beta$
- the root of $\boldsymbol{Sym}_T$ is $\tilde{\rho}_T = (\tau_{in}, \tau_{out}, \{(I_i, \sigma_i)\}_{0 \le i < \gamma})$

Note that there exists a symbolic tree of runs $\boldsymbol{Sym}$ over Γ satisfying φ_f iff $(\tau_{in}, \bot, \beta) \in \mathcal{R}_{T_1}$ for some τ_{in} satisfying the precondition of Γ, and $\beta(\varphi_f) = 1$. Thus, if $\mathcal{R}_T$ is computable for every T, then satisfiability of φ_f by some symbolic tree of runs over Γ is decidable, and yields an algorithm for model-checking HLTL-FO properties of HAS's.

We next describe an algorithm that computes the relations $\mathcal{R}_T(\tau_{in}, \tau_{out}, \beta)$ recursively. The algorithm uses as a key tool Vector Addition Systems with States (VASS) [16, 37], which we review next.

A VASS $\mathcal{V}$ is a pair (Q, A) where Q is a finite set of *states* and A is a finite set of *actions* of the form $(p, \bar{a}, q)$ where $\bar{a} \in \mathbb{Z}^d$ for some fixed $d > 0$, and $p, q \in Q$. A run of $\mathcal{V} = (Q, A)$ is a finite sequence $(q_0, \bar{z}_0) \ldots (q_n, \bar{z}_n)$ where $\bar{z}_0 = \bar{0}$ and for each $i \ge 0$, $q_i \in Q$, $\bar{z}_i \in \mathbb{N}^d$, and $(q_i, \bar{a}, q_{i+1}) \in A$ for some $\bar{a}$ such that $\bar{z}_{i+1} = \bar{z}_i + \bar{a}$. We will use the following decision problems related to VASS.

- *State Reachability*: For given states $q_0, q_f \in Q$, is there a run $(q_0, \bar{z}_0) \dots (q_n, \bar{z}_n)$ of $\mathcal{V}$ such that $q_n = q_f$?
- *State Repeated Reachability*: For given states $q_0, q_f \in Q$, is there a run $(q_0, \bar{z}_0) \dots (q_m, \bar{z}_m) \dots (q_n, \bar{z}_n)$ of $\mathcal{V}$ such that $q_m = q_n = q_f$ and $\bar{z}_m \leq \bar{z}_n$?

Both problems are known to be EXPSPACE-complete [37, 46, 55]. In particular, [37] shows that for a n-states, d-dimensional VASS where every dimension of each action has constant size, the state repeated reachability problem can be solved in $O((\log n)2^{c \cdot d \log d})$ non-deterministic space for some constant c. The state reachability problem has the same complexity.

VASS construction. Let T be a task, and suppose that relations $\mathcal{R}_{T_c}$ have been computed for all children T_c of T. We show how to compute $\mathcal{R}_T$ using an associated VASS. For each truth assignment β of Φ_T, we construct a VASS $\mathcal{V}(T, \beta) = (Q, A)$ as follows. The states in Q are all tuples $(\tau, \sigma, q, \bar{o}, \bar{c}_{ib})$ where τ is a T-isomorphism type, σ a service, q a state of $B(T, \beta)$, and $\bar{c}_{ib}$ a mapping from $TS_{ib}(T)$ to $\{0, 1\}$. The vector $\bar{o}$ indicates the current stage of each child T_c of T (`init`, `active` or `closed`) and also specifies the outputs of T_c (an isomorphism type or $\perp$). That is, $\bar{o}$ is a partial mapping associating to some of the children T_c of T the value $\perp$, a T_c-isomorphism type projected to $\bar{x}^{T_c}_{\text{in}} \cup \bar{x}^{T_c}_{\text{out}}$ or the value `closed`. Intuitively, $T_c \notin dom(\bar{o})$ means that T_c is in the `init` state, and $\bar{o}(T_c) = \perp$ indicates that T_c has been called but will not return. If $\bar{o}(T_c)$ is an isomorphism type τ, this indicates that T_c has been called, has not yet returned, and will return the isomorphism type τ. When T_c returns, $\bar{o}(T_c)$ is set to `closed`, and T_c cannot be called again before an internal service of T is applied.

The set of actions A consists of all triples $(\alpha, \bar{a}, \alpha')$ where $\alpha = (\tau, \sigma, q, \bar{o}, \bar{c}_{ib}), \alpha' = (\tau', \sigma', q', \bar{o}', \bar{c}'_{ib})$, δ' is the update of σ', and the following hold:

- τ' is a successor of τ by applying service σ';
- $\bar{a} = \bar{a}(\delta', \hat{\tau}, \hat{\tau}', \bar{c}_{ib})$ (defined in Section 5.1), where $\hat{\tau} = \tau | (\bar{x}^T_{\text{in}} \cup \bar{s}^T)$ and $\hat{\tau}' = \tau' | (\bar{x}^T_{\text{in}} \cup \bar{s}^T)$
- $\bar{c}'_{ib} = \bar{c}_{ib} + \bar{a}$
- if σ' is an internal service, $dom(\bar{o}') = \emptyset$.
- If $\sigma' = \sigma^o_{T_c}$, then $T_c \notin dom(\bar{o})$ and for $\tau^{T_c}_{in} = f^{-1}_{in}(\tau | (\bar{x}^T_{T_c\downarrow}, h(T_c)))$, for some output $\tau^{T_c}_{out}$ of T_c and truth assignment β^{T_c} to Φ_{T_c}, tuple $(\tau^{T_c}_{in}, \tau^{T_c}_{out}, \beta^{T_c})$ is in $\mathcal{R}_{T_c}$. Note that $\tau^{T_c}_{out}$ can be $\perp$, which indicates that this call to T_c does not return. Also, $\bar{o}' = \bar{o}[T_c \mapsto \tau^{T_c}_{out}]$.
- If $\sigma' = \sigma^c_{T_c}$, then $\bar{o}(T_c) = (f^{-1}_{out} \circ f^{-1}_{in})(\tau' | (\bar{x}^T_{T_c\downarrow} \cup \bar{x}^T_{T_c\uparrow}, h(T_c)))$ and $\bar{o}' = \bar{o}[T_c \mapsto \text{closed}]$.
- q' is a successor of q in $B(T, \beta)$ by evaluating Φ_T using (τ', σ'). If $\sigma' = \sigma^o_{T_c}$, formulas in Φ_{T_c} are assigned the truth values defined by β^{T_c}.

An *initial* state of $\mathcal{V}(T, \beta)$ is a state of the form $v_0 = (\tau_0, \sigma_0, q_0, \bar{o}_0, \bar{c}^0_{ib})$ where τ_0 is an initial T-isomorphism type (i.e., for every $x \in \bar{x}^T_{id} - \bar{x}^T_{in}$, $x \sim_{\tau_0} \text{null}$, and for every $x \in \bar{x}^T_{\mathbb{R}} - \bar{x}^T_{in}$, $x \sim_{\tau_0} 0$), $\sigma_0 = \sigma^o_T$, q_0 is the successor of some initial state of $B(T, \beta)$ under (τ_0, σ_0), $dom(\bar{o}_0) = \emptyset$, and $\bar{c}^0_{ib} = \bar{0}$.

Computing $\mathcal{R}_T(\tau_{in}, \tau_{out}, \beta)$ from $\mathcal{V}(T, \beta)$. Checking whether $(\tau_{in}, \tau_{out}, \beta)$ is in $\mathcal{R}_T$ can be done using a (repeated) reachability test on $\mathcal{V}(T, \beta)$, as stated in the following key lemma.

LEMMA 63. *$(\tau_{in}, \tau_{out}, \beta) \in \mathcal{R}_T$ iff there exists an initial state $v_0 = (\tau_0, \sigma_0, q_0, \bar{o}_0, \bar{c}^0_{ib})$ of $\mathcal{V}(T, \beta)$ for which $\tau_0 | \bar{x}^T_{\text{in}} = \tau_{in}$ and the following hold:*

- *If $\tau_{out} \neq \perp$, then there exists a state $v_n = (\tau_n, \sigma_n, q_n, \bar{o}_n, \bar{c}^n_{ib})$ where $\tau_{out} = \tau_n | (\bar{x}^T_{\text{in}} \cup \bar{x}^T_{\text{out}})$, $\sigma_n = \sigma^c_T$, $q_n \in Q^{fin}$ where Q^{fin} is the set of accepting states of $B(T, \beta)$ for finite runs, such that v_n is reachable from v_0. A path from $(v_0, \bar{0})$ to $(v_n, \bar{z}_n)$ is called a **returning path**.*
- *If $\tau_{out} = \perp$, then one of the following holds:*

- *there exists a state* $v_n = (\tau_n, \sigma_n, q_n, \bar{o}_n, \bar{c}^n_{ib})$ *in which* $q_n \in Q^{inf}$ *where* Q^{inf} *is the set of accepting states of* $B(T, \beta)$ *for infinite runs, such that* v_n *is repeatedly reachable from* v_0. *A path* $(v_0, \bar{0}) \dots (v_n, \bar{z}_n) \dots (v_n, \bar{z}'_n)$ *where* $\bar{z}_n \leq \bar{z}'_n$ *is called a* ***lasso path***.
- *there exists state* $v_n = (\tau_n, \sigma_n, q_n, \bar{o}_n, \bar{c}^n_{ib})$ *in which* $\bar{o}_n(T_c) = \perp$ *for some child* T_c *of* T *and* $q_n \in Q^{fin}$, *such that* v_n *is reachable from* v_0. *The path from* $(v_0, \bar{0})$ *to* $(v_n, \bar{z}_n)$ *is called a* ***blocking path***.

The proof of Lemma 63 is by induction on the task hierarchy $\mathcal{H}$.

Proof. **Base Case.** Consider $\mathcal{R}_T(\tau_{in}, \tau_{out}, \beta)$ where T is a leaf task. As T has no subtask, $dom(\bar{o})$ is always empty so $\bar{o}$ can be ignored. Note that, by definition, there can be no blocking path of $\mathcal{V}(T, \beta)$.

For the *if* part, consider $(\tau_{in}, \tau_{out}, \beta) \in \mathcal{R}_T$. Suppose first that $\tau_{out} \neq \perp$. By definition, there exists a finite local symbolic run $(\tau_{in}, \tau_{out}, \{(I_i, \sigma_i)\}_{0 \leq i < \gamma})$ accepted by $B(T, \beta)$, where $\gamma \in \mathbb{N}$ and $\sigma_{\gamma-1} = \sigma^c_T$. Consider an accepting computation $\{q_i\}_{0 \leq i < \gamma}$ of $B(T, \eta)$ on $\{(I_i, \sigma_i)\}_{0 \leq i < \gamma}$, such that $q_{\gamma-1} \in Q_{fin}$. We can construct a returning path $P = \{(p_i, \bar{z}_i)\}_{0 \leq i < \gamma}$ of $\mathcal{V}(T, \beta)$ where for each state $p_i = (\tau_i, \sigma_i, q_i, \bar{o}_i, \bar{c}^i_{ib})$, (τ_i, σ_i, q_i) is obtained directly from $\{(I_i, \sigma_i)\}_{0 \leq i < \gamma}$ and $\{q_i\}_{0 \leq i < \gamma}$, $\bar{z}_i = \bar{c}_i$, and $\bar{c}^i_{ib}$ is the projection of $\bar{c}_i$ to input-bound TS-isomorphism types.

Now suppose $\tau_{out} = \perp$. By definition, and since T is a leaf task, there exists an infinite symbolic run $(\tau_{in}, \tau_{out}, \{(I_i, \sigma_i)\}_{0 \leq i < \omega})$ accepted by $B(T, \beta)$. Consider the sequence $\{q_i\}_{0 \leq i < \omega}$ of states in an accepting computation of $B(T, \eta)$ on $\{(I_i, \sigma_i)\}_{0 \leq i < \omega}$. There must exist $q_f \in Q_{inf}$ such that for infinitely many i, $q_i = q_f$. So we can construct a path $P = \{(p_i, \bar{z}_i)\}_{0 \leq i < \omega}$ of $\mathcal{V}(T, \beta)$ where for each state $p_i = (\tau_i, \sigma_i, q_i, \bar{o}_i, \bar{c}^i_{ib})$ is obtained in the same way as in the case where $\tau_{out} \neq \perp$. By the Dickson's lemma [29], there exists two distinct indices m and n such that $m < n$, $(\tau_m, \sigma_m, q_m, \bar{c}^m_{ib}) = (\tau_n, \sigma_n, q_n, \bar{c}^n_{ib})$, $q_m = q_n = q_f$ and $\bar{z}_m \leq \bar{z}_n$. Thus, the sequence $(p_0, \bar{z}_0), \dots, (p_m, \bar{z}_m), \dots, (p_n, \bar{z}_n)$ is a lasso path of $\mathcal{V}(T, \beta)$.

For the *only-if* direction, if there exists a returning path in $\mathcal{V}(T, \beta)$, then by definition, τ_{in} and τ_{out} together with the sequence $\{(I_i, \sigma_i)\}_{0 \leq i \leq n}$ where each (I_i, σ_i) is obtained directly from $(p_i, \bar{z}_i)$ is a valid local symbolic run $\tilde{\rho}_T$. And $\tilde{\rho}_T$ is accepted by $B(T, \beta)$ since q_n is in Q^{fin}. If there exists a lasso path in $\mathcal{V}(T, \beta)$, then we can obtain a finite sequence $\{(I_i, \sigma_i)\}_{0 \leq i \leq n}$ similar to above. And we can construct $\{(I_i, \sigma_i)\}_{0 \leq i < \omega}$ by repeating the subsequence from index $m + 1$ to index n infinitely many times. As $q_n = q_f \in Q^{inf}$, $(\tau_{in}, \perp, \{(I_i, \sigma_i)\}_{0 \leq i < \omega})$ is an infinite local symbolic run accepted by $B(T, \beta)$, so $(\tau_{in}, \perp, \beta) \in \mathcal{R}_T$.

Induction. Consider a non-leaf task T, and suppose the statement is true for all its children tasks.

For the *if* part, suppose $(\tau_{in}, \tau_{out}, \beta) \in \mathcal{R}_T$. Then there exists an adorned symbolic tree of runs $\boldsymbol{Sym}_T$ with root $\tilde{\rho}_T = (\tau_{in}, \tau_{out}, \{(I_i, \sigma_i)\}_{0 \leq i < \gamma})$ accepted by $\mathcal{B}_{\tilde{\beta}}$. We construct a path $P = \{(p_i, \bar{z}_i)\}_{0 \leq i < \gamma}$ of $\mathcal{V}(T, \beta)$ as follows. The transitions in $\tilde{\rho}_T$ caused by internal services are treated as in the base case. Suppose that $\sigma_i = \sigma^o_{T_c}$ for some child T_c of T. Then there is an edge labeled (i, β^{T_c}) from $\tilde{\rho}_T$ to a symbolic tree of runs accepted by $\mathcal{B}_{\tilde{\beta}^{T_c}}$, rooted at a run $\tilde{\rho}_{T_c}$ of T_c with input $\tau^{T_c}_{in}$ and output $\tau^{T_c}_{out}$. Thus, $(\tau^{T_c}_{in}, \tau^{T_c}_{out}, \beta^{T_c}) \in \mathcal{R}_{T_c}$ and $\mathcal{V}(T, \beta)$ can make the transition from $(p_{i-1}, \bar{z}_{i-1})$ to $(p_i, \bar{z}_i)$ as in its definition (including the updates to $\bar{o}$). If $\tau^{T_c}_{out} \neq \perp$ then there exists a minimum $j > i$ for which $\sigma_j = \sigma^c_{T_c}$ and once again $\mathcal{V}(T, \beta)$ can make the transition from $(p_{j-1}, \bar{z}_{j-1})$ to $(p_j, \bar{z}_j)$ as in its definition, mimicking the return of T_c using the isomorphism type $\tau^{T_c}_{out}$ stored in $\bar{o}(T_c)$. Now consider the resulting path $P = \{(p_i, \bar{z}_i)\}_{0 \leq i < \gamma}$. By applying a similar analysis as in the base

case, if $\gamma \neq \omega$ and $\tau_{out} \neq \bot$, then P is a returning path. If $\gamma \neq \omega$ and $\tau_{out} = \bot$, then P is a blocking path. If $\gamma = \omega$, then there exists a prefix P' of P such that P' is a lasso path.

For the *only-if* direction, let P be a path of $\mathcal{V}(T, \beta)$, starting from a state $p_0 = (\tau_0, \sigma_0, q_0, \bar{o}_0, \bar{c}^0_{ib})$ where $\tau_0|\bar{x}^T_{\text{in}} = \tau_{in}$. If P is a returning path, let $v_n = (\tau_n, \sigma_n, q_n, \bar{o}_n, \bar{c}^n_{ib})$ be its last state and $\tau_{out} = \tau_n|(\bar{x}^T_{\text{in}} \cup \bar{x}^T_{\text{out}})$. If P is not a returning path, then $\tau_{out} = \bot$. From P we can construct a adorned symbolic tree of runs $\boldsymbol{Sym}_T$ accepted by $\mathcal{B}_{\bar{\beta}}$ as follows. The root of $\boldsymbol{Sym}_T$ is a local symbolic run $\tilde{\rho}_T$ constructed analogously to the construction in the only-if direction in the base case. Then for each $\sigma_i = \sigma^o_{T_c}$, by the induction hypothesis, there exists a symbolic tree of runs $\boldsymbol{Sym}_{T_c}$ whose root has input isomorphism type $\tau^{T_c}_{in}$, output isomorphism type $\tau^{T_c}_{out}$ and is accepted by $\mathcal{B}_{\beta^{T_c}}$ (note that $\tau^{T_c}_{in}$, $\tau^{T_c}_{out}$ and β^{T_c} are uniquely defined by P and i). We connect $\boldsymbol{Sym}_T$ with $\boldsymbol{Sym}_{T_c}$ with an edge labeled (i, β^{T_c}).

If P is a returning or blocking path, then $\boldsymbol{Sym}_T$ is accepted by $\mathcal{B}_{\bar{\beta}}$. If P is a lasso path, then we first modify the root $\tilde{\rho}_T$ of $\boldsymbol{Sym}_T$ by repeating the subsequence from $m+1$ to n infinitely, then for each integer i such that $m+1 \leq i \leq n$ and $\boldsymbol{Sym}_T$ is connected with some $\boldsymbol{Sym}_{T_c}$ with edge labeled index (i, β^{T_c}), for each repetition $I_{i'}$ of symbolic instance I_i, we make a copy of $\boldsymbol{Sym}_{T_c}$ and connect $\boldsymbol{Sym}_T$ with $\boldsymbol{Sym}_{T_c}$ with edge labeled (i', β^{T_c}). The resulting $\boldsymbol{Sym}_T$ is accepted by $\mathcal{B}_{\bar{\beta}}$. Thus, $(\tau_{in}, \tau_{out}, \beta) \in \mathcal{R}_T$. □

Complexity of verification. We now have all ingredients in place for our verification algorithm. Let Γ be a HAS and φ_f an HLTL-FO formula over Γ. In view of the previous development, $\Gamma \models \varphi_f$ iff $\neg\varphi_f$ is **not** satisfiable by a symbolic tree of runs of Γ. We outline a non-deterministic algorithm for checking satisfiability of $\neg\varphi_f$, and establish its space complexity $O(f)$, where f is a function of the relevant parameters. The space complexity of verification (the complement) is then upper bounded by $O(f^2)$ by Savitch's theorem [57].

