Verifying Recursive Active Documents with Positive Data Tree Rewriting

Blaise Genest[1,2], Anca Muscholl[3], and Zhilin Wu[4]

1 **CNRS, IPAL UMI, joint with I2R-A*STAR-NUS, Singapore**
2 **CNRS, IRISA UMR, joint with Université Rennes I, France**
`bgenest@irisa.fr`
3 **LaBRI, Université Bordeaux/CNRS, France**
`anca@labri.fr`
4 **LaBRI, Université Bordeaux/CNRS, France**
`zlwu@labri.fr`

Abstract

This paper considers a tree-rewriting framework for modeling documents evolving through service calls. We focus on the automatic verification of properties of documents that may contain data from an infinite domain. We establish the boundaries of decidability: while verifying documents with recursive calls is undecidable, we obtain decidability as soon as either documents are in the *positive-bounded* fragment (while calls are unrestricted), or when there is a bound on the number of service calls (bounded model-checking of unrestricted documents). In the latter case, the complexity is NexpTime-complete. Our data tree-rewriting framework resembles Guarded Active XML, a platform handling XML repositories that evolve through web services. The model here captures the basic features of Guarded Active XML and extends it by node renaming and subtree deletion.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.469

1 Introduction

From static in house solutions, databases have become more and more open to the world, offering e.g. half-open access through web services. As usual for open systems, their design requires a careful static analysis process, helping to guarantee that no malicious client may take advantage of the system in a way for which the system was not designed. Static analysis of such systems very recently brought together two areas - databases, with emphasis on semi-structured XML data, and automated verification, with emphasis on model-checking infinite-state systems. Systems modeling dynamical evolution of data are pretty challenging for automated verification, as they involve feedback loops between semi-structured data, possibly with values from unbounded domains, and the workflow of services. If both topics have been studied extensively on its own, very few papers tackle decidability of algorithms when all aspects are present at the same time.

An interesting model emerged recently for handling XML repositories evolving through web services, namely Active XML (AXML) [4]. These are XML documents that evolve dynamically, containing implicit data in form of embedded service calls. Services may be recursive, so the evolution of such documents is both non-deterministic and unbounded in time. A first paper analyzing the evolution of AXML documents considered *monotonous* documents [3]. With this restriction, as soon as a service is enabled in a document, then from this point on the service cannot be disabled and calling it can only extend the document. In particular, information can never be deleted. Recently, a workflow-oriented version of AXML was proposed in [5]: the *Guarded AXML* model (GAXML for short) adds guards to service calls, thus controlling the possible evolution of active documents. Decidability in co-2NexpTime of static analysis for the *recursion-free GAXML* fragment w.r.t. a variant of LTL with data tree patterns as atomic formulas was established in [5]. Static analysis is more complex in [5], due to the presence of unbounded data. The crucial restriction needed for decidability is a uniform bound on the number of possible service calls. Compared to [3], service invocation can terminate, and more importantly, negative guards can be used. But still, deletion of data is not possible. Finally, the GAXML model relies on a rather involved semantics of service calls.

In this work, our aim is twofold. First, we aim at embedding and extending the GAXML model in a simpler framework based on tree rewriting. Our model DTPRS (*data tree pattern rewriting systems*) uses the same basic ingredients as GAXML, which are tree patterns for guards and queries. However, our formalism allows to describe several possible effects of a service call: materialization of implicit data like in GAXML, but also deletion and modification of existing document parts. This model is a simplified version of the TPRS model proposed in [15], but in this setting it can additionally handle unbounded data.

Our second, and main objective is to get decidability of static analysis of DTPRS without relying on a bound on the number of service calls. For doing that, we use a technique that emerged in the verification of particular infinite-state systems such as Petri nets and lossy channel systems. The main concept is known in verification as *well-structured transition systems* (WSTS for short) [1, 13]. WSTSs are one example for infinite-state systems where (potentially) infinite sets of states can be represented (and effectively manipulated) symbolically in a finite way.

Our basic objects are data trees, i.e., trees with labels from an infinite domain. We view data trees as graphs, and define in a natural way a quasi-order on such graphs. Then we show that a uniform bound on the length of simple paths in such graphs, together with positive guards, makes DTPRS well-structured systems [1, 13]. As a technical tool we use here tree decompositions of graphs. In a nutshell we trade here recursion against positiveness, since considering both leads to undecidable static analysis. We show that for *positive-bounded* DTPRS, termination and tree pattern reachability are both decidable. On the negative side, we show that the verification of very simple Tree-LTL properties is undecidable even for *positive-bounded* DTPRS. On the positive side, the decidability result for pattern reachability can be extended to the verification of existential positive Tree-LTL properties. We then consider the type-checking problem, another static analysis problem, and show its Co-NexpTime-completeness for arbitrary DTPRS. Finally, we show that *bounded* model-checking of arbitrary DTPRS is NexpTime-complete.