Recall that φ_f is satisfiable by a symbolic tree of runs of Γ iff $(\tau_{in}, \bot, \beta) \in \mathcal{R}_{T_1}$ for some τ_{in} satisfying the precondition of Γ, and $\beta(\neg\varphi_f) = 1$. By Lemma 63, membership in $\mathcal{R}_{T_1}$ can be reduced to state (repeated) reachability in the VASS $\mathcal{V}(T_1, \beta)$. For a given VASS, (repeated) reachability is decided by non-deterministically generating runs of the VASS up to a certain length, using space $O(\log n \cdot 2^{c \cdot d \log d})$ where n is the number of states, d is the vector dimension and c is a constant [37]. The same approach can be used for the VASS $\mathcal{V}(T_1, \beta)$, with the added complication that generating transitions requires membership tests in the relations $\mathcal{R}_{T_c}$'s for $T_c \in child(T_1)$. These in turn become (repeated) reachability tests in the corresponding VASS. Assuming that n and d are upper bounds for the number of states and dimensions for all $\mathcal{V}(T, \beta)$ with $T \in \mathcal{H}$, this yields a total space bound of $O(h \log n \cdot 2^{c \cdot d \log d})$ for membership testing in $\mathcal{V}(T_1, \beta)$, where h is the depth of $\mathcal{H}$.

In our construction of $\mathcal{V}(T, \beta)$, the vector dimension d is the number of TS-isomorphism types. The number of states n is at most the product of the number of T-isomorphism types, the number states in $B(T, \beta)$, the number of all possible $\bar{o}$ and the number of possible states of $\bar{c}_{ib}$. The worst-case complexity occurs for HAS with unrestricted schemas (cyclic foreign keys) and artifact relations. To understand the impact of the foreign key structure and artifact relations, we also consider the complexity for acyclic and linear-cyclic schemas, and without artifact relations. A careful analysis yields the following (see Appendix C). For better readability, we state the complexity for HAS over a fixed schema (database and maximum arity of artifact relations). The impact of the schema is detailed in Appendix C.

Theorem 64. *Let Γ be a HAS over a fixed schema and φ_f an HLTL-FO formula over Γ. The deterministic space complexity upper bounds of checking whether $\Gamma \models \varphi_f$ are summarized in Table 1.* [5]

Table 1. Space complexity upper bounds for verification without arithmetic (N: size of (Γ, φ_f); h: depth of hierarchy; c: constants depending on the schema).

	Acyclic	Linearly-Cyclic	Cyclic
w/o. Artifact relations	$c \cdot N^{O(1)}$	$O(N^{c \cdot h})$	h- exp$(O(N))$
w. Artifact relations	$O(\exp(N^c))$	$O(2\text{- exp}(N^{c \cdot h}))$	$(h+2)$- exp$(O(N))$

Note that the worst-case space complexity is non-elementary, as for feedback-free systems [21]. However, the height of the tower of exponentials in [21] is the square of the total number of artifact variables of the system, whereas in our case it is the depth of the hierarchy, likely to be much smaller.

Lower bounds. Several complexity lower bounds for verification can be immediately obtained. When there are no artifact relations, the verification problem is PSPACE-hard, as LTL model checking alone is already PSPACE-complete [63]. When artifact relations are present, one can show EXPSPACE-hardness by a direct simulation of VASS by HAS, and the fact that model checking for VASS is EXPSPACE-complete [37, 46, 55]. These lower bounds are tight when the schema is acyclic. Lower bounds for the other cases remain open.

6 VERIFICATION WITH ARITHMETIC

We next outline the extension of our verification algorithm to handle HAS and HLTL-FO properties whose conditions use arithmetic constraints expressed as polynomial inequalities with integer coefficients over the numeric variables (ranging over $\mathbb{R}$). We note that one could alternatively limit the arithmetic constraints to linear inequalities with integer coefficients (and variables ranging over $\mathbb{Q}$), yielding the same complexity. These are sufficient for many applications.

The seed idea behind our approach is that, in order to determine whether the arithmetic constraints are satisfied, we do not need to keep track of actual valuations of the task variables and the numeric navigation expressions they anchor (for which the search space would be infinite). Instead, we show that these valuations can be partitioned into a finite set of equivalence classes with respect to satisfaction of the arithmetic constraints, which we then incorporate into the isomorphism types of Section 5, extending the algorithm presented there. This however raises some significant technical challenges, which we discuss next.

Intuitively, this approach uses the fact that a finite set of polynomials $\mathcal{P}$ partitions the space into a bounded number of *cells* containing points located in the same region ($= 0, < 0, > 0$) with respect to every polynomial $P \in \mathcal{P}$. Isomorphism types are extended to include a cell, which determines which arithmetic constraints are satisfied in the conditions of services and in the property. In addition to the requirements detailed in Section 5, we need to enforce cell compatibility across symbolic service calls. For instance, when a task executes an internal service, the corresponding symbolic transition from cell c to c' is possible only if the projections of c and c' on the subspace corresponding to the task's input variables have non-empty intersection (since input variables are preserved). Similarly, when the opening or closing service of a child task is called, compatibility is required

[5] k- exp is the tower of exponential functions of height k.

between the parent's and the child's cell on the shared variables, which amounts again to non-empty intersection between cell projections. This suggests the following first-cut (and problematic) attempt at a verification algorithm: once a local transition imposes new constraints, represented by a cell c', these constraints are propagated *back* to previously guessed cells, refining them via intersection with c'. If an intersection becomes empty, the candidate symbolic run constructed so far has no corresponding actual run and the search is pruned. The problem with this attempt is that it is incompatible with the way we deal with sets in Section 5: the contents of sets are represented by associating counters to the isomorphism types of their elements. Since extended isomorphism types include cells, retroactive cell intersection invalidates the counters and the results of previous VASS reachability checks.

We develop an alternative solution that avoids retroactive cell intersection altogether. More specifically, for each task, our algorithm extends isomorphism types with cells guessed from a *pre-computed* set constructed by following the task hierarchy bottom-up and including in the parent's set those cells obtained by appropriately projecting the children's cells on shared variables and expressions. Only non-empty cells are retained. We call the resulting cell collection the Hierarchical Cell Decomposition (HCD).

The key benefit of the HCD is that it arranges the space of cells so that consistency of a symbolic run can be guaranteed by performing simple local compatibility tests on the cells involved in each transition. Specifically, (i) in the case of internal service calls, the next cell c' must *refine* the current cell c on the shared variables (that is, the projection of c' must be contained in the projection of c); (ii) in the case of child task opening/closing services, the parent cell c must refine the child cell c'. This ensures that in case (i) the intersection with c' of all relevant previously guessed cells is non-empty (because we only guess non-empty cells and c' refines all prior guesses), and in case (ii) the intersection with the child's cell c' is a no-op for the parent cell. Consequently, retroactive intersection can be skipped as it can never lead to empty cells.

A natural starting point for constructing the HCD is to gather for each task all the polynomials appearing in its arithmetic constraints (or in the property sub-formulas referring to that task), and associate sign conditions to each. This turns out to be insufficient. For example, the projection from the child cell can impose on the parent variables new constraints which do not appear explicitly in the parent task. It is a priori not obvious that the constrained cells can be represented symbolically, let alone efficiently computed. The tool enabling our solution is the Tarski-Seidenberg Theorem [61], which ensures that the projection of a cell is representable by a union of cells defined by a set of polynomials (computed from the original ones) and sign conditions for them. The polynomials can be efficiently computed using quantifier elimination.

Observe that a bound on the number of newly constructed polynomials yields a bound on the number of cells in the HCD, which in turn implies a bound on the number of distinct extended isomorphism types manipulated by the verification algorithm, ultimately yielding decidability of verification. A naive analysis produces a bound on the number of cells that is hyperexponential in the height of the task hierarchy, because the number of polynomials can proliferate at this rate when constructing all possible projections, and p polynomials may produce 3^p cells. Fortunately, a classical result from real algebraic geometry ([6], reviewed in Appendix D.2) bounds the number of distinct *non-empty* cells to only exponential in the number of variables (the exponent is independent of the number of polynomials). This yields an upper bound of the number of cells (and also the number of extended isomorphism types) which is singly exponential in the number of numeric expressions and doubly exponential in the height of the hierarchy $\mathcal{H}$. We state below our complexity

results for verification with arithmetic, relegating details (including a fine-grained analysis) to Appendix D.

Theorem 65. *Let Γ be a HAS over a fixed database schema and φ_f an HLTL-FO formula over Γ. If arithmetic is allowed in (Γ, φ_f), then the deterministic space complexity upper bounds for checking whether $\Gamma \models \varphi_f$ are summarized in Table 2.*

Table 2. Space complexity upper bounds for verification with arithmetic (N: size of (Γ, φ); h: depth of hierarchy; c: constants depending on the schema.)

	Acyclic	Linearly-Cyclic	Cyclic
w/o. Artifact relations	$O(\exp(N^{c \cdot h}))$	$O(\exp(N^{c \cdot h^2}))$	$(h+1)$- $\exp(O(N))$
w. Artifact relations	$O(2\text{-}\exp(N^{c \cdot h}))$	$O(2\text{-}\exp(N^{c \cdot h^2}))$	$(h+2)$- $\exp(O(N)))$

7 RELATED WORK

We have already discussed the main related work in the introduction. We summarize next other related work on verification of artifact systems.

Initial work on formal analysis of artifact-based business processes in restricted contexts has investigated reachability [35, 36], general temporal constraints [36], and the existence of complete execution or dead end [13]. For each considered problem, verification is generally undecidable; decidability results were obtained only under rather severe restrictions, e.g., restricting all pre-conditions to be "true" [35], restricting to bounded domains [13, 36], or restricting the pre- and post-conditions to be propositional, and thus not referring to data values [36]. [17] adopts an artifact model variation with arithmetic operations but no database. Decidability relies on restricting runs to bounded length. [66] addresses the problem of the existence of a run that satisfies a temporal property, for a restricted case with no database and only propositional LTL properties. All of these works model no underlying database, sets (artifact relations), task hierarchy, or arithmetic.

A recent line of work has tackled verification of artifact-centric processes with an underlying relational database. [7–10, 23] evolve the business process model and property language, culminating in [38], which addresses verification of first-order μ-calculus (hence branching time) properties over business processes expressed in a framework that is equivalent to artifact systems whose input is provided by external services. [11, 19] extend the results of [38] to artifact-centric multi-agent systems where the property language is a version of first-order branching-time temporal-epistemic logic expressing the knowledge of the agents. This line of work uses variations of a business process model called DCDS (data-centric dynamic systems), which is sufficienty expressive to capture the GSM model, as shown in [59]. In their unrestricted form, DCDS and HAS have similar expressive power. However, the difference lies in the tackled verification problem and in the restrictions imposed to achieve decidability. We check satisfaction of linear-time properties for every possible choice of initial database instance, whereas the related line checks branching-time properties and assumes that the initial database is given. None of the related works address arithmetic. In the absence of arithmetic, the restrictions introduced for decidability are incomparable (neither subsumes the other).

Beyond artifact systems, there is a plethora of literature on data-centric processes, dealing with various static analysis problems and also with runtime monitoring and synthesis. We discuss the most related works here and refer the reader to the surveys [18, 27] for more. Static analysis for semantic web services is considered in [51], but in a context restricted to finite domains. The

works [3, 28, 60] are ancestors of [26] from the context of verification of electronic commerce applications. Their models could conceptually (if not naturally) be encoded in HAS but correspond only to particular cases supporting no arithmetic, sets, or hierarchies. Also, they limit external inputs to essentially come from the active domain of the database, thus ruling out fresh values introduced during the run.

Petri nets are widely used in modeling business processes (see [62] for a survey). A few decidability results have been obtained for Petri-net-based models extended with data components [4, 43, 44, 50]. Although equivalent to Petri nets, our use of VASS in this paper is fundamentally different because our model of business data does not include an explicit Petri net component. Instead, we reduce the verification problem to known solvable problems for VASS.

The present paper extends our previous work [1] by including formal definitions, technical development, and the full proofs of the main results. The proof techniques provide new insights, including a novel use of VASS and quantifier elimination in artifact verification algorithms.

8 CONCLUSION

We showed decidability of verification for a rich artifact model capturing core elements of IBM's successful GSM system: task hierarchy, concurrency, database keys and foreign keys, arithmetic constraints, and richer artifact data. The extended framework requires the use of novel techniques including nested Vector Addition Systems and a variant of quantifier elimination tailored to our context. We improve significantly on previous work on verification of artifact systems with arithmetic [21], which only exhibits non-elementary upper bounds regardless of the schema shape, even absent artifact relations. In contrast, for acyclic and linearly-cyclic schemas, even in the presence of arithmetic and artifact relations, our new upper bounds are elementary (doubly-exponential in the input size and triply-exponential in the depth of the hierarchy). Moreover, the complexity of our verification algorithm gracefully reduces to PSPACE (for acyclic schema) and EXPSPACE in the hierarchy depth (for linearly-cyclic schema) when arithmetic and artifact relations are not present. The sole remaining case of nonelementary complexity occurs for arbitrary cyclic schemas. Altogether, our results provide substantial new insight and techniques for the automatic verification of artifact systems. We recently used the techniques developed in the paper to implement a verifier for HAS with acyclic database schemas, that exhibits very good performance on a realistic benchmark obtained from existing sets of business process specifications and properties by extending them with data-aware features [2]. This points to HAS with acyclic schemas as a sweet spot for verification, and is a strong indication of the practical potential of our approach.

REFERENCES

[1] ANONYMOUS1. details omitted due to double-blind reviewing.
[2] ANONYMOUS2. details omitted due to double-blind reviewing.
[3] Serge Abiteboul, Victor Vianu, Brad Fordham, and Yelena Yesha. 2000. Relational transducers for electronic commerce. *JCSS* 61, 2 (2000), 236–269.
[4] Eric Badouel, Loïc Hélouët, and Christophe Morvan. 2016. Petri nets with structured data. *Fundamenta Informaticae* 146, 1 (2016), 35–82.
[5] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. 2006. *Algorithms in Real Algebraic Geometry (Algorithms and Computation in Mathematics).* Springer-Verlag New York, Inc., Secaucus, NJ, USA.
[6] Saugata Basu, Richard Pollak, and Marie-Françoise Roy. 1996. On the number of cells defined by a family of polynomials on a variety. *Mathematika* 43, 1 (1996), 120–126.
[7] Francesco Belardinelli, Alessio Lomuscio, and Fabio Patrizi. 2011. A Computationally-Grounded Semantics for Artifact-Centric Systems and Abstraction Results. In *IJCAI*. 738–743.

[8] Francesco Belardinelli, Alessio Lomuscio, and Fabio Patrizi. 2011. Verification of Deployed Artifact Systems via Data Abstraction. In *ICSOC*.

[9] Francesco Belardinelli, Alessio Lomuscio, and Fabio Patrizi. 2012. An Abstraction Technique for the Verification of Artifact-Centric Systems. In *KR*.

[10] Francesco Belardinelli, Alessio Lomuscio, and Fabio Patrizi. 2012. Verification of GSM-Based Artifact-Centric Systems through Finite Abstraction. In *ICSOC*.

[11] Francesco Belardinelli, Alessio Lomuscio, and Fabio Patrizi. 2014. Verification of Agent-Based Artifact Systems. *J. Artif. Intell. Res. (JAIR)* 51 (2014), 333–376.

[12] Kamal Bhattacharya, Nathan S Caswell, Santhosh Kumaran, Anil Nigam, and Frederick Y Wu. 2007. Artifact-centered operational modeling: Lessons from customer engagements. *IBM Systems Journal* 46, 4 (2007), 703–721.

[13] Kamal Bhattacharya, Cagdas Gerede, Richard Hull, Rong Liu, and Jianwen Su. 2007. Towards formal analysis of artifact-centric business process models. In *International Conference on Business Process Management*. Springer, 288–304.

[14] Kamal Bhattacharya, Robert Guttman, Kelly Lyman, Fenno F Heath III, Santhosh Kumaran, Prabir Nandi, Frederick Wu, Prasanna Athma, Christoph Freiberg, Lars Johannsen, et al. 2005. A model-driven approach to industrializing discovery processes in pharmaceutical research. *IBM Systems Journal* 44, 1 (2005), 145–162.

[15] BizAgi and Cordys and IBM and Oracle and Singularity and SAP AG and Agile Enterprise Design and Stiftelsen SINTEF and TIBCO and Trisotech. 2013. Case Management Model and Notation (CMMN), FTF Beta 1. (Jan. 2013). http://www.omg.org/spec/CMMN/1.0/Beta1/ OMG Document Number dtc/2013-01-01, Object Management Group.

[16] Michel Blockelet and Sylvain Schmitz. 2011. Model checking coverability graphs of vector addition systems. In *Mathematical Foundations of Computer Science 2011*. Springer, 108–119.

[17] Diego Calvanese, Giuseppe De Giacomo, Richard Hull, and Jianwen Su. 2009. Artifact-centric workflow dominance. In *Service-Oriented Computing*. Springer, 130–143.

[18] Diego Calvanese, Giuseppe De Giacomo, and Marco Montali. 2013. Foundations of data-aware process analysis: a database theory perspective. In *PODS*.

[19] Diego Calvanese, Giorgio Delzanno, and Marco Montali. 2015. Verification of Relational Multiagent Systems with Data Types. In *AAAI*. 2031–2037.

[20] Tian Chao, David Cohn, Adrian Flatgard, Sandy Hahn, Mark Linehan, Prabir Nandi, Anil Nigam, Florian Pinel, John Vergo, and Frederick y Wu. 2009. Artifact-based transformation of IBM global financing. In *International Conference on Business Process Management*. Springer, 261–277.

[21] Elio Damaggio, Alin Deutsch, and Victor Vianu. 2012. Artifact systems with data dependencies and arithmetic. *ACM Transactions on Database Systems (TODS)* 37, 3 (2012), 22. Also in ICDT 2011.

[22] Elio Damaggio, Richard Hull, and Roman Vaculín. 2013. On the equivalence of incremental and fixpoint semantics for business artifacts with Guard–Stage–Milestone lifecycles. *Information Systems* 38, 4 (2013), 561–584.

[23] Giuseppe De Giacomo, Riccardo De Masellis, and Riccardo Rosati. 2012. Verification of Conjunctive Artifact-Centric Services. *Int. J. Cooperative Inf. Syst.* 21, 2 (2012), 111–140.

[24] Giuseppe De Giacomo and Moshe Y Vardi. 2013. Linear temporal logic and linear dynamic logic on finite traces. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*. AAAI Press, 854–860.

[25] Henk de Man. 2009. Case management: Cordys approach. (2009).

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

[27] Alin Deutsch, Richard Hull, and Victor Vianu. 2014. Automatic Verification of Database-Centric Systems. *SIGMOD Record* 43, 3 (2014), 5–17.

[28] Alin Deutsch, Liying Sui, and Victor Vianu. 2007. Specification and Verification of Data-driven Web services. *JCSS* 73, 3 (2007), 442–474.

[29] Leonard Eugene Dickson. 1913. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *American Journal of Mathematics* 35, 4 (1913), 413–422.

[30] Volker Diekert and Paul Gastin. 2004. Pure Future Local Temporal Logics Are Expressively Complete for Mazurkiewicz Traces. In *LATIN 2004: Theoretical Informatics, 6th Latin American Symposium, Buenos Aires, Argentina, April 5-8, 2004, Proceedings*. 232–241.

[31] Volker Diekert and Paul Gastin. 2006. Pure future local temporal logics are expressively complete for Mazurkiewicz traces. *Inf. Comput.* 204, 11 (2006), 1597–1619.

[32] Volker Diekert and Grzegorz Rozenberg. 1995. *The book of traces*. World scientific.

[33] E. Allen Emerson. 1990. Temporal and Modal Logic. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, J. Van Leeuwen (Ed.). North-Holland Pub. Co./MIT Press, 995–1072.

[34] Expedia.Com. 2016. (2016). Retrieved Aug 25, 2016 from http://www.expedia.com

[35] Cagdas E Gerede, Kamal Bhattacharya, and Jianwen Su. 2007. Static analysis of business artifact-centric operational models. In *SOCA*. IEEE, 133–140.

[36] Cagdas E Gerede and Jianwen Su. 2007. Specification and verification of artifact behaviors in business process models. In *ICSOC*. Springer, 181–192.

[37] Peter Habermehl. 1997. On the complexity of the linear-time μ-calculus for Petri nets. In *Application and Theory of Petri Nets 1997*. Springer, 102–116.

[38] Babak Bagheri Hariri, Diego Calvanese, Giuseppe De Giacomo, Alin Deutsch, and Marco Montali. 2013. Verification of relational data-centric dynamic systems with external services. In *PODS*.

[39] Joos Heintz, Pablo Solernó, and Marie-Françoise Roy. 1989. On the Complexity of Semialgebraic Sets. In *IFIP Congress*. 293–298.

[40] Richard Hull, Elio Damaggio, Riccardo De Masellis, Fabiana Fournier, Manmohan Gupta, Fenno Terry Heath III, Stacy Hobson, Mark Linehan, Sridhar Maradugu, Anil Nigam, et al. 2011. Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In *DEBS*. ACM, 51–62.

[41] H.W. Kamp. 1968. Tense logic and the theory of linear order. (1968). Phd thesis, University of California, Los Angeles.

[42] Ralph Kimball and Margy Ross. 2011. The data warehouse toolkit: the complete guide to dimensional modeling. (2011).

[43] Sławomir Lasota. 2016. Decidability border for petri nets with data: WQO dichotomy conjecture. In *International Conference on Applications and Theory of Petri Nets and Concurrency*. Springer, 20–36.

[44] Ranko Lazić, Tom Newcomb, Joël Ouaknine, Andrew W Roscoe, and James Worrell. 2008. Nets with tokens which carry data. *Fundamenta Informaticae* 88, 3 (2008), 251–274.

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

[46] Richard Lipton. 1976. The reachability problem requires exponential space. *Research Report 62, Department of Computer Science, Yale University, New Haven, Connecticut* (1976).

[47] Mike Marin, Richard Hull, and Roman Vaculín. 2012. Data Centric BPM and the Emerging Case Management Standard: A Short Survey. In *BPM Workshops*.

[48] Richard Mayr. 2003. Undecidable problems in unreliable computations. *Theoretical Computer Science* 297, 1 (2003), 337–354.

[49] Antoni Mazurkiewicz. 1977. Concurrent program schemes and their interpretations. *DAIMI Report Series* 6, 78 (1977).

[50] Marco Montali and Andrey Rivkin. 2016. Model checking Petri nets with names using data-centric dynamic systems. *Formal Aspects of Computing* 28, 4 (2016), 615–641.

[51] Srini Narayanan and Sheila A McIlraith. 2002. Simulation, verification and automated composition of web services. In *WWW*. ACM, 77–88.

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

[53] Doron Peled. 1994. Combining partial order reductions with on-the-fly model-checking. In *Computer aided verification*. Springer, 377–390.

[54] E. L. Post. 1947. Recursive Unsolvability of a Problem of Thue. *J. of Symbolic Logic* 12 (1947), 1–11.

[55] Charles Rackoff. 1978. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science* 6, 2 (1978), 223–231.

[56] Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. 2005. *Database System Concepts, 5th Edition*. McGraw-Hill.

[57] Michael Sipser. 1997. *Introduction to the theory of computation*. PWS Publishing Company.

[58] A. Prasad Sistla, Moshe Y. Vardi, and Pierre Wolper. 1987. The Complementation Problem for Büchi Automata with Applications to Temporal Logic. *Theoretical Computer Science* 49 (1987), 217–237.

[59] Dmitry Solomakhin, Marco Montali, Sergio Tessaris, and Riccardo De Masellis. 2013. Verification of Artifact-Centric Systems: Decidability and Modeling Issues. In *ICSOC*. 252–266.

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

[61] Alfred Tarski. 1951. A decision method for elementary algebra and geometry. *1948* (1951).

[62] Wil MP Van der Aalst. 2013. Business process management: a comprehensive survey. *ISRN Software Engineering* 2013 (2013).

[63] Moshe Y Vardi and Pierre Wolper. 1986. An automata-theoretic approach to automatic program verification. In *LICS*. 322–331.

[64] Panos Vassiliadis and Timos Sellis. 1999. A survey of logical models for OLAP databases. *ACM Sigmod Record* 28, 4 (1999), 64–69.

[65] Pierre Wolper, Moshe Y Vardi, Prasad Sistla, et al. 1983. Reasoning about infinite computation paths. In *Foundations of Computer Science, 1983., 24th Annual Symposium on*. IEEE, 185–194.

[66] Xiangpeng Zhao, Jianwen Su, Hongli Yang, and Zongyan Qiu. 2009. Enforcing constraints on life cycles of business artifacts. In *Theoretical Aspects of Software Engineering*. IEEE, 111–118.

[67] Wei-Dong Zhu, Brian Benoit, Bob Jackson, Johnson Liu, Mike Marin, Seema Meena, Juan Felipe Ospina, Guillermo Rios, et al. 2015. Advanced Case Management with IBM Case Manager. (2015).

A FRAMEWORK AND HLTL-FO

A.1 Review of LTL

We review the classical definition of linear-time temporal logic (LTL) over a set P of propositions. LTL specifies properties of infinite words (ω-words) $\{\tau_i\}_{i\geq 0}$ over the alphabet consisting of truth assignments to P. Let $\tau_{\geq j}$ denote $\{\tau_i\}_{i\geq j}$, for $j \geq 0$.

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

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

Observe that the above temporal operators can simulate all commonly used operators, including **G** (always) and **F** (eventually). Indeed, $\mathbf{F}\varphi \equiv \mathit{true}\ \mathbf{U}\ \varphi$ and $\mathbf{G}\varphi \equiv \neg(\mathbf{F}\neg\varphi)$.

The standard construction of a Büchi automaton B_φ corresponding to an LTL formula φ is given in [58, 63]. The automaton B_φ has exponentially many states and accepts precisely the set of ω-words that satisfy φ.

It is sometimes useful to apply LTL on *finite* words rather than ω-words. The finite semantics we use for temporal operators is the following [24]. Let $\{\tau_i\}_{0\leq i\leq n}$ a finite sequence of truth values of P. Similarly to the above, let $\tau_{\geq j}$ denote $\{\tau_i\}_{j\leq i\leq n}$, for $0 \leq j \leq n$. The semantics of **X** and **U** are defined as follows:

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

It is easy to verify that for the B_φ obtained by the standard construction [58, 63] there is a subset Q^{fin} of its states such that B_φ viewed as a finite-state automaton with final states Q^{fin} accepts precisely the finite words that satisfy φ.

A.2 Proof of Theorem 20

We show that it is undecidable whether a HAS $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ satisfies an LTL formula over Σ. The proof is by reduction from the repeated state reachability problem of VASS with reset arcs and bounded lossiness (RB-VASS) [48]. An RB-VASS extends the VASS reviewed in Section 5 as follows. In addition to increments and decrements of the counters, an action of RB-VASS also allows resetting the values of some counters to 0. And after each transition, the value of each counter can decrease non-deterministically by an integer value bounded by some constant c. The results in [48] (Definition 2 and Theorem 18) indicate that the repeated state reachability problem for RB-VASS is undecidable for every fixed $c \geq 0$, since the structural termination problem for Reset Petri-net with bounded lossiness can be reduced to the repeated state reachability problem for RB-VASS's. In our proof, we use RB-VASS's with $c = 1$.

Formally, a RB-VASS $\mathcal{V}$ (with lossiness bound 1 and dimension $d > 0$) is a pair (Q, A) where Q is a finite set of states and A is a set of actions of the form $(p, \bar{a}, q)$ where $\bar{a} \in \{-1, +1, r\}^d$, and $p, q \in Q$. A run of $\mathcal{V} = (Q, A)$ is a sequence $(q_0, \bar{z}_0), \ldots (q_n, \bar{z}_n)$ where $\bar{z}_0 = \bar{0}$ and for each $i \geq 0$, $q_i \in Q$, $\bar{z}_i \in \mathbb{N}^d$, and for some $\bar{a}$ such that $(q_i, \bar{a}, q_{i+1}) \in A$, and for $1 \leq j \leq d$:

- if $\bar{a}(j) \in \{-1, +1\}$, then $\bar{z}_{i+1}(j) = \bar{z}_i(j) + \bar{a}(j)$ or $\bar{z}_{i+1}(j) = \bar{z}_i(j) + \bar{a}(j) - 1$, and

- if $\bar{a}(j) = r$, then $\bar{z}_{i+1}(j) = 0$.

For a given RB-VASS $\mathcal{V} = (Q, A)$ and a pair of states $q_0, q_f \in Q$, we say that q_f is repeatedly reachable from q_0 if there exists a run $(q_0, \bar{z}_0) \dots (q_n, \bar{z}_n) \dots (q_m, \bar{z}_m)$ of $\mathcal{V}$ such that $q_n = q_m = q_f$ and $\bar{z}_n \leq \bar{z}_m$. As discussed above, checking whether q_f is repeatedly reachable from q_0 is undecidable.

We now show that for a given RB-VASS $\mathcal{V} = (Q, A)$ and (q_0, q_f), one can construct a HAS $\Gamma = \langle \mathcal{A}, \Sigma, \Pi \rangle$ and LTL property Φ over Σ such that q_f is repeatedly reachable from q_0 iff $\Gamma \models \Phi$. At a high level, the construction of Γ uses d tasks to simulate the d-dimensional vector of counters. Each task is equipped with an artifact relation, and the number of elements in the artifact relation is the current value of the corresponding counter. Increment and decrement the counters are simulated by internal services of these tasks, and reset of the counters are simulated by closing and reopening the task (recall that this resets the artifact relation to empty). Then we specify in the LTL formula Φ that the updates of the counters of the same action are grouped in sequence. Note that this requires coordinating the actions of sibling tasks, which is not possible in HLTL-FO. The construction is detailed next.

The database schema of Γ consists of a single unary relation $R(\underline{id})$. The artifact system has a root task T_1 and subtasks $\{P_0, P_1, \dots, P_d, C_1, \dots, C_d\}$ which form the following tasks hierarchy:

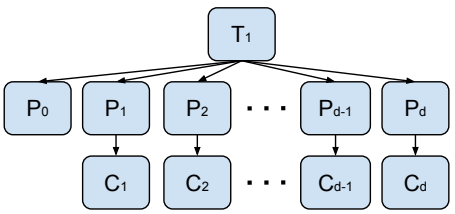

Fig. 14. Tasks hierarchy.

The tasks are defined as follows. The root task T_1 has no variables nor internal services. The task P_0 contains a numeric variable s, indicating the current state of the RB-VASS. For each $q \in Q$, P_0 has a service σ^q, whose pre-condition is true and post-condition sets s to q.

For $i \geq 1$, task P_i has no variable. It has a single internal service σ_i^r whose pre- and post-conditions are both `true`.

Each C_i has an ID variable x, an artifact relation S_i and a pair of services σ_i^+ and σ_i^-, which simply insert x into S_i and removes an element from S_i, respectively. Intuitively, the size of S_i is the current value of the i-th counter. Application of service σ_i^r corresponds to resetting the i-th counter. And application of services σ_i^+ and σ_i^- correspond to increment and decrement of the i-th counter, respectively.

Except for the closing condition of T_1, all opening and closing conditions of tasks are `true`.

We encode the set of actions A into an LTL formula as follows. For each state $p \in Q$, we denote by $\alpha(p)$ the set of actions starting from p. For each action $\alpha = (p, \bar{a}, q) \in A$, we construct an LTL formula $\varphi(\alpha)$ as follows. First, let $\phi_1, \dots \phi_d, \phi_{d+1}$ be LTL formulas where:

- $\phi_{d+1} = \mathbf{X}\sigma^q$,
- for $i = d, d-1, \dots, 1$:
  - if $\bar{a}(i) = +1$, then $\phi_i = \sigma_i^+ \wedge \mathbf{X}\phi_{i+1}$,

- if $\bar{a}(i) = -1$, then $\phi_i = (\sigma_i^- \wedge \mathbf{X}\phi_{i+1}) \vee (\sigma_i^- \wedge \mathbf{X}(\sigma_i^- \wedge \mathbf{X}\phi_{i+1}))$, and
- if $\bar{a}(i) = r$, then $\phi_i = \sigma_i^c \wedge \mathbf{X}(\sigma_i^r \wedge \mathbf{X}(\sigma_i^o \wedge \mathbf{X}\phi_{i+1}))$ where σ_i^o and σ_i^c are the opening and closing services of task C_i.

Let $\varphi(\alpha) = \mathbf{X}\phi_1$. Intuitively, $\varphi(\alpha)$ specifies a sequence of service calls that update the content of the artifact relations $S_1, \ldots S_d$ according to the vector $\bar{a}$. In particular, for $\bar{a}(i) = r$, the subsequence of services $\sigma_i^c \sigma_i^r \sigma_i^o$ first closes task C_i then reopens it. This empties S_i. For $\bar{a}(i) = +1$, by executing σ_i^+, the size of S_i might be increased by 1 or 0, depending on whether the element to be inserted is already in S_i. And for $\bar{a}(i) = -1$, we let σ_i^- to be executed either once or twice, so the size of S_i can decrease by 1 or 2 nondeterministically. Then we let

$$\Phi = \Phi_{\texttt{init}} \wedge \bigwedge_{p \in Q} \mathbf{G}\left(\sigma^p \rightarrow \bigvee_{\alpha \in \alpha(p)} \varphi(\alpha)\right) \wedge \mathbf{GF}\sigma^{q_f}$$

where $\Phi_{\texttt{init}}$ is a formula specifying that the run is correctly initialized, which simply means that the opening services σ_T^o of all tasks are executed once at the beginning of the run, and then a σ^{q_0} is executed.

The second clause says that for every state $p \in Q$, whenever the run enters a state p (by calling σ^p), a sequence of services as specified in $\varphi(\alpha)$ is called to update $S_1, \ldots, S_k$, simulating the action α that starts from p.

Finally, the last clause $\mathbf{GF}\sigma^{q_f}$ guarantees that the service σ^{q_f} is applied infinitely often, which means that q_f is reached infinitely often in the run.

We can prove the following lemma, which implies Theorem 20:

Lemma 66. *For RB-VASS (Q, A) and states $q_0, q_f \in Q$, there exists a run $(q_0, \bar{z}_0), \ldots, (q_m, \bar{z}_m), \ldots, (q_n, \bar{z}_n)$ of (Q, A) where $q_m = q_n = q_f$ and $\bar{z}_m \leq \bar{z}_n$ iff there exists a global run ρ of Γ such that $\rho \models \Phi$.*

A.3 Simplifications

We first show that the global variables, as well as set atoms, can be eliminated from HLTL-FO formulas.

Lemma 67. *Let Γ be a HAS and $\forall \bar{y} \varphi_f(\bar{y})$ an HLTL-FO formula over Γ. One can construct in linear time a HAS $\bar{\Gamma}$ and an HLTL-FO formula $\bar{\varphi}_f$, where $\bar{\varphi}_f$ contains no atoms $S^T(\bar{z})$, such that $\Gamma \models \forall \bar{y} \varphi_f(\bar{y})$ iff $\bar{\Gamma} \models \bar{\varphi}_f$.*

Proof. Consider first the elimination of global variables. Suppose Γ has tasks $T_1, \ldots, T_k$. The Hierarchical artifact system $\bar{\Gamma}$ is constructed from Γ by adding $\bar{y}$ to the variables of T_1 and augmenting the input variables of all other tasks with $\bar{y}$ (appropriately renamed). Note that $\bar{y}$ is unconstrained, so it can be initialized to an arbitrary valuation and then passed as input to all other tasks. Let $\bar{\Gamma}$ consist of the resulting tasks, $\bar{T}_1, \ldots, \bar{T}_k$. It is clear that $\Gamma \models \forall \bar{y} \varphi_f(\bar{y})$ iff $\bar{\Gamma} \models \bar{\varphi}_f$.

Consider now how to eliminate atoms of the form $S^T(\bar{z})$ from $\bar{\varphi}_f$. Recall that for all such atoms, $\bar{z} \subseteq \bar{y}$, so $\bar{z}$ is fixed throughout each run. The idea is keep track of the membership of $\bar{z}$ in S^T using two additional numeric artifact variables $x_{\bar{z}}$ and $y_{\bar{z}}$, such that $x_{\bar{z}} = y_{\bar{z}}$ indicates that $S^T(\bar{z})$ holds[6]. Specifically, a pre-condition ensures that $x_{\bar{z}} \neq y_{\bar{z}}$ initially holds, then $x_{\bar{z}} \neq y_{\bar{z}}$ is enforced as soon as there is an insertion $+S^T(\bar{s}^T)$ for which $\bar{s}^T = \bar{z}$, and $x_{\bar{z}} \neq y_{\bar{z}}$ is enforced again whenever

[6]This is done to avoid introducing constants, that could also be used as flags.

there is a retrieval of a tuple equal to $\bar{z}$. This can be achieved using pre-and-post conditions of services carrying out the insertion or retrieval. Then the atom $S^T(\bar{z})$ can be replaced in $\bar{\varphi}_f$ with $(x_{\bar{z}} = y_{\bar{z}})$. □

We next consider two simplifications of artifact systems regarding the interaction of tasks with their subtasks.

Lemma 68. *Let Γ be a HAS and φ_f an HLTL-FO property over Γ. One can construct a HAS $\tilde{\Gamma}$ and an HLTL-FO formula $\tilde{\varphi}_f$ such that $\Gamma \models \varphi_f$ iff $\tilde{\Gamma} \models \tilde{\varphi}_f$ and: (i) $\bigcup_{T_c \in child(T)} \bar{x}^T_{T_c\uparrow}$ and $\bigcup_{T_c \in child(T)} \bar{x}^T_{T_c\downarrow}$ are disjoint for each task T in $\tilde{\Gamma}$, (ii) for each child task $T_c \in child(T)$, $\bar{x}^T_{T_c\uparrow} \cap \mathrm{VAR}_{\mathbb{R}} = \emptyset$.*

Proof. Consider (i). We describe here informally the construction of $\tilde{\Gamma}$ that eliminates overlapping between $\bigcup_{T_c \in child(T)} \bar{x}^T_{T_c\uparrow}$ and $\bigcup_{T_c \in child(T)} \bar{x}^T_{T_c\downarrow}$. For each task T and for each subtask T_c of T, for each variable $x \in \bar{x}^T_{T_c\downarrow}$, we introduce to T a new variable $\hat{x}$ whose type is the same as the type (id or numeric) of x. We denote by $\hat{x}^T_{T_c\downarrow}$ the set of variables added to T for subtask T_c. Then instead of passing $\bar{x}^T_{T_c\downarrow}$ to T_c, T passes $\hat{x}^T_{T_c\downarrow}$ to T_c when T_c opens. And for the opening service $\sigma^o_{T_c}$ with opening condition π, we check π in conjunction with $\bigwedge_{x \in \bar{x}^T_{T_c\downarrow}} (x = \hat{x})$. Note that $\bigcup_{T_c \in child(T)} \hat{x}^T_{T_c\downarrow}$ and $\bigcup_{T_c \in child(T)} \bar{x}^T_{T_c\uparrow}$ are disjoint. By this construction, in each run of $\tilde{\Gamma}$, after each application of an internal service σ of task T, the variables in $\hat{x}^T_{T_c\downarrow}$ for each subtask T_c receives a set of non-deterministically chosen values. Then each subtask T_c can be opened only when $\hat{x}^T_{T_c\downarrow}$ and $\bar{x}^T_{T_c\downarrow}$ have the same values. So passing $\hat{x}^T_{T_c\downarrow}$ to T_c is equivalent to passing $\bar{x}^T_{T_c\downarrow}$ to T_c.