Related work: Verification of web services often ignores unbounded data (c.f. e.g. [17, 14]). On the other hand, several data-driven workflow process models have been proposed. Document-driven workflow was proposed in [20]. Artifact-based workflow was outlined in [16], in which artifacts are used to represent key business entities, including both their data and life cycles. An early line of results involving data establishes decidability boundaries for the verification of temporal (first-order based) properties of a data-driven workflow processes, based on a relational data model [11, 10, 12]. This approach has been recently extended to the artifact-based model [9].

On the verification side, there is a rich literature on the verification of well-structured infinite transition systems [1, 13], ranging from faulty communication systems [7] to programs manipulating dynamic data [2] (citing only a few recent contributions). The latter work is one of the few examples where well-quasi-order on graphs is used.

Organization of the paper: In the next section, we fix some definitions and notations, then define the DTPRS model. In Section 3 we show that analysis of DTPRS with recursive DTD or negated tree patterns is undecidable. In Section 4 we define positive-bounded DTPRS and prove our decidability results. Then in Section 5, we show the undecidability of the verification of simple Tree-LTL properties and the decidability of existential positive Tree-LTL properties for positive-bounded DTPRS. In Section 6, we consider the type-checking problem for DTPRS and show its Co-NexpTime completeness. Finally in Section 7, we consider bounded model-checking problem for DTPRS and show its NexpTime-completeness.

2 Definitions and notations

In this paper, documents are labeled, unranked, unordered trees. We fix a finite alphabet Σ (with symbols $a, b, c, \dots$, called tags) and an infinite data domain $\mathcal{D}$ (with symbols $d, \dots$). A *data tree* (see Figure 1) is a (rooted) tree T with nodes labeled by $\Sigma \cup \mathcal{D}$. A data tree T can be represented as a tuple $T = \langle V, E, \mathrm{root}, \ell \rangle$, with labeling function $\ell : V \to \Sigma \cup \mathcal{D}$. Internal nodes are Σ-labeled, whereas leaves are $(\Sigma \cup \mathcal{D})$-labeled. We also fix a (finite) set of variables $\mathcal{X}$ (with symbols $X, Y, Z, \dots$) that will take values in $\mathcal{D}$, and use $*$ as special symbol standing for any tag. Let $\mathcal{T}$ denote the set $\Sigma \cup \mathcal{X} \cup \{*\}$.

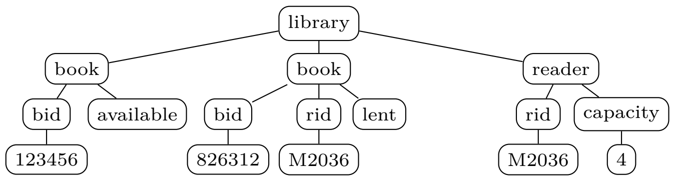

Figure 1 A document for a library system: The reader $M2036$ borrows a book with identifier 826312 from the library, and has capacity 4, namely he or she is able to borrow at most 4 other books.

We now introduce the different components used in our rewriting rules: *data tree patterns* to locate and specify a pattern of the document, *data constraints* to express data equalities and inequalities, *data tree pattern queries* to extract information from a document.

A *data constraint* is a Boolean combination of relations $X = Y$, with[1] $X, Y \in \mathcal{X}$. A *data tree pattern* (DTP) $P = \langle V, E, \mathrm{root}, \ell, \tau, \mathit{cond} \rangle$ is a (rooted) $\mathcal{T}$-labeled tree $\langle V, E, \mathrm{root}, \ell \rangle$, together with an edge-labeling function $\tau : E \to \{|, ||\}$ and a data constraint *cond*. Edges that are $|$-labeled denote child edges, and $||$-labeled edges denote descendant ones. Internal nodes are labeled by $\Sigma \cup \{*\}$, and leaves by $\mathcal{T}$. A *matching* of a DTP P into a data tree T is defined as a mapping preserving the root, the Σ- and $\mathcal{D}$-labels (with $*$ as wildcard), the child and the descendant relations, satisfying *cond* and mapping $\mathcal{X}$-labeled nodes to $\mathcal{D}$-labeled ones. In particular, a relation $X = Y$ $(X, Y \in \mathcal{X})$ in a DTP P means that the corresponding leaves of P must be mapped to leaves of T carrying the same data value. An *injective* matching of P into T means that the mapping above is injective. A *relative* DTP is a DTP with one designated node *self*. A relative DTP (P, self) is matched to a pair (T, v), where T is a tree and v is a node of T.

We use *Boolean combinations* of (relative) DTPs as rule guards. DTPs in a Boolean combination are matched *independently* of each other, except that nodes designated by *self*

[1] For simplicity we disallow here explicit data constants $X = d$ $(d \in \mathcal{D})$: they can be simulated by tags from Σ.

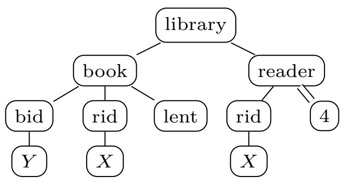

Figure 2 A data tree pattern (DTP).

must be matched to the same node of T. Boolean operators are interpreted by the standard meaning.