To guarantee that there is a bijection from the runs of Γ to the runs of $\tilde{\Gamma}$, we also need to make sure that the values of $\hat{x}^T_{T_c\downarrow}$ are non-deterministically chosen before the first application of internal service. (Recall that they either contain 0 or `null` at the point when T is opened.) So we extend T with an extra binary variable x_{init} and an extra internal service σ^{init}_T. Variable x_{init} indicates whether task T has been "initialized". The service σ^{init}_T has precondition that checks whether $x_{\mathsf{init}} = 0$ and post-condition sets $x_{\mathsf{init}} = 1$. It sets all id variables to `null` and numeric variables 0 except for variables in $\hat{x}^T_{T_c\downarrow}$ for any T_c. So application of σ^{init}_T assigns values to $\hat{x}^T_{T_c\downarrow}$ for every subtask T_c non-deterministically and all other variables are initialized to the initial state when T is opened. All other services are modified such that they can be applied only when $x_{\mathsf{init}} = 1$ and initialize $\hat{x}^T_{T_c\downarrow}$ with non-deterministically chosen values for all subtask T_c. So in a projected run ρ_T of $\tilde{\Gamma}$, the suffix with $x_{\mathsf{init}} = 1$ corresponds to the original projected run of Γ. Thus we only need to rewrite the HLTL-FO property φ_f to $\tilde{\varphi}_f$ such that each formula in Φ_T only looks at the suffix of projected run ρ_T after x_{init} is set to be 1 (namely, each $\psi \in \Phi_T$ is replaced with $\mathbf{F}((x_{\mathsf{init}} = 1) \wedge \psi)$).

Now consider (ii). We outline the construction of $\tilde{\Gamma}$ and $\tilde{\varphi}_f$ informally. For each task T, we introduce a set of new numeric variables $\{x_{T_c} | T_c \in child(T), x \in \bar{x}^T_{T_c\uparrow} \cap VAR_{\mathbb{R}}\}$ to $\bar{x}^T$. Intuitively, these variables contain non-deterministically guessed returning values from each child task T_c. These are passed to each child task T_c as additional input variables. Before T_c returns, these are compared to the values of the returning numeric variables of T_c, and T_c returns only if they are identical. More formally, for each child task T_c of T, variables $\{x_{T_c} | x \in \bar{x}^T_{T_c\uparrow} \cap VAR_{\mathbb{R}}\}$ are passed from T to T_c as part of the input variables of T_c. For each variable x_{T_c} in T, we let $x_{T_c \to T} \in \bar{x}^{T_c}$ be the corresponding input variable of x_{T_c}. And for each x_{T_c}, we denote by x_{ret} the variable in $\bar{x}^{T_c}$ satisfying that $f_{out}(x) = x_{ret}$ for f_{out} in the original Γ. Then at T_c, we remove all numeric

variables from $\bar{x}^{T_c}_{ret}$ and add condition $\bigwedge_{x \in \bar{x}^T_{T_c\uparrow} \cap VAR_{\mathbb{R}}} x_{ret} = x_{T_c \to T}$ to the closing condition of T_c. Note that we need to guarantee that the variables in $\{x_{T_c} | T_c \in child(T), x \in \bar{x}^T_{T_c\uparrow} \cap VAR_{\mathbb{R}}\}$ obtain non-deterministically guessed values. This can be done as in the simulation for (i).

Conditions on $\bar{x}^T$ after a subset T's children has returned are evaluated using the guessed values for the variables returned so far. Specifically, the correct value to be used is the latest returned by a child transaction, if any (recall that children tasks can overwrite each other's numeric return variables in the parent). Keeping track of the sequence of returned transactions and evaluating conditions with the correct value can be easily done directly in the verification algorithm, at negligible extra cost. This means that we can assume that tasks have the form in (ii) without the exponential blowup in the conditions, but with a quadratic blowup in the number of variables.

To achieve the simulation fully via the specification is costlier because some of the conditions needed have exponential size. We next show how this can be done. Intuitively, we guess initially an order of the return of the children transactions and enforce that it be respected. We also keep track of the children that have already returned. Let $child(T) = \{T_1, \ldots, T_n\}$. To guess an order of return, we use new ID variables $\bar{o} = \{o_{ij} \mid 1 \leq i, j \leq n\}$. Intuitively, $o_{ij} \neq \texttt{null}$ says that T_i returns before T_j. We also use new ID variables $\{t_i \mid 1 \leq i \leq n\}$, where $t_i \neq \texttt{null}$ means that T_i has returned. The variables $\bar{o}$ are subject to a condition specifying the axioms for a total order:

$$\wedge_{1 \leq i,j \leq n}(o_{ij} \neq \texttt{null} \vee o_{ji} \neq \texttt{null})$$
$$\wedge_{1 \leq i < j \leq n} \neg(o_{ij} \neq \texttt{null} \wedge o_{ji} \neq \texttt{null})$$
$$\wedge_{1 \leq i,j,m \leq n}((o_{ij} \neq \texttt{null} \wedge o_{jm} \neq \texttt{null}) \rightarrow o_{im} \neq \texttt{null})$$

These are enforced using pre-conditions of services as well as one additional initial internal service (which in turn requires a minor modification to φ_f, similarly to (i)). When T_i returns, t_i is set to a non-null value, and the condition

$$\bigwedge_{1 \leq i,j \leq n} (t_i \neq \texttt{null} \wedge t_j = \texttt{null}) \rightarrow o_{ij} \neq \texttt{null}$$

enforcing that transactions return in the order specified by $\bar{o}$ is maintained using pre-conditions. Observe that, at any given time, the latest transaction that has returned is the T_i such that

$$t_i \neq \texttt{null} \wedge \bigwedge_{1 \leq j \leq n} ((o_{ij} \neq \texttt{null}) \rightarrow t_j = \texttt{null})$$

For each formula π over $\bar{x}^T$, we construct a formula $o(\pi)$ by replacing each variable $x \in \bar{x}^T_{\mathbb{R}}$ with x_{T_c} for the latest T_c where $x \in \bar{x}^T_{T_c\uparrow}$ if there is such T_c). The size of the resulting $o(\pi)$ is exponential in the maximum arity of database relations. Finally we obtain $\tilde{\Gamma}$ and $\tilde{\varphi}_f$ by replacing, for every $T \in \mathcal{H}$, each condition π over $\bar{x}^T$ with $o(\pi)$. One can easily verify that $\tilde{\Gamma} \models \tilde{\varphi}_f$ iff $\Gamma \models \varphi_f$ and for every task T of Γ, $\bar{x}^T_{T_c\uparrow}$ does not contain numeric variables. This completes the proof of (ii). □

The construction in (i) takes linear time in the original specification and property. For (ii), the construction introduces a quadratic number of new variables and the size of conditions becomes exponential in the maximum arity of data-base relations. However, as discussed in Appendix A, the verification algorithm can be slightly adapted to circumvent the blowup in the specification without penalty to the complexity. Intuitively, this makes efficient use of non-determinism, avoiding the explicit enumeration of choices required in the specification, which leads to the exponential blowup.

A.4 Proof of Theorem 27

For conciseness, we refer throughout the proof to *propositionally* interleaving-invariant LTL-FO simply as interleaving-invariant LTL-FO.

Showing that HLTL-FO expresses only interleaving-invariant LTL-FO properties is straightforward. The converse however is non-trivial. We begin by showing a normal form for LTL formulas, which facilitates the application to our context of results from [30, 31] on temporal logics for concurrent processes. Consider the alphabet $\mathbf{H}(\Gamma) = \{(\kappa, \sigma) \mid (\kappa, stg, \sigma) \in \mathbf{A}(\Gamma)\}$. Thus, $\mathbf{H}(\Gamma)$ is $\mathbf{A}(\Gamma)$ with the stage information omitted. Let $\mathcal{H}(\Gamma) = h(\mathcal{L}(\Gamma))$ where $h((\kappa, stg, \sigma)) = (\kappa, \sigma)$. We define local-LTL to be LTL using the set of propositions $\mathrm{P}\Sigma = \{(p, \sigma) \mid p \in P_T, \sigma \in \Sigma_T^{obs}\}$. A proposition (p, σ) holds in $(\bar\kappa, \bar\sigma)$ iff $\bar\sigma = \sigma$ and $\bar\kappa(p)$ is true. The definition of interleaving-invariant local-LTL formula is the same as for LTL.

Lemma 69. *For each interleaving-invariant LTL formula φ over $\mathcal{L}(\Gamma)$ one can construct an interleaving-invariant local-LTL formula $\bar\varphi$ over $\mathcal{H}(\Gamma)$ such that for every $u \in \mathcal{L}(\Gamma)$, $u \models \varphi$ iff $h(u) \models \bar\varphi$ where $h((\kappa, stg, \sigma)) = (\kappa, \sigma)$.*

Proof. We use the equivalence of FO and LTL over ω-words [41]. It is easy to see that each LTL formula φ over $\mathcal{L}(\Gamma)$ can be translated into an FO formula $\psi(\varphi)$ over $\mathcal{H}(\Gamma)$ using only propositions in $\mathrm{P}\Sigma$, such that for every $u \in \mathcal{L}(\Gamma)$, $u \models \varphi$ iff $h(u) \models \psi(\varphi)$. Indeed, it is straightforward to define by FO means the stage of each transaction in a given configuration, as well as each proposition in $P \cup \Sigma$ in terms of propositions in $\mathrm{P}\Sigma$, on words in $\mathcal{H}(\Gamma)$. One can then construct from the FO sentence $\psi(\varphi)$ an LTL formula $\bar\varphi$ equivalent to it over words in $\mathcal{H}(\Gamma)$, using the same set of propositions $\mathrm{P}\Sigma$. The resulting LTL formula is thus in local-LTL, and it is easily seen that it is interleaving-invariant. □

We use the propositional variant HLTL of HLTL-FO, whose semantics over ω-words in $\mathcal{H}(\Gamma)$ is defined similarly to the semantics of HLTL-FO on global runs. Recall that in HLTL, the LTL formulas applying to transaction T use propositions in $P_T \cup \Sigma_T^{obs}$ and expressions $[\psi]_{T_c}$ where T_c is a child of T and ψ is an HLTL formula applying to T_c.

We show the following key fact.

Lemma 70. *For each interleaving-invariant local-LTL formula over $\mathcal{H}(\Gamma)$ there exists an equivalent HLTL formula over $\mathcal{H}(\Gamma)$.*

Proof. To show completeness of HLTL, we use a logic shown in [30, 31] to be complete for expressing LTL properties invariant with respect to valid interleavings of actions of concurrent processes (or equivalently, well-defined on Mazurkievicz traces). The logic, adapted to our framework, operates on partial orders $\preceq_u$ of words $u \in \mathcal{H}(\Gamma)$, and is denoted LTL($\preceq$). For $u = \{(\kappa_i, \sigma_i) \mid i \geq 0\}$, we define the projection of u on T as the subsequence $\pi_T(u) = \{(\kappa_{i_j}|_{P_T}, \sigma_{i_j})\}_{j \geq 0}$ where $\{\sigma_{i_j} \mid j \geq 0\}$ is the subsequence of $\{\sigma_i \mid i \geq 0\}$ retaining all services in Σ_T^{obs}. LTL($\preceq$) uses the set of propositions $\mathrm{P}\Sigma$ and the following temporal operators on $\preceq_u$:

- $\mathbf{X}_T\varphi$, which holds in (κ_i, σ_i) if $\pi_T(v) \neq \epsilon$ for $v = \{(\kappa_j, \sigma_j) \mid j \geq m\}$, where m is a minimum index such that $i \prec_u m$, and φ holds on $\pi_T(v)$;
- $\varphi\ \mathbf{U}_T\ \psi$, which holds in (κ_i, σ_i) if $\pi_T(v) \neq \epsilon$ for $v = \{(\kappa_j, \sigma_j) \mid j \geq i\}$, and $\varphi\ \mathbf{U}\ \psi$ holds on $\pi_T(v)$.

From Theorem 18 in [30] and Proposition 2 and Corollary 26 in [31] it follows that LTL($\preceq$) expresses all local-LTL properties over $\mathcal{H}(\Gamma)$ invariant with respect to interleavings.

We next show that HLTL can simulate LTL($\preceq$). To this end, we consider an extension of HLTL in which LTL($\preceq$) formulas may be used in addition to propositions in $P_T \cup \Sigma_T^{obs}$ in every formula applying to transaction T. We denote the extension by HLTL+LTL($\preceq$). Note that each formula ξ in LTL($\preceq$) is an HLTL+LTL($\preceq$) formula. The proof consists in showing that the LTL($\preceq$) formulas can be eliminated from HLTL+LTL($\preceq$) formulas. This is done by recursively reducing the depth of nesting of X_T and U_T operators, and finally eliminating propositions. We define the *rank* of an LTL($\preceq$) formula to be the maximum number of X_T and U_T operators along a path in its syntax tree. For a formula ξ in HLTL+LTL($\preceq$), we define $r(\xi) = (n, m)$ where n is the maximum rank of an LTL($\preceq$) formula occurring in ξ, and m is the number of such formulas with rank n. The pairs (n, m) are ordered lexicographically.

Let ξ be an HLTL+LTL($\preceq$) formula. For uniformity of notation, we define $[\xi]_{T_1} = \xi$. We associate to ξ the tree $Tree(\xi)$ with root $[\xi]_{T_1}$, whose nodes are all occurrences of subformulas of the form $[\psi]_T$ in ξ, with an edge from $[\psi_i]_{T_i}$ to $[\psi_j]_{T_j}$ if the latter occurs in ψ_i and T_j is a child of T_i in $\mathcal{H}$.

Consider an HLTL+LTL($\preceq$) formula ξ such that $r(\xi) \geq (1, 1)$. Suppose ξ has a subformula $\mathbf{X}_T\varphi$ in LTL($\preceq$) of maximum rank. Pick one such occurrence and let $\bar{T}$ be the minimum task (wrt $\mathcal{H}$) such that $\mathbf{X}_T\varphi$ occurs in $[\psi]_{\bar{T}}$. We construct an HLTL+LTL($\preceq$) formula $\bar{\xi}$ such that $r(\bar{\xi}) < r(\xi)$, essentially by eliminating $\mathbf{X}_T$. We consider 4 cases: $T = \bar{T}$, T is a descendant or ancestor of $\bar{T}$, or neither.

Suppose first that $T = \bar{T}$. Consider an occurrence of $\mathbf{X}_T\varphi$. Intuitively, there are two cases: $\mathbf{X}_T\varphi$ is evaluated inside the run of T corresponding to $[\psi]_T$, or at the last configuration. In the first case ($\neg\sigma_T^c$ holds), $\mathbf{X}_T\varphi$ is equivalent to $\mathbf{X}\varphi$. In the second case (σ_T^c holds), $\mathbf{X}_T\varphi$ holds iff φ holds at the next call to T. Thus, ξ is equivalent to $\xi_1 \vee \xi_2$, where:

(1) ξ_1 says that φ does not hold at the next call to T (or no such call exists) and $\mathbf{X}_T\varphi$ is replaced in ψ by $\neg\sigma_T^c \wedge \mathbf{X}\varphi$

(2) ξ_2 says that φ holds at the next call to T (which exists) and $\mathbf{X}_T\varphi$ is replaced in ψ by $\neg\sigma_T^c \rightarrow \mathbf{X}\varphi$.

We next describe how ξ_1 states that φ does not hold at the next call to T (ξ_2 is similar). We need to state that either there is no future call to T, or such a call exists and $\neg\varphi$ holds at the first such call. Consider the path from T_1 to T in $\mathcal{H}$. Assume for simplicity that the path is $T_1, T_2, \ldots, T_k$ where $T_k = T$. For each i, $1 \leq i < k$, we define inductively (from $k-1$ to 1) formulas $\alpha_i, \beta_i(\neg\varphi)$ such that α_i says that there is no call leading to T in the remainder of the current subrun of T_i, and $\beta_i(\neg\varphi)$ says that such a call exists and the first call leads to a subrun of T satisfying $\neg\varphi$. First, $\alpha_{k-1} = \mathbf{G}(\neg\sigma^o_{T_k})$ and $\beta_{k-1}(\neg\varphi) = \neg\sigma^o_{T_k}\ \mathbf{U}\ [\neg\varphi]_{T_k}$. For $1 \leq i < k-1$, $\alpha_i = \mathbf{G}(\sigma^o_{T_{i+1}} \rightarrow [\alpha_{i+1}]_{T_{i+1}})$ and $\beta_i(\neg\varphi) = (\sigma^0_{T_{i+1}} \rightarrow [\alpha_{i+1}]_{T_{i+1}})\ \mathbf{U}\ [\beta_{i+1}(\neg\varphi)]_{T_{i+1}}$. Now $\xi_1 = \xi_1^0 \vee \bigvee_{1 \leq j < k} \xi_1^j$ where ξ_1^0 states that there is no next call to T and ξ_1^j states that T_j is the minimum task such that the next call to T occurs during the *same* run of T_j (and satisfies $\neg\varphi$). More precisely, let $[\psi_1]_{T_1}, [\psi_2]_{T_2}, \ldots [\psi_k]_{T_k}$ be the path leading from $[\xi]_{T_1}$ to $[\psi]_T$ in $Tree(\xi)$ (so $\psi_1 = \xi$ and $\psi_k = \psi$). Then ξ_1^0 is obtained by replacing each ψ_i by $\bar{\psi}_i$, $1 \leq i < k$, defined inductively as follows. First, $\bar{\psi}_{k-1}$ is obtained from ψ_{k-1} by replacing $[\psi_k]_{T_k}$ with $[\psi_k]_{T_k} \wedge \alpha_{k-1}$. For $1 \leq i < k-1$, $\bar{\psi}_i$ is obtained from ψ_i by replacing $[\psi_{i+1}]_{T_{i+1}}$ with $[\bar{\psi}_{i+1}]_{T_{i+1}} \wedge \alpha_i$. For $1 \leq j < k$, ξ_1^j is obtained by replacing in ψ_j, $[\psi_{j+1}]_{T_{j+1}}$ with $[\bar{\psi}_{j+1}]_{T_{j+1}} \wedge \beta_j(\neg\varphi)$. It is clear that ξ_1 states the desired property. The formula ξ_2 is constructed similarly. Note that $r(\xi_1 \vee \xi_2) < r(\xi)$.

Now suppose T is an ancestor of $\bar{T}$. We reduce this case to the previous ($T = \bar{T}$). Let T' be the child of T. Suppose $[\psi_T]_T$ is the ancestor of $[\psi]_{\bar{T}}$ in $\mathit{Tree}(\xi)$. Then ξ is equivalent to $\bar{\xi} = \xi_1 \vee \xi_2$ where:

(1) ξ_1 says that φ does not hold at the next action of T wrt $\preceq$ (or no such next action exists) and ψ is replaced by $\psi(\mathbf{X}_T\varphi \leftarrow \mathit{false})$ ($\leftarrow$ denotes substitution)

(2) ξ_2 says that φ holds at the next action of T wrt $\preceq$ and ψ is replaced by $\psi(\mathbf{X}_T\varphi \leftarrow \mathit{true})$

To state that φ does not hold at the next call to T (or no such call exists) ξ_1 is further modified by replacing in ψ_T, $[\psi_{T'}]_{T'}$ with $[\psi_{T'}]_{T'} \wedge (\mathbf{G}(\neg\sigma^c_{T'}) \vee (\neg\sigma^c_{T'}\ \mathbf{U}\ (\sigma^c_{T'} \wedge \neg\mathbf{X}_T\varphi))$. Smilarly, ξ_2 is further modified by replacing in ψ_T, $[\psi_{T'}]_{T'}$ with $[\psi_{T'}]_{T'} \wedge (\neg\sigma^c_{T'}\ \mathbf{U}\ (\sigma^c_{T'} \wedge \mathbf{X}_T\varphi))$. Note that there are now two occurrences of $\mathbf{X}_T\varphi$ in the modified ψ_T's. By applying twice the construction for the case $\bar{T} = T$ we obtain an equivalent $\bar{\bar{\xi}}$ such that $r(\bar{\bar{\xi}}) < r(\xi)$.

Next consider the case when $\bar{T}$ is an ancestor of T. Suppose the path from T_1 to T in $\mathcal{H}$ is $T_1, \ldots, T_i, \ldots T_k$ where $T_i = \bar{T}$ and $T_k = T$. Consider the value of $\mathbf{X}_T\varphi$ in the run ρ_ψ of $\bar{T}$ on which ψ is evaluated. Similarly to the case $T = \bar{T}$, there are two cases: φ holds at the next invocation of T following ρ_ψ, or it does not. Thus, ξ is equivalent to $\xi_1 \vee \xi_2$, where:

(1) ξ_1 says that φ does not hold at the next call to T (or no such call exists) and $\mathbf{X}_T\varphi$ is replaced in ψ by $\beta_i(\varphi)$, where $\beta_i(\varphi)$ says that there exists a future call leading to T in the current run of $\bar{T}$, and the first such run of T satisfies φ; $\beta_i(\varphi)$ is constructed as in the case $T = \bar{T}$.

(2) ξ_2 says that φ holds at the next call to T following the current run of $\bar{T}$ and $\mathbf{X}_T\varphi$ is replaced in ψ by $\alpha_i \vee \beta_i(\varphi)$ where α_i, constructed as for the case $T = \bar{T}$, says that there is no future call leading to T in the current run of $\bar{T}$.

To say that φ does not hold at the next call to T following ρ_ψ (or no such call exists), ξ_1 is modified analogously to the case $\bar{T} = T$, and similarly for ξ_2.

Finally suppose the least common ancestor of $\bar{T}$ and T is $\hat{T}$ distinct from both. Let $[\psi_{\hat{T}}]_{\hat{T}}$ be the ancestor of $[\psi]_{\bar{T}}$ in $\mathit{Tree}(\xi)$. Consider the value of $\mathbf{X}_T\varphi$ in the run of $\bar{T}$ on which ψ is evaluated. There are two cases: φ holds at the next invocation of T following the run of $\bar{T}$, or it does not. Thus, ξ is equivalent to $\xi_1 \vee \xi_2$, where:

(1) ξ_1 says that φ does not hold at the next call to T (or no such call exists) and ψ is replaced by $\psi(\mathbf{X}_T\varphi \leftarrow \mathit{false})$

(2) ξ_2 says that φ holds at the next call to T and ψ is replaced by $\psi(\mathbf{X}_T\varphi \leftarrow \mathit{true})$

To say that φ does not hold at the next call to T (or no such call exists), ξ_1 is modified analogously to the case $\bar{T} = T$, and similarly for ξ_2, taking into account the fact that the next call to T, if it exists, must take place in the current run of $\hat{T}$ or of one of its ancestors. This completes the simulation of $\mathbf{X}_T\varphi$.

Now suppose ξ has a subformula $(\varphi_1\ \mathbf{U}_T\ \varphi_2)$ of maximum rank. Pick one such occurrence and let $\bar{T}$ be the minimum task (wrt $\mathcal{H}$) such that $(\varphi_1\ \mathbf{U}_T\ \varphi_2)$ occurs in $[\psi]_{\bar{T}}$. There are several cases: $\bar{T} = T$, $\bar{T}$ is an ancestor or descendant of T, or neither. The simulation technique is similar to the above. We outline the construction for the most interesting case when $\bar{T} = T$.

Consider the run of T on which $[\psi]_T$ is evaluated. There are two cases: (†) $(\varphi_1 \mathbf{U}_T \varphi_2)$ holds on the concatenation of the future runs of T, or (†) does not hold. Thus, ξ is equivalent to $\xi_1 \vee \xi_2$ where:

(1) ξ_1 says that (†) holds and ψ is modified by replacing the occurrence of $(\varphi_1 \mathbf{U}_T \varphi_2)$ with $\mathbf{G}\varphi_1 \vee (\varphi_1 \mathbf{U} \varphi_2)$, and

(2) ξ_2 says that (†) does not hold and ψ is modified by replacing the occurrence of $(\varphi_1 \mathbf{U}_T \varphi_2)$ with $(\varphi_1 \mathbf{U} \varphi_2)$.

We show how ξ_1 ensures (†). Let $T_1, \ldots, T_k$ be the path from root to T in $\mathcal{H}$. For each i, $1 \leq i < k$, we define inductively (from $k-1$ to 1) formulas α_i, β_i as follows. Intuitively, α_i says that all future calls leading to T from the current run of T_i must result in runs satisfying $\mathbf{G}\ \varphi_1$:

- $\alpha_{k-1} = \mathbf{G}(\sigma^o_{T_k} \rightarrow [\mathbf{G}\ \varphi_1]_{T_k})$,
- for $1 \leq i < k-1$, $\alpha_i = \mathbf{G}(\sigma^o_{T_{i+1}} \rightarrow [\alpha_{i+1}]_{T_{i+1}})$

The formula β_i says that there must be a future call to T in the current run of T_i satisfying $\varphi_1 \mathbf{U} \varphi_2$ and all prior calls result in runs satisfying $\mathbf{G}\varphi_1$:

- $\beta_{k-1} = (\sigma^o_{T_k} \rightarrow [\mathbf{G}\varphi_1]_{T_k})\ \mathbf{U}\ [\varphi_1 \mathbf{U} \varphi_2]_{T_k}$,
- for $1 \leq i < k-1$, $\beta_i = (\sigma^o_{T_{i+1}} \rightarrow [\alpha_{i+1}]_{T_{i+1}})\ \mathbf{U}\ [\beta_{i+1}]_{T_{i+1}}$.

Now ξ_1 is $\bigvee_{1 \leq j < k} \xi_j$ where ξ_j states that the concatenation of runs resulting from calls to T within the run of T_j on which $[\psi_j]_{T_j}$ is evaluated, satisfies $(\varphi_1\ \mathbf{U}\ \varphi_2)$. More precisely, let $[\psi_1]_{T_1}, \ldots, [\psi_k]_{T_k}$ be the path from $[\xi]_{T_1}$ to $[\psi]_T$ in $\mathit{Tree}(\xi)$ (so $\psi_1 = \xi$ and $\psi_k = \psi$). For each j we define ψ^j_i, $1 \leq i < k$ as follows:

- if $j < k-1$, ψ^j_{k-1} is obtained from ψ_{k-1} by replacing $[\psi_k]_{T_k}$ with $[\psi_k]_{T_k} \wedge \alpha_{k-1}$
- if $j = k-1$, ψ^j_{k-1} is obtained from ψ_{k-1} by replacing $[\psi_k]_{T_k}$ with $[\psi_k]_{T_k} \wedge \beta_{k-1}$
- for $j < i < k-1$, ψ^j_i is obtained from ψ_i by replacing $[\psi^j_{i+1}]_{T_{i+1}}$ with $[\psi^j_{i+1}]_{T_{i+1}} \wedge \alpha_i$
- ψ^j_j is obtained from ψ_j by replacing $[\psi^j_{j+1}]_{T_{j+1}}$ with $[\psi^j_{j+1}]_{T_{j+1}} \wedge \beta_j$
- for $1 \leq i < j$, ψ^j_i is obtained from ψ_i by replacing $[\psi_{i+1}]_{T_{i+1}}$ with $[\psi^j_{i+1}]_{T_{i+1}}$.

Finally, $\xi_j = [\psi^j_1]_{T_1}$. The formula ξ_2 is constructed along similar lines. This completes the case $(\varphi_1 \mathbf{U}_T \varphi_2)$.

Consider now the case when the formula of maximum rank is a proposition $(p, \sigma) \in \mathrm{P}\Sigma$, where $p \in P_T$ and $\sigma \in \Sigma^{obs}_T$. There are several cases:

- (p, σ) occurs in $[\psi]_T$. Then (p, σ) is replaced with $p \wedge \sigma$.
- (p, σ) occurs in $[\psi]_{\bar{T}}$ where $\bar{T} \neq T$ and $\bar{T}$ is not a child or parent of T. Then (p, σ) is replaced with *false*.
- (p, σ) occurs in $[\psi_{T'}]_{T'}$ for some parent T' of T. If $\sigma \in \Sigma_T$ then (p, σ) is replaced with *false* in $\psi_{T'}$. If $\sigma = \sigma^o_T$ then (p, σ) is replaced by $[p]_T$. If $\sigma = \sigma^c_T$, we use the past temporal operator $\mathbf{S}$ whose semantics is symmetric to $\mathbf{U}$. This can be simulated in LTL, again as a consequence of Kamp's Theorem [41]. The proposition (p, σ) is replaced in $\psi_{T'}$ by $\sigma^c_T \wedge ((\neg\sigma^o_T)\ \mathbf{S}\ [\mathbf{F}(\sigma^c_T \wedge p)]_T)$

- (p, σ) occurs in $[\psi_{T'}]_{T'}$ for some child T' of T. Let $[\psi_T]_T$ be the parent of $[\psi_{T'}]_{T'}$ in $Tree(\xi)$. As above, if $\sigma \in \Sigma_T$ then (p, σ) is replaced with *false* in $\psi_{T'}$. If $\sigma = \sigma^o_{T'}$, there are two cases: (1) p holds in T when the call to T' generating the run on which $\psi_{T'}$ is evaluated is made, and (2) the above is false. Thus, ψ_T is replaced by $\psi^1_T \vee \psi^2_T$ where ψ^1_T corresponds to (1) and ψ^2_T to (2). Specifically:
  - ψ^1_T is obtained from ψ_T by replacing $[\psi_{T'}]_{T'}$ with $p \wedge [\psi^1_{T'}]_{T'}$, where $\psi^1_{T'}$ is obtained from $\psi_{T'}$ by replacing $(p, \sigma^o_{T'})$ with $\sigma^o_{T'}$
  - ψ^2_T is obtained from ψ_T by replacing $[\psi_{T'}]_{T'}$ with $\neg p \wedge [\psi^2_{T'}]_{T'}$ where $\psi^2_{T'}$ is obtained from $\psi_{T'}$ by replacing $(p, \sigma^o_{T'})$ with *false*.

  Now suppose $\sigma = \sigma^c_{T'}$. Again, there are two cases: (1) if T' returns then p holds in the run of T when T' returns, and (2) this is false. The two cases are treated similarly to the above.

This concludes the proof of the lemma. □

Theorem 27 now follows. Let φ_f be an interleaving-invariant LTL-FO formula over Γ. By Lemma 69, we can assume that φ is in local-LTL and in particular uses the set of propositions $P\Sigma$. By Lemma 70, there exists an HLTL formula ξ equivalent to φ over ω-words in $\mathcal{H}(\Gamma)$, using propositions in $P \cup \Sigma$. Moreover, by construction, each sub-formula $[\psi]_T$ of ξ uses only propositions in $P_T \cup \Sigma^{obs}_T$. It is easily seen that ξ_f is a well-formed HLTL-FO formula equivalent to φ_f on all runs of Γ.

B RESTRICTIONS AND UNDECIDABILITY

We provide a proof of Theorem 28 for relaxing restriction (2). Recall that HAS$^{(2)}$ allows subtasks of a given task to overwrite non-null ID variables. The same proof idea can be used for restrictions (1) to (7).

Proof. We show undecidability by reduction from the Post Correspondence Problem (PCP) [54, 57]. Given an instance $P = \{(a_i, b_i)\}_{1 \le i \le k}$ of PCP, where each (a_i, b_i) is a pair of non-empty strings over $\{0, 1\}$, we show how to construct a HAS$^{(2)}$ Γ and HLTL-FO formula φ_f such that there is a solution to P iff there exists a run of Γ satisfying φ_f (i.e., $\Gamma \not\models \neg\varphi_f$).

The database schema of Γ contains a single relation

$$G(\underline{id}, \texttt{next}, \texttt{label})$$

where `next` is a foreign-key attribute referencing attribute *id* and `label` is a non-key attribute. Let α, β be distinct id values in G. A *path* in G from α to β is a sequence of IDs $i_0, \ldots, i_n$ in G where $\alpha = i_0$, $\beta = i_n$, and for each $j, 0 \le j < n$, $i_{j+1} = i_j.\texttt{next}$. It is easy to see that there is at most one path from α to β for which $i_j \ne \alpha, \beta$ for $0 < j < n$, and the path must be simple ($i_0, i_1, \ldots, i_n$ are distinct). If such a path exists, we denote by $w(\alpha, \beta)$ the sequence of labels $i_0.\texttt{label}, \ldots, i_n.\texttt{label}$ (a word over $\{0, 1\}$, assuming the values of `label` are 0 or 1). Intuitively, Γ and φ_f do the following given database G:

(1) non-deterministically pick two distinct ids α, β in G

(2) check that there exists a simple path from α to β and that $w(\alpha, \beta)$ witnesses a solution to P; the uniqueness of the simple path from α to β is essential to ensure that $w(\alpha, \beta)$ is well defined.

Step 2 requires simultaneously parsing $w(\alpha, \beta)$ as $a_{s_1} \dots a_{s_m}$ and $b_{s_1} \dots b_{s_m}$ for some $s_i \in [1, k], 1 \leq i \leq m$, by synchronously walking the path from α to β with two pointers P_a and P_b. More precisely, P_a and P_b are initialized to α. Then repeatedly, an index $s_j \in [1, k]$ is picked non-deterministically, and P_a advances $|a_{s_j}|$ steps to a new position P'_a, such that the sequence of labels along the path from P_a to P'_a is a_{s_j} and no id along the path equals α or β. Similarly, P_b advances $|b_{s_j}|$ steps to a new position P'_b, such that the sequence of labels along the path from P_b to P'_b is b_{s_j} and no id along the path equals α or β. This step repeats until P_a and P_b simultaneously reach β (if ever). The property φ_f checks that eventually $P_a = P_b = \beta$, so $w(\alpha, \beta)$ witnesses a solution to P.

In more detail, we use two tasks T_p and T_c where T_c is a child task of T_p (see Figure 15).

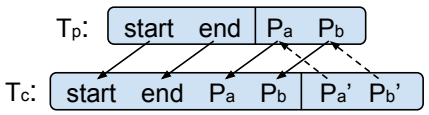

Fig. 15. Undecidability for HAS$^{(2)}$.

Task T_p has two input variables *start*, *end* (initialized to distinct ids α and β by the global precondition), and two artifact variables P_a and P_b (holding the two pointers). T_p also has a binary artifact relation S whose set variables are (P_a, P_b). At each segment of T_p, the subtask T_c is called with $(P_a, P_b, \textit{start}, \textit{end})$ passed as input. Then an internal service of T_c computes P'_a and P'_b, such that P_a, P'_a, P_b and P'_b satisfy the condition stated above for some $s_j \in [1, k]$. Then T_c closes and returns P'_a and P'_b to T_p, overwriting P_a and P_b (note that this is only possible because restriction (2) is lifted). At this point we would like to call T_c again, but multiple calls to a subtasks are disallowed between internal transitions. To circumvent this, we equip T_p with an internal service that simply propagates $(P_a, P_b, \textit{start}, \textit{end})$. The variables *start*, *end* are automatically propagated as input variables of T_p. Propagating (P_a, P_b) is done by inserting it into S and retrieving it in the next configuration (so $\delta = \{+S(P_a, P_b), -S(P_a, P_b)\}$). Now we are allowed to call again T_c, as desired.

It can be shown that there exists a solution to P iff there exists a run of the above system that reaches a configuration in which $P_a = P_b = \textit{end}$. This can be detected by a second internal service *success* of T_p with pre-condition $P_a = P_b = \textit{end}$. Thus, the HLTL-FO property φ_f is simply $[\mathbf{F}(\textit{success})]_{T_p}$. Note that this is in fact an HLTL formula. Thus, checking HLTL-FO (and indeed HLTL) properties of HAS$^{(2)}$ systems is undecidable. □

C COMPLEXITY OF VERIFICATION WITHOUT ARITHMETIC

Let Γ be a HAS and φ_f an HLTL-FO formula over Γ. Recall the VASS $\mathcal{V}(T, \beta)$ constructed for each task T and assignment β to Φ_T. According to the discussion of the complexity of verification in Section 5, checking whether $\Gamma \not\models \varphi_f$ can be done in $O(h \log n \cdot 2^{c \cdot d \log(d)})$ nondeterministic space, where c is a constant, h is the depth of $\mathcal{H}$, and n, d bound the number of states, resp. vector dimensions of $\mathcal{V}(T, \beta)$ for all T and β. We will estimate these bounds using the maximum number of T-isomorphism types, denoted M, and the maximum number of TS-isomorphism types, denoted D. We also denote by N the size of (Γ, φ_f). To complete the analysis, the specific bounds M and D will be computed for acyclic, linear-cyclic, and cyclic schemas, as well as with and without artifact relations.

By our construction, the vector dimension of each $\mathcal{V}(T, \beta)$ is the number of TS-isomorphism types, so bounded by D. The number of states is at most the product of the number of distinct

T-isomorphism types, the number states in $B(T, \beta)$, the number of all possible $\bar{o}$ and the number of possible states of $\bar{c}_{ib}$. And since the number of T_c-isomorphism types is no more than the number of T-isomorphism types if T_c is child of T, the number of all possible $\bar{o}$ is at most $(3+M)^{|child(T)|} \leq (3+M)^N$. Note that the number of states in $B(T, \beta)$ is at most exponential in the size of the HLTL-FO property φ_f (extending the classical construction [65]). Thus, $n = M \cdot 2^{O(N)} \cdot (3+M)^N \cdot 2^D$ bounds the number of states of all $\mathcal{V}(T, \beta)$. It follows that $O(h \log n \cdot 2^{c \cdot d \log(d)}) = O(h \cdot N \cdot \log M \cdot 2^{c \cdot D \cdot \log D})$, yielding the complexity of checking $\Gamma \not\models \varphi_f$. Thus, checking whether $\Gamma \models \varphi_f$ can be done in $O(h^2 \cdot N^2 \log^2 M \cdot 2^{c \cdot D \log D})$ deterministic space by Savitch's Theorem [57], for some constant c.

For artifact systems with no artifact relation, the bounds degrade to $O(h \cdot N \log M)$ and $O(h^2 \cdot N^2 \log^2 M)$.

The number of T- and TS-isomorphism types depends on the type of the schema DB of Γ, as described next. In our analysis, we denote by r the number of relations in DB and a the maximum arity of relations in DB. We also let $k = \max_{T \in \mathcal{H}} |\bar{x}^T|$, $s = \max_{T \in \mathcal{H}} |\bar{s}^T|$ and h be the height of $\mathcal{H}$.

Acyclic schema. if DB is acyclic, then the length of each expression in the navigation set is bounded by the number of relations in DB. So the size of the navigation set of each T-isomorphism type is at most $a^r k$. The total number of T-isomorphism types is at most the product of the number of possible navigation sets and the number of possible equality types. So $M = (r+1)^k \cdot (a^r k)^{a^r k}$ is a bound for the number of T-isomorphism types for every T.

For TS-isomorphism types, we note that within the same path in $\mathcal{V}(T, \beta)$, all TS-isomorphism types have the same projections on $\bar{x}^T_{\text{in}}$ since the input variables are unchanged throughout a local symbolic run. So within each query of (repeated) reachability, each TS-isomorphism type can be represented by (1) the equality connections from expressions starting with $x \in \bar{x}^T_{\text{in}}$ to expressions starting with $x \in \bar{s}^T$ and (2) the equality connections within expressions starting with $x \in \bar{s}^T$. For (1), the total number of all possible connections is at most ${M_1}^{M_2}$ where M_1 is the number of expressions starting with $x \in \bar{x}^T_{\text{in}}$ and M_2 is the number of expressions starting with $x \in \bar{s}^T$. For (2), the total number of all possible connections is at most $M_2^{M_2}$. Note that $M_1 \leq a^r k$ and $M_2 \leq a^r s$. So the total number of TS-isomorphism type is at most $D = (r+1)^s \cdot (a^r k \cdot a^r s)^{a^r s} = (r+1)^s \cdot (a^{2r} k \cdot s)^{a^r s}$. So for DB of fixed size and S^T of fixed arity, the number of T-isomorphism type is exponential in k and the number of TS-isomorphism type is polynomial in k.

By substituting the above values of M and D in the space bound $O(h^2 \cdot N^2 \log^2 M \cdot 2^{c \cdot D \log D})$, we obtain:

Theorem 71. *For HAS Γ with acyclic schema and HLTL-FO property φ_f over Γ, $\Gamma \models \varphi_f$ can be checked in $O(\exp(N^{c_1}))$ deterministic space, where $c_1 = O(a^r \log r s)$. If Γ does not contain artifact relations, then $\Gamma \models \varphi_f$ can be checked in $c_2 \cdot N^{O(1)}$ deterministic space, where $c_2 = O(a^{2r} \log^2 a^r)$.*

Note that if DB is a Star schema [42, 64], which is a special case of acyclic schema, then the size of the navigation set is at most ark instead of $a^r k$. So verification has the complexities stated in Theorem 71, with constants $c_1 = O(ars)$ and $c_2 = O(ar^2 \log^2 ar)$ respectively.

Note that with the simulation used in Lemma 68, the number of variables is at most quadratic in the original number of variables. This only affects the constants in the above complexities.

Linearly-cyclic schema. Consider the case where DB is linearly cyclic. To bound the number of T- and TS-isomorphism types, it is sufficient to bound $h(T)$, which equals to $1 + k \cdot F(\delta)$ where $\delta = \max_{T_c \in child(T)}\{h(T_c)\}$ if T is a non-leaf task and $\delta = 1$ if T is a leaf. And recall that $F(\delta)$ is the

maximum number of distinct paths of length at most δ starting from any relation in the foreign key graph FK. If DB is linearly cyclic, then by definition, the graph of cycles in FK form an acyclic graph G (each node in G is a cycle in the FK graph and there is an edge from cycle u to cycle v iff there is an edge from some node in u to some node in v in FK).

Consider each path P of length at most δ in FK. P can be decomposed into a list of subsequences of nodes, where each subsequence consists of nodes within the same cycle in FK (as shown in Figure 16).

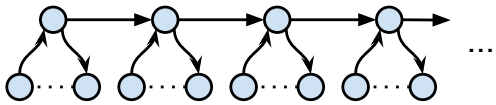

Fig. 16. A path in a Linearly-Cyclic Foreign Key graph.

So $F(\delta)$ can be bounded by the product of (1) the number of distinct paths in G starting from any cycle and (2) the maximum number distinct paths of length at most δ formed using subsequences of nodes from cycles within the same path in G. It is easy to see that (1) is at most a^r. And since the length of a path in G is at most r, (2) is at most δ^r. Thus $F(\delta)$ is bounded by $a^r \cdot \delta^r = (a \cdot \delta)^r$.

So if DB is linearly cyclic, then $h(T)$ is bounded by $1 + a^r k$ if T is a leaf task and $h(T)$ is bounded by $1 + (a \cdot \delta)^r \cdot k$ if T is non-leaf task where $\delta = \max_{T_c \in child(T)}\{h(T_c)\}$. By solving the recursion, for every task T, we have that $h(T) \leq c \cdot (a \cdot k)^{r \cdot h}$ for some constant c. So the size of the navigation set of each T-isomorphism type is at most $c \cdot (a \cdot k)^{r(h+1)}$. Thus the number of T- and TS-isomorphism types are bounded by $(r+1)^k \cdot (c \cdot (a \cdot k)^{r(h+1)})^{c \cdot (a \cdot k)^{r(h+1)}}$. By an analysis similar to that for acyclic schemas, we can show that

Theorem 72. *For HAS Γ with linearly-cyclic schema and HLTL-FO property φ_f over Γ, $\Gamma \models \varphi_f$ can be checked in $O(2\text{-}\exp(N^{c_1 \cdot h}))$ deterministic space where $c_1 = O(r)$. If Γ does not contain artifact relations, then $\Gamma \models \varphi_f$ can be checked in $O(N^{c_2 \cdot h})$ deterministic space where $c_2 = O(r)$.*

Cyclic schema. If DB is cyclic, then each relation in FK has at most a outgoing edges so $F(\delta)$ is bounded by a^δ. So $h(T) = O(k \cdot a^\delta)$ where $\delta = 1$ if T is a leaf task and $\delta = \max_{T_c \in child(T)} h(T_c)$ otherwise. Solving the recursion yields $h(T) = h\text{-}\exp(O(N))$. By pursuing the analysis similarly to the above, we obtain the following:

Theorem 73. *For HAS Γ with cyclic schema and HLTL-FO property φ_f over Γ, $\Gamma \models \varphi_f$ can be checked in $(h+2)\text{-}\exp(O(N))$ deterministic space. If Γ does not contain artifact relations, then $\Gamma \models \varphi_f$ can be checked in $h\text{-}\exp(O(N))$ deterministic space.*

To summarize, the schema type determines the size of the navigation set, and hence the complexity of verification, as follows (h the height of the task hierarchy and N the size of (Γ, φ_f)).

- Acyclic schemas are the least general, yet sufficiently expressive for many applications. A special case of acyclic schema is the Star schema [42, 64] (or Snowflake schema) which is widely used in modeling business process data. For fixed acyclic schemas, the navigation sets have constant depth.

- Linearly-cyclic schemas extend acyclic schemas but yield higher complexity. In general, the size of the navigation set is exponential in h and polynomial in N. Linearly-cyclic schemas allow very simple cyclic foreign key relations such as a single Employee-Manager relation. They include important special cases such as schemas where each relation has at most one foreign key attribute.
- Cyclic schemas allow arbitrary foreign keys but also come with much higher complexity (a tower of exponentials of height h), as the size of navigation sets become hyper-exponential wrt h.

D VERIFICATION WITH ARITHMETIC

We next outline the technical details for verification with arithmetic, starting with a review of quantifier elimination and real algebraic geometry.

D.1 Review of Quantifier Elimination

The quantifier elimination (QE) problem for the reals can be stated as follows.

Definition 74. *For real variables $Y = \{y_i\}_{1 \le i \le l}$ and a formula $\Phi(Y)$ of the form $(Q_1 x_1) \dots (Q_k x_k)$ $F(y_1 \dots y_l, x_1 \dots x_k)$ where $Q_i \in \{\exists, \forall\}$ and $F(y_1 \dots y_l, x_1 \dots x_k)$ is a Boolean combination of polynomial inequalities with integer coefficients, the quantifier elimination problem is to output a quantifier-free formula $\Psi(Y)$ such that for every $Y \in \mathbb{R}^l$, $\Phi(Y)$ is true iff $\Psi(Y)$ is true.*

The best known algorithm for solving the QE problem for the reals has time and space complexity doubly-exponential in the number of quantifier alternations and singly-exponential in the number of variables. When applying QE in verification of HAS, we are only interested in formulas that are existentially quantified. According to Algorithm 14.6 of [5], the result for this special case can be stated as follows:

Theorem 75. *For existentially quantified formula $\Phi(Y)$, an equivalent quantifier-free formula $\Psi(Y)$ can be computed in time and space $(s \cdot d)^{O(k)O(l)}$, where s is the number of polynomials in Φ, d is the maximum degree of the polynomials, k is the quantifier rank of Φ and $l = |Y|$.*

Note that in the special case when $l = 0$, quantifier elimination simply checks satisfiability. Thus we have:

Corollary 76. *Satisfiability over the reals of a Boolean combination Φ of polynomial inequalities with integer coefficients can be decided in time and space $(s \cdot d)^{O(k)}$, where s is the number of polynomials in Φ, d is the maximum degree of the polynomials, and k is the number of variables in Φ.*

Also in Section 14.3 of [5], it is shown that if the bit-size of coefficients in Φ is bounded by τ, then the bit-size of coefficients in Ψ is bounded by $\tau \cdot d^{O(k)O(l)}$.

D.2 Review of General Real Algebraic Geometry

We next review a classic result in general real algebraic geometry.

Definition 77. *For a given set of polynomials $\mathcal{P} = \{P_1, \dots, P_s\}$ over k variables $\{x_i\}_{1 \le i \le k}$, a sign condition of $\mathcal{P}$ is a mapping $\sigma : \mathcal{P} \mapsto \{-1, 0, +1\}$. We denote by $\kappa(\sigma, \mathcal{P})$ the semialgebraic set $\{x | x \in \mathbb{R}^k, sign(P(x)) = \sigma(P), \forall P \in \mathcal{P}\}$[7] called the* cell *of the sign condition σ for $\mathcal{P}$.*

[7] The sign function $sign(x)$ equals -1 if $x < 0$, 0 if $x = 0$ and 1 if $x > 0$.

We use the following result from [6, 39]:

Theorem 78. *Given a set of polynomials $\mathcal{P}$ with integer coefficients over k variables $\{x_i\}_{1\le i\le k}$, the number of distinct non-empty cells, namely $\#\{\sigma : \mathcal{P} \mapsto \{-1,0,+1\} \mid \kappa(\sigma,\mathcal{P}) \neq \emptyset\}$, is at most $(s\cdot d)^{O(k)}$, where $s = |\mathcal{P}|$ and d is the maximum degree of polynomials in $\mathcal{P}$.*

Given a set of polynomials $\mathcal{P}$, we can use the following naive approach to compute the set of sign conditions resulting in non-empty cells. We simply enumerate sign conditions of $\mathcal{P}$ and discard sign conditions that results in empty cells or cells equivalent to any recorded sign conditions known to be non-empty. Checking whether a cell is empty and checking whether two cells are equivalent can be reduced to checking satisfiability of a formula of polynomial inequalities. By Corollary 76, this naive approach takes space $(s\cdot d)^{O(k)}$.

Theorem 79. *Given a set of polynomials $\mathcal{P}$ over $\{x_i\}_{1\le i\le k}$, the set of non-empty cells $\{\sigma : \mathcal{P} \mapsto \{-1,0,+1\} \mid \kappa(\sigma,\mathcal{P}) \neq \emptyset\}$ defined by $\mathcal{P}$ can be computed in space $(s\cdot d)^{O(k)}$ where $s = |\mathcal{P}|$ and d is the maximum degree of polynomials in $\mathcal{P}$.*

D.3 Cells for Verification

Intuitively, in order to handle arithmetic in our verification framework, we need to extend each isomorphism type τ with a set of polynomial inequality constraints over the set of numeric expressions in the extended navigation set $\mathcal{E}^+_T$.

We say that an expression e is numeric if $e = x$ for some numeric variable x or $e = x_R.w$ and the last attribute of w is numeric. For each task T, we denote by $\mathcal{E}^T_{\mathbb{R}}$ the set of numeric expressions of T where for each $x_R.w \in \mathcal{E}^T_{\mathbb{R}}$, $|w| \le h(T)$.

The constraints over the numeric expressions are represented by a non-empty cell κ (formally defined below). When a service is applied, the arithmetic parts of the conditions are evaluated against κ. And for every transition $I \xrightarrow{\sigma'} I'$ where κ, κ' are the cells of I, I' respectively, if any variables are modified by the transition, then the projection of κ' onto the preserved numeric expressions has to *refine* the projection of κ onto the preserved numeric expressions. Similar compatibility checks are required when a child task returns to its parent.

We introduce some more notation. For every $T \in \mathcal{H}$, we consider polynomials in the polynomial ring $\mathbb{Z}[\mathcal{E}^T_{\mathbb{R}}]$. For each polynomial P, we denote by $var(P)$ the set of numeric expressions mentioned in P and for a set of polynomials $\mathcal{P}$, we denote by $var(\mathcal{P})$ the set $\bigcup_{P\in\mathcal{P}} var(P)$. For $\mathcal{P} \subset \mathbb{Z}[\mathcal{E}^T_{\mathbb{R}}]$ and $\mathcal{E} \subseteq \mathcal{E}^T_{\mathbb{R}}$, we denote by $\mathcal{P}|\mathcal{E}$ the set of polynomials $\{P | P \in \mathcal{P}, var(P) \subseteq \mathcal{E}\}$.

We next define the cells used in our verification algorithm. At task T, for a set of numeric expressions $\mathcal{E} \subseteq \mathcal{E}^T_{\mathbb{R}}$ and a set of polynomials $\mathcal{P}$ where $var(\mathcal{P}) \subseteq \mathcal{E}$, we define the cells over $(\mathcal{E},\mathcal{P})$ as follows.

Definition 80. *A cell κ over $(\mathcal{E},\mathcal{P})$ is a subset of $\mathbb{R}^{|\mathcal{E}|}$ for which there exists a sign condition σ of $\mathcal{P}$ such that $\kappa = \kappa(\sigma,\mathcal{P})$.*

For $\mathcal{P} \subset \mathbb{Z}[\mathcal{E}^T_{\mathbb{R}}]$, we denote by $\mathcal{K}(\mathcal{P},\mathcal{E})$ the set of cells over $(\mathcal{E},\mathcal{P}|\mathcal{E})$. Namely, $\mathcal{K}(\mathcal{P},\mathcal{E}) = \{\kappa(\sigma,\mathcal{P}|\mathcal{E}) | \sigma \in \mathcal{P}|\mathcal{E} \mapsto \{-1,0,+1\}\}$. And we denote by $\mathcal{K}(\mathcal{P})$ the set of cells $\bigcup_{\mathcal{E}\subseteq\mathcal{E}^T_{\mathbb{R}}} \mathcal{K}(\mathcal{P},\mathcal{E})$.

Compatibility between cells is tested using the notion of refinement. Intuitively, a cell κ refines another cell κ' if κ can be obtained by adding extra numeric expressions and/or constraints to κ'. Formally,

Definition 81. *For cell κ over $(\mathcal{E},\mathcal{P})$ and cell κ' over $(\mathcal{E}',\mathcal{P}')$ where $\kappa = \kappa(\sigma,\mathcal{P})$ and $\kappa' = \kappa(\sigma',\mathcal{P}')$, we say that κ refines κ', denoted by $\kappa \sqsubseteq \kappa'$, if $\mathcal{E}' \subseteq \mathcal{E}$, $\mathcal{P}' \subseteq \mathcal{P}$ and $\sigma|\mathcal{P}' = \sigma'$. Note that if $\mathcal{E} = \mathcal{E}'$, then $\kappa \sqsubseteq \kappa'$ iff $\kappa \subseteq \kappa'$.*

We next define the projection of a cell onto a set of variables. For each cell κ over $(\mathcal{E},\mathcal{P})$ where $\mathcal{E} \subseteq \mathcal{E}_{\mathbb{R}}^T$ and variables $\bar{x} \subseteq \bar{x}^T$, the projection of κ onto $\bar{x}$, denoted by $\kappa|\bar{x}$, is defined to be the projection of κ onto the expressions $\mathcal{E}|\bar{x}$ where $\mathcal{E}|\bar{x} = \{e \in \mathcal{E} | e = x_R.w \vee e = x, x \in \bar{x}\}$. By the Tarski-Seidenberg theorem [61], $\kappa|\bar{x}$ is a union of disjoint cells. Also, the projections $\kappa|\bar{x}$ can be obtained by quantifier elimination. Let $\Phi(\kappa)$ be the conjunctive formula defining κ using polynomials in $\mathcal{P}$. Then by treating $\mathcal{E}|\bar{x}$ as the set of free variables, the formula $\Psi(\kappa)$ obtained by eliminating $\mathcal{E} - \mathcal{E}|\bar{x}$ from $\Phi(\kappa)$ defines $\kappa|\bar{x}$. We denote by $\mathtt{proj}(\kappa,\bar{x})$ the set of polynomials mentioned in $\Psi(\kappa)$. It is easy to see that $\kappa|\bar{x}$ is a union of cells over $(\mathcal{E}|\bar{x}, \mathtt{proj}(\kappa,\bar{x}))$.

The following notation is useful for checking compatibility between a cell and the projection of another cell: we define that a cell κ refines another cell κ' wrt to projection to $\bar{x}$, denoted as $\kappa \sqsubseteq_{\bar{x}} \kappa'$, if there exists a cell $\tilde{\kappa} \subseteq \kappa'|\bar{x}$ such that $\kappa \sqsubseteq \tilde{\kappa}$.

Example 82. *We illustrate the key notations of cells, projection and refinement in Figure 17. Consider a task T with only two numeric variables $\{x, y\}$ where x is an input variable. Figure 17(a) illustrates a cell κ_1 defined by 3 half-planes L_1, L_2 and L_3, which are polynomial constraints from the task T. When an internal service of T is applied, the pre-condition is first evaluated against $\{L_1, L_2, L_3\}$, then the value of x is propagated, so the triangular region κ_1 is projected to the x-axis and comes the cell κ_2 illustrated in Figure 17(b). Finally, the post-condition is evaluated thus the resulting cell κ_3 is a refinement of κ_2 with the additional constraints L_4 and L_5 from the post-condition.*

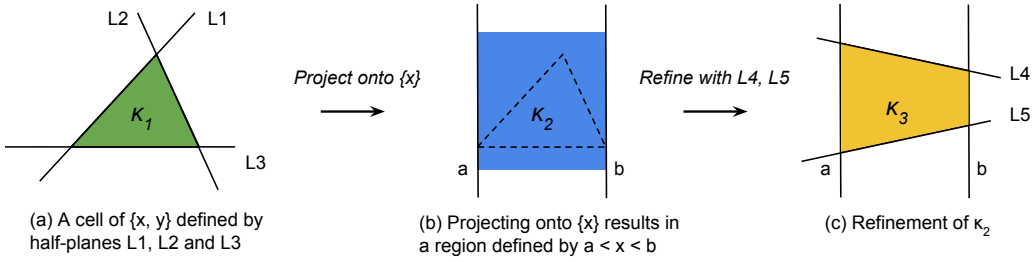

Fig. 17. Illustration of cells, projection, and refinement

Finally, we introduce notation relative to variable passing between parent task and child task. For each task T and $T_c \in child(T)$, we denote by $\mathcal{E}_{\mathbb{R}}^{T_c \to T}$ the set of numeric expressions $\{e|e \in \mathcal{E}_{\mathbb{R}}^{T_c}, e = x \vee e = x_R.w, x \in \bar{x}_{\text{in}}^{T_c} \cup \bar{x}_{\text{out}}^{T_c}\}$. In other words, $\mathcal{E}_{\mathbb{R}}^{T_c \to T}$ is the subset of expressions in $\mathcal{E}_{\mathbb{R}}^{T_c}$ connected with expressions in $\mathcal{E}_{\mathbb{R}}^{T}$ by calls/returns of T_c. Let f_{in}, f_{out} be the input and output mapping between T and T_c. For each expression $e \in \mathcal{E}_{\mathbb{R}}^{T_c \to T}$, we define $e^{T_c \to T}$ to be an expression in $\mathcal{E}_{\mathbb{R}}^{T}$ as follows. If $e = x$, then $e^{T_c \to T} = (f_{in} \circ f_{out})(x)$. If $e = x_R.w$, then $e^{T_c \to T} = ((f_{in} \circ f_{out})(x))_R.w$. For a set of variables $\mathcal{E} \subseteq \mathcal{E}_{\mathbb{R}}^{T_c \to T}$, we define $\mathcal{E}^{T_c \to T}$ to be $\{e^{T_c \to T} | e \in \mathcal{E}\}$. For a polynomial P over $\mathcal{E}_{\mathbb{R}}^{T_c \to T}$ where $T_c \in child(T)$, we denote by $P^{T_c \to T}$ the polynomial obtained by replacing in P each numeric expression e with $e^{T_c \to T}$. For a cell κ of T_c where $\kappa = \kappa(\sigma,\mathcal{P})$ and $var(P) \subseteq \mathcal{E}_{\mathbb{R}}^{T_c \to T}$ for every $P \in \mathcal{P}$, we let $\kappa^{T_c \to T}$ to be the cell of T which equals $\kappa(\sigma',\mathcal{P}')$, where $\mathcal{P}' = \{P^{T_c \to T} | P \in \mathcal{P}\}$ and σ' is a sign condition over $\mathcal{P}'$ such that $\sigma'(P^{T_c \to T}) = \sigma(P)$ for every $P \in \mathcal{P}$.

D.4 Hierarchical Cell Decomposition

We now introduce the Hierarchical Cell Decomposition. Intuitively, for each task T, we would like to compute a set of polynomials $\mathcal{P}$ and a set of cells $\mathcal{K}_T$ such that for each subset $\mathcal{E}$ of $\mathcal{E}^T_{\mathbb{R}}$, the set of cells over $(\mathcal{E}, \mathcal{P}|\mathcal{E})$ in $\mathcal{K}_T$ is a partition of $\mathbb{R}^{|\mathcal{E}|}$.

The set of cells $\mathcal{K}_T$ satisfies the property that for the set of polynomials $\mathcal{P}$ mentioned at any condition of T in the specification Γ and HLTL-FO property φ_f, each cell $\kappa \in \mathcal{K}_T$ uniquely defines the sign condition of $\mathcal{P}$. This allows us to compute the signs of any polynomial in any condition in the local symbolic runs. In addition, for each pair of cells $\kappa, \kappa' \in \mathcal{K}_T$, we require that the projection of κ and κ' to the input variables $\bar{x}^T_{\mathsf{in}}$ (and $\bar{x}^T_{\mathsf{in}} \cup \bar{s}^T$) be disjoint or identical. So to check whether two cells κ and κ' of two consecutive symbolic instances in a local symbolic run are compatible when applying an internal service, we simply need to check whether their projections on $\bar{x}^T_{\mathsf{in}}$ are equal (note that refinement is implied by equality). Finally, for each child task T_c of T, for each cell $\kappa \in \mathcal{K}_T$ and $\kappa' \in \mathcal{K}_{T_c}$, κ uniquely defines the sign condition for the set of polynomials that defines $\kappa'|\bar{x}^{T_c}_{\mathsf{in}}$ and $\kappa'|(\bar{x}^{T_c}_{\mathsf{out}} \cup \bar{x}^{T_c}_{\mathsf{in}})$. This reduces to the problem of checking cell refinements when child tasks are called or return.

The Hierarchical Cell Decomposition is formally defined as follows.

Definition 83. *The Hierarchical Cell Decomposition associated to an artifact system $\mathcal{H}$ and property φ_f is a collection $\{\mathcal{K}_T\}_{T\in\mathcal{H}}$ of sets of cells, such that for each $T \in \mathcal{H}$, $\mathcal{K}_T = \mathcal{K}(\mathcal{P}'_T)$, where the set of polynomials $\mathcal{P}'_T$ is defined as follows. First, let $\mathcal{P}_T$ consist of the following:*

(1) *all polynomials mentioned in any condition over $\bar{x}^T$ in Γ and the property φ_f,*

(2) *polynomials $\{e|e \in \mathcal{E}^T_{\mathbb{R}}\} \cup \{e - e'|e, e' \in \mathcal{E}^T_{\mathbb{R}}\}$, and*

(3) *for every $T_c \in child(T)$ and subset $\bar{x} \subseteq \bar{x}^{T_c}_{\mathsf{out}}$, the set of polynomials $\{P^{T_c\to T}|P \in \mathtt{proj}(\kappa, \bar{x}^{T_c}_{\mathsf{in}} \cup \bar{x}), \kappa \in \mathcal{K}_{T_c}\}$.*

Next, let $\mathcal{P}^s_T = \mathcal{P}_T \cup \bigcup_{c\in\mathcal{K}(\mathcal{P}_T)} \mathtt{proj}(\kappa, \bar{x}^T_{\mathsf{in}} \cup \bar{s}^T)$. Finally, $\mathcal{P}'_T = \mathcal{P}^s_T \cup \bigcup_{c\in\mathcal{K}(\mathcal{P}^s_T)} \mathtt{proj}(\kappa, \bar{x}^T_{\mathsf{in}})$.

Intuitively, when T is a leaf task, the set of cells $\mathcal{K}_T$ is constructed simply from (1) all polynomials within services of T and the parts of HLTL-FO property φ_f related T and (2) all possible tests of equality. When T is a non-leaf task, in addition to (1) and (2), we also need to take into account the constraints propagated to T from each child task T_c of T, which can be obtained by projecting cells in $\mathcal{K}_{T_c}$ onto T_c's input/output variables, resulting in the set of polynomials in (3).

The Hierarchical Cell Decomposition satisfies the following property, as desired.

Lemma 84. *Let T be a task and $\mathcal{P}'_T$ as above. For every pair of cells $\kappa_1, \kappa_2 \in \mathcal{K}_T$, and $\bar{x} = (\bar{x}^T_{\mathsf{in}} \cup \bar{s}^T)$ or $\bar{x} = \bar{x}^T_{\mathsf{in}}$, if $\kappa_1 \in \mathcal{K}(\mathcal{P}'_T, \mathcal{E}_1)$ and $\kappa_2 \in \mathcal{K}(\mathcal{P}'_T, \mathcal{E}_2)$ where $\mathcal{E}_1|\bar{x} = \mathcal{E}_2|\bar{x}$, then $\kappa_1|\bar{x}$ and $\kappa_2|\bar{x}$ are either equal or disjoint.*

Proof. We prove the lemma for the case when $\bar{x} = \bar{x}^T_{\mathsf{in}}$. The proof is similar for $\bar{x} = \bar{x}^T_{\mathsf{in}} \cup \bar{s}^T$.

Let $\tilde{\mathcal{P}}^s_T = \bigcup_{c\in\mathcal{K}(\mathcal{P}^s_T)} \mathtt{proj}(\kappa, \bar{x}^T_{\mathsf{in}})$. For each cell $\kappa \in \mathcal{K}(\mathcal{P}'_T, \mathcal{E})$, since $\mathcal{P}'_T|\mathcal{E} = (\mathcal{P}^s_T|\mathcal{E}) \cup (\tilde{\mathcal{P}}^s_T|\mathcal{E})$ as $\mathcal{P}'_T = \mathcal{P}^s_T \cup \tilde{\mathcal{P}}^s_T$, there exist $\kappa_1 \in \mathcal{K}(\mathcal{P}^s_T, \mathcal{E})$ and $\kappa_2 \in \mathcal{K}(\tilde{\mathcal{P}}^s_T, \mathcal{E})$ such that $\kappa = \kappa_1 \cap \kappa_2$. Then consider $\kappa|\bar{x}^T_{\mathsf{in}}$. Since all polynomials in $\tilde{\mathcal{P}}^s_T$ are over expressions of $\bar{x}^T_{\mathsf{in}}$, we have $\kappa|\bar{x}^T_{\mathsf{in}} = (\kappa_1 \cap \kappa_2)|\bar{x}^T_{\mathsf{in}} = (\kappa_1|\bar{x}^T_{\mathsf{in}}) \cap \kappa_2$. And by definition, $\mathtt{proj}(\kappa_1, \bar{x}^T_{\mathsf{in}}) \subseteq \tilde{\mathcal{P}}^s_T$, so κ_2 uniquely defines the sign conditions for

$\texttt{proj}(\kappa_1, \bar{x}^T_{\text{in}})$, which means that either $\kappa_2 \cap \kappa_1|\bar{x}^T_{\text{in}} = \emptyset$ or $\kappa_2 \subseteq \kappa_1|\bar{x}^T_{\text{in}}$. And as $\kappa_2 \cap \kappa_1|\bar{x}^T_{\text{in}} = \kappa|\bar{x}^T_{\text{in}}$ is non-empty, $\kappa|\bar{x}^T_{\text{in}} = \kappa_2$.

Therefore, for every $\kappa_1 \in \mathcal{K}(\mathcal{P}'_T, \mathcal{E}_1)$ and $\kappa_2 \in \mathcal{K}(\mathcal{P}'_T, \mathcal{E}_2)$ where $\mathcal{E}_1|\bar{x}^T_{\text{in}} = \mathcal{E}_2|\bar{x}^T_{\text{in}} = \mathcal{E}$, there exist cells $\tilde{\kappa}_1, \tilde{\kappa}_2 \in \mathcal{K}(\mathcal{P}'_T, \mathcal{E})$ such that $\kappa_1|\bar{x}^T_{\text{in}} = \tilde{\kappa}_1$ and $\kappa_2|\bar{x}^T_{\text{in}} = \tilde{\kappa}_2$. Since $\tilde{\kappa}_1$ and $\tilde{\kappa}_2$ are either disjoint or equal, $\kappa_1|\bar{x}^T_{\text{in}}$ and $\kappa_2|\bar{x}^T_{\text{in}}$ are also either disjoint or equal. □

From the above lemma, the following Corollary is obvious:

Corollary 85. *For every task T and $\kappa \in \mathcal{K}_T$, $\kappa|\bar{x}^T_{\text{in}}$ and $\kappa|(\bar{x}^T_{\text{in}} \cup \bar{s}^T)$ are single cells in $\mathcal{K}_T$.*

In view of the corollary, we use the notations of single-cell operators (projection, refinement, etc.) on $\kappa|\bar{x}^T_{\text{in}}$ and $\kappa|(\bar{x}^T_{\text{in}} \cup \bar{s}^T)$ in the rest of our discussion.

To be able to connect with child tasks, we show the following property of $\mathcal{K}_T$:

Lemma 86. *For all tasks T and T_c where $T_c \in child(T)$, and every cell $\kappa_1 \in \mathcal{K}_T$ and $\kappa_2 \in \mathcal{K}_{T_c}$ where $\kappa_1 \in \mathcal{K}(\mathcal{P}'_T, \mathcal{E}_1)$ and $\kappa_2 \in \mathcal{K}(\mathcal{P}'_{T_c}, \mathcal{E}_2)$, for each set of variables $\bar{x} = \bar{x}^T_{T_c\uparrow} \cup \bar{y}$ where $\bar{y}$ is some subset of $\bar{x}^T_{T_c\downarrow}$, if $\mathcal{E}_1|\bar{x} = (\mathcal{E}_2)^{T_c\to T}|\bar{x}$, then either (1) $\kappa_1 \sqsubseteq_{\bar{x}} (\kappa_2)^{T_c\to T}$ or (2) $\kappa_1|\bar{x}$ is disjoint from $(\kappa_2)^{T_c\to T}|\bar{x}$.*

Proof. Denote by $\mathcal{P}^{\bar{x}}_{T_c}$ the set of polynomials $\{P^{T_c\to T}|P \in \texttt{proj}(\kappa, \bar{x}), \kappa \in \mathcal{K}_{T_c}\}$. For each cell $\kappa_1 \in \mathcal{K}(\mathcal{P}'_T, \mathcal{E}_1)$, there exists $\tilde{\kappa}_1 \in \mathcal{K}(\mathcal{P}^{\bar{x}}_{T_c}, \mathcal{E}_1)$ such that $\kappa_1 \subseteq \tilde{\kappa}_1$. For each cell $\kappa_2 \in \mathcal{K}(\mathcal{P}'_{T_c}, \mathcal{E}_2)$, as $\mathcal{E}_1|\bar{x} = (\mathcal{E}_2)^{T_c\to T}|\bar{x}$, $(\kappa_2)^{T_c\to T}|\bar{x}$ is a union of cells in $\mathcal{K}(\mathcal{P}^{\bar{x}}_{T_c}, \mathcal{E}_1)$. So either $\tilde{\kappa}_1$ is disjoint with or contained in $(\kappa_2)^{T_c\to T}|\bar{x}$. If $\tilde{\kappa}_1$ and $(\kappa_2)^{T_c\to T}|\bar{x}$ are disjoint, then $(\kappa_2)^{T_c\to T}|\bar{x}$ and $\kappa_1|\bar{x}$ are disjoint. If $\tilde{\kappa}_1 \subseteq (\kappa_2)^{T_c\to T}|\bar{x}$, then we have $\kappa_1 \sqsubseteq \tilde{\kappa}_1 \subseteq (\kappa_2)^{T_c\to T}|\bar{x}$ so $\kappa_1 \sqsubseteq_{\bar{x}} (\kappa_2)^{T_c\to T}$. □

D.5 Extended Isomorphism Types

Given the Hierarchical Cell Decomposition $\{\mathcal{K}_T\}_{T\in\mathcal{H}}$, we can extend our notion of isomorphism type to support arithmetic.

Definition 87. *For navigation set $\mathcal{E}_T$, equality type $\sim_\tau$ over $\mathcal{E}^+_T$ and $\kappa \in \mathcal{K}_T$, the triple $\tau = (\mathcal{E}_T, \sim_\tau, \kappa)$ is an extended T-isomorphism type if*

- *$(\mathcal{E}_T, \sim_\tau)$ is a T-isomorphism type, and*
- *$\kappa = \kappa(\sigma, \mathcal{P}'_T|(\mathcal{E}^T_{\mathbb{R}} \cap \mathcal{E}^+_T))$ for some sign condition σ of $\mathcal{P}'_T|(\mathcal{E}^T_{\mathbb{R}} \cap \mathcal{E}^+_T)$ such that for every numeric expression $e, e' \in \mathcal{E}^+_T$, $e \sim_\tau e'$ iff $\sigma(e - e') = 0$ and $e \sim_\tau 0$ iff $\sigma(e) = 0$.*

For each condition π over $\bar{x}^T$ and extended T-isomorphism type τ, $\tau \models \pi$ is defined as follows. For each polynomial inequality "$P \circ 0$" in π where $\circ \in \{<, >, =\}$, $P \circ 0$ is true iff $\sigma(P) \circ 0$ where σ is the sign condition of κ. The rest of the semantics is the same as in normal T-isomorphism type.

The projection of an extended T-isomorphism type τ on $\bar{x}^T_{\text{in}}$ and $\bar{x}^T_{\text{in}} \cup \bar{s}^T$ is defined in the obvious way. For $\tau = (\mathcal{E}_T, \sim_\tau, \kappa)$, we define that $\tau|\bar{x} = (\mathcal{E}_T|\bar{x}, \sim_\tau |\bar{x}, \kappa|\bar{x})$ for $\bar{x} = \bar{x}^T_{\text{in}}$ or $\bar{x} = \bar{x}^T_{\text{in}} \cup \bar{s}^T$. The projection of τ on $\bar{x}^T_{\text{in}}$ and $\bar{x}^T_{\text{in}} \cup \bar{s}^T$ up to length k is defined analogously. The projection of every extended T-isomorphism type on $\bar{x}^T_{\text{in}} \cup \bar{s}^T$ is an extended TS-isomorphism type.

To extend the definitions of local symbolic run and symbolic tree of runs, we first replace T-isomorphism type with extended T-isomorphism type and TS-isomorphism type with extended

TS-isomorphism type in the original definitions. The semantics is extended with the following rules.

For two symbolic instances I and I' where the cell of I is κ and the cell of I' is κ', I' is a valid successor of I by applying service σ' if the following conditions hold in addition to the original requirements:

- if σ' is an internal service, then $\kappa|\bar{x}^T_{\text{in}} = \kappa'|\bar{x}^T_{\text{in}}$.
- if σ' is an opening service of $T_c \in child(T)$ or closing service of T, then $\kappa = \kappa'$.
- if σ' is a closing service of $T_c \in child(T)$, then $\kappa' \sqsubseteq \kappa$.

The counters $\bar{c}$ are updated as in transitions between symbolic instances without arithmetic. Each dimension of $\bar{c}$ corresponds to an extended TS-isomorphism type.

For each local symbolic run $\tilde{\rho}_T = (\tau_{in}, \tau_{out}, \{(I_i, \sigma_i)\}_{0 \le i < \gamma})$, the following are additionally satisfied:

- $\kappa_{in} = \kappa_0|\bar{x}^T_{\text{in}}$, where κ_{in} is the cell of τ_{in} and κ_0 is the cell of τ_0;
- if $\tau_{out} \neq \bot$, then $\kappa_{out} \sqsubseteq_{\bar{x}^T_{\text{in}} \cup \bar{x}^T_{\text{out}}} \kappa_{\gamma-1}$, where κ_{out} is the cell of τ_{out} and $\kappa_{\gamma-1}$ is the cell of $\tau_{\gamma-1}$.

In a symbolic tree of runs ***Sym***, for every two local symbolic runs $\tilde{\rho}_T = (\tau_{in}, \tau_{out}, \{(I_i, \sigma_i)\}_{0 \le i < \gamma})$ and $\tilde{\rho}_{T_c} = (\tau'_{in}, \tau'_{out}, \{(I'_i, \sigma'_i)\}_{0 \le i < \gamma'})$ where $T_c \in child(T)$, if $\tilde{\rho}_{T_c}$ is connected to $\tilde{\rho}_T$ by an edge labeled with index i, then the following conditions must be satisfied in addition to the original requirements:

- for the cell κ_i of symbolic instance I_i and the cell κ_{in} of τ'_{in}, $\kappa_i \sqsubseteq \kappa_{in}^{T_c \to T}$.
- if $\tilde{\rho}_{T_c}$ is a returning local symbolic run, then for the cells κ_{out} of τ'_{out} and κ_j of I_j where j is the smallest index such that $\sigma_j = \sigma^c_{T_c}$ and $j > i$, we have that $\kappa_j \sqsubseteq_{\bar{x}_{\texttt{null}}} \kappa_{out}^{T_c \to T}$, where $\bar{x}_{\texttt{null}} = \{x | x \in \bar{x}^T_{T_c\uparrow}, x \sim_{\tau_{j-1}} \texttt{null}\}$.

D.6 Connecting Actual Runs with Symbolic Runs

We next show that the connection between actual runs and symbolic runs established in Theorem 37 still holds for the extended local and symbolic runs. The structure of the proof is the same, so we only state the necessary modifications needed to handle arithmetic.

D.6.1 From Trees of Local Runs to Symbolic Trees of Runs. Given a tree of local runs ***Tree***, the construction of a corresponding symbolic tree of runs ***Sym*** can be done as follows. We first construct ***Sym*** from ***Tree*** without the cells following the construction described in the proof of the only-if part of Theorem 37. Then for each task T and symbolic instance I with extended isomorphism type τ in some local symbolic run of T, let $\mathcal{E}$ be the set of numeric expressions in τ and $\upsilon : \mathcal{E} \mapsto \mathbb{R}$ the valuation of $\mathcal{E}$ at I. Then the cell κ of I is chosen to be the unique cell in $\mathcal{K}(\mathcal{P}'_T, \mathcal{E})$ that contains υ. For cells κ and κ' of two consecutive symbolic instances I and I' where the service that leads to I' is σ',

- if σ' is an internal service, by Lemma 84, as $\kappa|\bar{x}^T_{\text{in}}$ and $\kappa'|\bar{x}^T_{\text{in}}$ overlaps, we have $\kappa|\bar{x}^T_{\text{in}} = \kappa'|\bar{x}^T_{\text{in}}$,
- if σ' is an opening service, $\kappa = \kappa'$ is obvious, and
- if σ' is a closing service, let $\mathcal{E}$ be the numeric expressions of κ and $\mathcal{E}'$ be the numeric expressions of κ'. We have $\mathcal{E} \subseteq \mathcal{E}'$ so $\mathcal{P}'_T|\mathcal{E} \subseteq \mathcal{P}'_T|\mathcal{E}'$. So κ' can be written as $\kappa_1 \cap \kappa_2$ where

$\kappa_1 \in \mathcal{K}(\mathcal{P}'_T, \mathcal{E})$ and $\kappa_2 \in \mathcal{K}(\mathcal{P}'_T, \mathcal{E}' - \mathcal{E})$. As the values of the preserved numeric expressions are equal in the two consecutive instances, we have $\kappa_1 = c$ so $\kappa \sqsubseteq \kappa'$.

Thus, each local symbolic run in ***Sym*** is valid. Following a similar analysis, one can verify that for every two connected local symbolic runs $\tilde{\rho}_T$ and $\tilde{\rho}_{T_c}$, the conditions for symbolic tree of runs stated in Appendix D.5 are satisfied due to Lemma 86.

D.6.2 From Symbolic Trees of Runs to Trees of Local Runs. Given a symbolic tree of runs ***Sym***, we construct the tree of local runs ***Tree*** as follows. Recall that in the original proof, for each local symbolic run $\tilde{\rho}_T$, we construct the global isomorphism type Λ of $\tilde{\rho}_T$ and use Λ to construct the local run ρ_T and database instance D_T. With arithmetic, the construction of Λ remains unchanged but we use a different construction for ρ_T and D_T.

To construct ρ_T and D_T, we first define a sequence of mappings $\{p_i\}_{0 \le i < \gamma}$ from the sequence of cells $\{\kappa_i\}_{0 \le i < \gamma}$ of $\tilde{\rho}_T$ where each p_i is a mapping from $\mathcal{E}^+_T \cap \mathcal{E}^T_{\mathbb{R}}$ to $\mathbb{R}$ and $\mathcal{E}^+_T$ is the extended navigation set of τ_i. Note that each p_i can be also viewed as a point in κ_i. The sequence of mappings $\{p_i\}_{0 \le i < \gamma}$ determines the values of numeric expressions, as we shall see next. For each mapping p whose domain is the set of numeric expressions $\mathcal{E}$, we denote by $p|\bar{x}$ the projection of p to $\mathcal{E} \cap (\bar{x} \cup \{x_R.w | x \in \bar{x}\})$. Then $\{p_i\}_{0 \le i < \gamma}$ is constructed as follows:

- First, we pick an arbitrary point (mapping) p_{in} from κ_{in} where κ_{in} is the cell of the input isomorphism type of $\tilde{\rho}_T$.
- Then, for each equivalence class $\mathcal{L}$ of life cycles in $\tilde{\rho}_T$, let $\kappa_{\mathcal{L}}$ be the cell of the last symbolic instances in the last dynamic segments of life cycles in $\mathcal{L}$. Pick a mapping $p_{\mathcal{L}} \in \kappa_{\mathcal{L}}$ such that $p_{\mathcal{L}}|\bar{x}^T_{\text{in}} = p_{in}$. Such a mapping always exists because, by Lemma 84, for each $0 \le i < \gamma$, $\kappa_i|\bar{x}^T_{\text{in}} = \kappa_{in}$.
- Next, for each equivalence class $\mathcal{S}$ of segments in $\mathcal{L}$, let $\kappa_{\mathcal{S}}$ be the cell of the last symbolic instance in segments in $\mathcal{S}$. Pick a mapping $p_{\mathcal{S}}$ from $\kappa_{\mathcal{S}}$ such that $p_{\mathcal{S}}|(\bar{x}^T_{\text{in}} \cup \bar{s}^T) = p_{\mathcal{L}}|(\bar{x}^T_{\text{in}} \cup \bar{s}^T)$. Such a mapping always exists because for each life cycle $L \in \mathcal{L}$ and I_i in L, $\kappa_{\mathcal{L}}|(\bar{x}^T_{\text{in}} \cup \bar{s}^T) \sqsubseteq \kappa_i|(\bar{x}^T_{\text{in}} \cup \bar{s}^T)$.
- Finally, for each segment $S = \{(I_i, \sigma_i)\}_{a \le i \le b} \in \mathcal{S}$, let $p_b = p_{\mathcal{S}}$, and for $a \le i < b$, let $p_i = p_{i+1}|\bar{x}$ where $\bar{x} = \{x | x \nsim_{\tau_i} \texttt{null}\}$ are the preserved variables from I_i to I_{i+1}. Such mappings always exist because for each $a \le i < b$, $\kappa_{i+1} \sqsubseteq \kappa_i$.

For the sequence of mappings $\{p_i\}_{0 \le i < \gamma}$ constructed above, the following is easily shown:

Lemma 88. *For all local expressions (i, e) and (i', e') in the global isomorphism type Λ, where e and e' are numeric, $(i, e) \sim (i', e')$ implies that $p_i(e) = p_{i'}(e')$.*

Given the above property, we can construct ρ_T and D_T as follows. We first construct ρ_T and D_T as in the case without arithmetic. Then for each equivalence class $[(i, e)]$, we replace the value $[(i, e)]$ in ρ_T and D_T with the value $p_i(e)$. It is clear that Lemmas 42 and 51 still hold since the global equality type in Λ remains unchanged.

To construct the full tree of local runs ***Tree*** from the symbolic tree of runs, we perform the above construction in a top-down manner. For each local symbolic run $\tilde{\rho}_T$, we first construct $\{p_i\}_{0 \le i < \gamma}$ for the root $\tilde{\rho}_{T_1}$ of ***Sym*** using the above construction. Then recursively for each $\tilde{\rho}_T \in$ ***Sym*** and child $\tilde{\rho}_{T_c}$ connected to $\tilde{\rho}_T$ by an edge labeled with index i, we pick a mapping p_{in} from κ_{in} of $\tilde{\rho}_{T_c}$ such that $p_{in}^{T_c \to T} = p_i|\bar{x}^T_{T_c\downarrow}$. And if $\tilde{\rho}_{T_c}$ is a returning run, we pick p_{out} from κ_{out} of $\tilde{\rho}_{T_c}$ such that

$p_{out}^{T_c \to T}|\bar{x}_{\texttt{null}} = p_j|\bar{x}_{\texttt{null}}$ where j is index of the corresponding closing service $\sigma_{T_c}^c$ at $\tilde{\rho}_T$, and $\bar{x}_{\texttt{null}}$ is defined as above.

We next construct $\{p_i\}_{0 \le i < \gamma}$ of $\tilde{\rho}_{T_c}$ similarly to above, except that (1) p_{in} is given, and (2) if $\tilde{\rho}_{T_c}$ is a returning run, then for the equivalence class $\mathcal{L}$ of life cycles where $I_{\gamma-1}$ is contained in some life cycle $L \in \mathcal{L}$, we pick $p_{\mathcal{L}}$ such that $p_{\mathcal{L}}|\bar{x}_{\text{in}}^{T_c} \cup \bar{x}_{\text{out}}^{T_c} = p_{out}$. Then ρ_{T_c} and D_{T_c} are constructed following the above approach. The tree of local runs ***Tree*** is constructed as described in the proof of Theorem 37. Following the same approach, we can show:

Theorem 89. *For every HAS Γ and HLTL-FO property φ_f with arithmetic, there exists a symbolic tree of runs **Sym** accepted by $\mathcal{B}_\varphi$ iff there exists a tree of local runs **Tree** and database D such that **Tree** is accepted by $\mathcal{B}_\varphi$ on D.*

D.7 Complexity of Verification with Arithmetic

Similarly to the analysis in Appendix C, it is sufficient to upper-bound the number of T-and TS-isomorphism types. To do so, we need to bound the size of $\{\mathcal{K}_T\}_{T \in \mathcal{H}}$. By the construction of each $\mathcal{K}_T$ and by Theorem 78, it is sufficient to bound the size of each $\mathcal{P}'_T$.

We denote by l the number of numeric expressions, s the number of polynomials in Γ and φ_f, d the maximum degree of these polynomials, t the maximum bitsize of the coefficients, and h the height of the task hierarchy $\mathcal{H}$. For each task T, we denote by $s(T)$ the number of polynomials in $\mathcal{P}'_T$ and $d(T)$ the maximum degree of polynomials in $\mathcal{P}'_T$.

If T is a leaf task, then $|\mathcal{P}_T| \le s + l^2$. The number of polynomials in $\mathcal{P}_T^s$ is no more than the product of (1) the number of subsets of $\mathcal{E}_{\mathbb{R}}^T$, (2) the maximum number of non-empty cells over $(\mathcal{E}, \mathcal{P}_T|\mathcal{E})$ and (3) the maximum number of polynomials in each $\texttt{proj}(\kappa, \bar{x}_{\text{in}}^T \cup \bar{s}^T)$. By Theorem 75, the number of polynomials is no more than the running time, which is bounded by $((s+l^2)\cdot d)^{O(l^2)}$. Then by Theorem 78, the number of non-empty cells over $(\mathcal{E}, \mathcal{P}_T|\mathcal{E})$ is at most $((s+l^2)\cdot d)^{O(l)}$. Thus, $|\mathcal{P}_T^s| \le ((s+l^2)\cdot d)^{O(l^2)}$. By the same analysis, we obtain that for $\mathcal{P}'_T$, $s(T) = |\mathcal{P}'_T| \le ((s+l^2)\cdot d)^{O(l^4)}$. Similarly, $d(T)$ can be upper-bounded by $((s+l^2)\cdot d)^{O(l^4)}$.

Next, if T is a non-leaf task, we denote by s' the size of $\mathcal{P}_T$ and by d' the maximum degree of polynomials in $\mathcal{P}_T$. We have that $s' \le (s+l^2) + \sum_{T_c \in child(T)} 2^l (s(T_c)\cdot d(T_c))^{O(l^2)} \cdot (s(T_c)\cdot d(T_c))^{O(l)} \le (s+l^2) + (s(T_c)\cdot d(T_c))^{O(l^2)}$, and $d' \le \max_{T_c \in child(T)} (s(T_c)\cdot d(T_c))^{O(l^2)}$.

Following the same analysis as above, we have that both $s(T)$ and $d(T)$ are at most $((s'+l^2)\cdot d')^{O(l^4)}$. By solving the recursion, we obtain that $s(T), d(T) \le ((s+l^2)\cdot d)^{(c\cdot l^6)^h}$ for some constant c. Then by Theorem 78, $|\mathcal{K}_T|$ is at most $(s(T)\cdot d(T))^{O(k)}$. So we have

Lemma 90. *For each task T, the number of cells in $\mathcal{K}_T$ is at most $((s+l^2)\cdot d)^{(c\cdot l^6)^h}$ for some constant c.*

The space used by the verification algorithm with arithmetic is no more than the space needed to pre-compute $\{\mathcal{K}_T\}_{T \in \mathcal{H}}$ plus the space for the VASS (repeated) reachability for each task T. By Theorem 79, for each task T, the set $\mathcal{K}_T$ can be computed in space $O\left(((s+l^2)\cdot d)^{(c\cdot l^6)^h}\right)$.

For VASS (repeated) reachability, according to the analysis in Appendix C, state (repeated) reachability can be computed in $O(h^2 \cdot N^2 \log^2 M \cdot 2^{c \cdot D \log D})$ space ($O(h^2 \cdot N^2 \log^2 M)$ w/o. artifact

relation), where h is the height of $\mathcal{H}$, N is the size of (Γ, φ_f), M is the number of extended T-isomorphism types and D is the number of extended TS-isomorphism types. With arithmetic, M and D are the products of number of normal T-and TS-isomorphism types multiplied by $|\mathcal{K}_T|$ respectively. As l is less than the number of expressions whose upper bounds are obtained in Appendix C, by applying Lemma 90, we obtain upper bounds for M and D for the different types of schema.

By substituting the bounds for M and D, we have the following results. Note that for Γ without artifact relations, the complexity is dominated by the space for pre-computing $\{\mathcal{K}_T\}_{T\in\mathcal{H}}$.

Theorem 91. *Let Γ be a HAS with* **acyclic** *schema and φ_f an HLTL-FO property over Γ, where arithmetic is allowed in Γ and φ_f. $\Gamma \models \varphi_f$ can be verified in $2\text{-}\exp(N^{O(h+r)})$ deterministic space. If Γ does not contain artifact relation, then $\Gamma \models \varphi_f$ can be verified in $\exp(N^{O(h+r)})$ deterministic space.*

Theorem 92. *Let Γ be a HAS with* **linearly-cyclic** *schema and φ_f an HLTL-FO property over Γ, where arithmetic is allowed in Γ and φ_f. $\Gamma \models \varphi_f$ can be verified in $O(2\text{-}\exp(N^{c_1\cdot h^2}))$ deterministic space, where $c_1 = O(r)$. If Γ does not contain artifact relation, then $\Gamma \models \varphi_f$ can be verified in $O(\exp(N^{c_2\cdot h^2}))$ deterministic space, where $c_2 = O(r)$.*

Theorem 93. *Let Γ be a HAS with* **cyclic** *schema and φ_f an HLTL-FO property over Γ, where arithmetic is allowed in Γ and φ_f. $\Gamma \models \varphi_f$ can be verified in $(h+2)\text{-}\exp(O(N))$ deterministic space. If Γ does not contain artifact relation, then $\Gamma \models \varphi_f$ can be verified in $(h+1)\text{-}\exp(O(N))$ deterministic space.*

Table 3. Symbols for HAS and HLTL-FO.

DB	database schema of a HAS
$\bar{x}^T$	artifact variables of task T
$\bar{x}^T_{\text{in}}$ / $\bar{x}^T_{\text{out}}$	input/output variables of task T
$\bar{x}^T_{T_c\uparrow}$ / $\bar{x}^T_{T_c\downarrow}$	variables of task T passed to T_c / returned from T_c
$\bar{x}^T_{id}$ / $\bar{x}^T_{\mathbb{R}}$	ID/numeric variables of task T
$\mathcal{S}^T$	artifact relations of task T
s^T	variables of task T for updating $\mathcal{S}^T$
$\mathcal{H}$	task hierarchy
$child(T)$	set of children of task T
$desc(T)$	set of descendants of task T (excluding T)
$desc^*(T)$	set of descendants of task T (including T)
$\mathcal{S}_{\mathcal{H}}$	artifact relations of all tasks in $\mathcal{H}$
stg	stage mapping from tasks to {`init`, `active`, `closed`}
π	pre-condition of an internal task
ψ	post-condition of an internal task
δ	update of an internal task
σ^o_T	opening service of task T
σ^c_T	closing service of task T
Σ_T	set of internal services of task T
Σ^{oc}_T	$\Sigma_T \cup \{\sigma^o_T, \sigma^c_T\}$
Σ^{obs}_T	$\Sigma^{oc}_T \cup \{\sigma^o_{T_c}, \sigma^c_{T_c} \mid T_c \in child(T)\}$
$\mathcal{A}$	artifact schema
$\mathcal{C}$	the infinite set of arithmetic relation symbols
Γ	HAS
Π	global pre-condition of a HAS Γ
$I = (\nu, S)$	instance of a task
$\rho_T = (\nu_{in}, \nu_{out}, \{(I_i, \sigma_i)\}_{i\geq 0})$	local run of task T
Tree	tree of local runs
$\rho = \{(I_i, \sigma_i)\}_{i\geq 0}$	global run of a HAS
$\mathcal{L}(\textbf{\textit{Tree}})$	set of all global runs obtained by linearizing ***Tree***
$Runs_D(\Gamma)$	set of global runs of Γ on a database D
$Runs(\Gamma)$	set of all global runs of Γ
B_φ	Büchi automaton for the LTL formula φ
ω	ordinal omega
FO	first-order logic
$\exists$FO	existential FO
φ_f	HLTL-FO formula over task T_1
Φ_T	the set of all HLTL-FO subformulas of φ_f over task T
$B(T, \eta)$	the Büchi automaton constructed from Φ_T with adornment η
$\mathcal{B}_\varphi$	the collection of all $B(T, \eta)$

Table 4. Symbols for verification without arithmetic.

$x_R.w$	navigation expression starting from x with a path w
$\mathcal{E}_T$	set of navigation expressions via foreign keys of task T
$\mathcal{E}_T^+$	extended navigation set (i.e. $\mathcal{E}_T \cup \bar{x}^T\{0, \texttt{null}\}$)
$\sim_\tau$	equality type over $\mathcal{E}_T^+$
τ	T-isomorphism type
$I = (\tau, \bar{c})$	symbolic instance with counters $\bar{c}$
$\tilde{\rho}_T = (\tau_{in}, \tau_{out}, \{(I_i, \sigma_i)\}_{0 \le i < \gamma})$	local symbolic run of task T
$\hat{\tau}$	TS-isomorphism type
TS_{ib}	set of input-bound TS-isomorphism type
$\bar{x}^T_{const}$	subsequence of $\bar{x}^T$ that stay unchanged within a transition
Sym	symbolic tree of runs
$\nu^*(e)$	value of expression e in valuation ν of $\bar{x}^T$
$\mathcal{I}^+$ / $\mathcal{I}^-$	sets of inserting/retrieving symbolic instances
$S = \{(I_i, \sigma_i)\}_{a \le i \le b}$	segment
$L = \{(I_i, \sigma_i)\}_{i \in J}$	life-cycle
(i, e)	local expression
$\mathcal{E}_i$ / $\sim_i$	local navigation set / local equality type
$\Lambda = (\mathcal{E}, \sim)$	global-isomorphism type over local expressions (i.e. (i, e)'s)
$[(i, e)]_i$	the equivalence class of (i, e) wrt $\sim_i$
$G(\sim_i)$	navigation graph of the local equality type $\sim_i$
$\texttt{Reach}_i(x, w)$	the unique node of $G(\sim_i)$ reachable from $[(i, x)]_i$ with path w
$S_1 \equiv S_2$	equivalence of two segments S_1 and S_2
$L_1 \equiv L_2$	equivalence of two life cycles L_1 and L_2
$dyn(L)$	the sequence of dynamic segments of life cycle L
$sp(L)$	the timespan of a life cycle L
VASS	vector addition system with states
$\mathcal{R}_T$	reachability relation with input/output of task T
$\mathcal{V}(T, \beta)$	VASS constructed from task T and truth assignment β to Φ_T
$(\tau_i, \sigma_i, q_i, \bar{o}_i, \bar{c}^i_{ib})$	state of $\mathcal{V}(T, \beta)$ consisting of a T-isomorphism type, a service, a Büchi automaton state, stages of child tasks, and counters for input-bound TS-isomorphism types

Table 5. Symbols for verification with arithmetic.

$\mathbb{Z}[\mathcal{E}_{\mathbb{R}}^{T}]$	polynomial ring over the set of numeric expressions of T
P	polynomial
$\mathcal{P}$	set of polynomials
σ	sign condition
κ	cell
$\kappa(\sigma, \mathcal{P})$	cell defined by sign condition σ over the set of polynomials $\mathcal{P}$
$\mathcal{K}(\mathcal{P}, \mathcal{E})$	set of cells over $(\mathcal{E}, \mathcal{P}\|\mathcal{E})$
$\texttt{proj}(\kappa, \bar{x})$	the set of polynomials in the projection of κ onto $\bar{x}$
$\kappa \sqsubseteq \kappa'$	refinement of cells
$\kappa \sqsubseteq_{\bar{x}} \kappa'$	refinement of cells wrt the projection onto variables $\bar{x}$
$(\cdot)^{T_c \to T}$	renaming of expressions / variables / polynomials / cells from a child task to its parent
$\{\mathcal{K}_T\}_{T \in \mathcal{H}}$	Hierarchical Cell Decomposition (HCD)
$\tau = (\mathcal{E}_T, \sim_\tau, \kappa)$	extended isomorphism type