A *data tree pattern query* (DTPQ) is of the form *body* $\rightsquigarrow$ *head*, with *body* a DTP and *head* a tree such that

- the internal nodes of *head* are labeled by Σ and its leaves are labeled by $(\Sigma \cup \mathcal{D} \cup \mathcal{X})$,
- every variable occurring in *head* also occurs in *body*,
- there is at least one variable occurring in *head*, i.e., at least one leaf of *head* is labeled by $\mathcal{X}$ (i.e., *head* is not a constant tree).

Let T be a data tree and $Q = \mathit{body} \rightsquigarrow \mathit{head}$ be a DTPQ. The evaluation result of Q over T is the forest $Q(T)$ of all instantiations of *head* by matchings from *body* to T. For example, the DTPQ $P \rightsquigarrow \mathit{head}$ with P given by Figure 2 and *head* consisting of a unique node labeled by Y, returns a forest consisting of one-node trees labeled by identifiers of books which are borrowed by readers with capacity 4. A *relative* DTPQ is defined like a DTPQ, except that its *body* is a relative DTP. A relative DTPQ Q is evaluated on a pair (T, v). The result of $Q(T, v)$ is defined as above, except that matchings of *body* must map node *self* to v. For instance, the relative DTPQ $P \rightsquigarrow \mathit{head}$ where the *reader* node is labeled *self* will return a forest of one-node trees labeled by identifiers of books which are borrowed by a particular reader with capacity 4 designated by *self*.

Similar to GAXML, data tree rewriting rules are guarded by (Boolean combinations of) DTPs and they can add information to a tree via queries. In addition, our rules can rename tags and delete nodes, together with their subtrees. Each rule comes along with a context called *locator*, that also describes all the operations related to a rewriting step. A locator L is a relative DTP with additional labels *append*, *del*, and ren_a $(a \in \Sigma)$. The meaning of these labels is to add information (*append*), delete a node and its subtree (*del*) and rename a tag into a (ren_a). Labels *append* and ren_a are not exclusive. They are restricted to be attached to nodes of L that are labeled by $\Sigma \cup \{*\}$. Label *del* can be attached to any node. We assume that all descendants in L of a node with *del* are also labeled by *del*, and that nodes labeled by *del* cannot be labeled by *append* or ren_a.

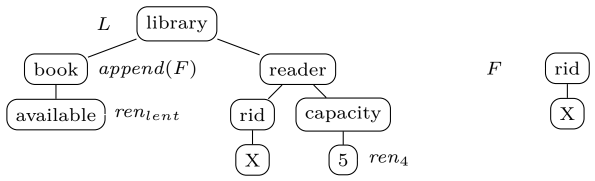

Figure 3 DTP rule "borrow": The reader identifier is added as a subtree to the node "book" whose state is renamed as "lent", and the capacity of the reader is decreased by renaming.

A *data tree pattern rewriting rule* (DTP rule for short) R is a tuple $\langle L, G, \mathcal{Q}, \mathcal{F}, \chi \rangle$ with:

- L is a *locator*,

- G is a *guard*: a Boolean combination of (relative) DTPs,
- $\mathcal{Q}$ is a finite set of relative DTPQs,
- $\mathcal{F}$ is a finite set of forests with internal nodes labeled by Σ and leaves labeled by $\Sigma \cup \mathcal{D} \cup \mathcal{X} \cup \mathcal{Q}$,
- χ is a mapping from the set of nodes of L labeled by *append* to $\mathcal{F}$.

A *DTP rewriting system (DTPRS)* is a pair $(\mathcal{R}, \Delta)$ consisting of a finite set $\mathcal{R}$ of DTP rules and a *static invariant* Δ. The latter is a *DTD* τ together with a *data invariant inv*, i.e. a Boolean combination of DTPs. As usual for unordered trees, the DTD τ describes the syntax of trees. It is defined as a tuple $(\Sigma_r, \mathcal{P})$ such that Σ_r is the set of allowed root labels, and $\mathcal{P}$ is a finite set of rules $a \to \psi$ such that $a \in \Sigma$ and ψ is a Boolean combination of inequalities of the form $|b| \geq k$, where $b \in \Sigma \cup \{dom\}$ (*dom* is a symbol standing for any data value), and k is a non-negative integer. A *positive* DTD is one where the Boolean combinations above are positive. A *non-recursive* DTD is one where the rule graph is acyclic (the rule graph has Σ as vertex set and edges (a, b) for every a, b such that b occurs in a $a \to \psi$). For $\Delta = (\tau, inv)$ we write $T \models \Delta$ for a data tree T satisfying both τ and *inv*.

An example of a DTP rule for a reader of capacity 5 to borrow a book from a library is illustrated in Figure 3, including the locator L and $\mathcal{F} = \{F\}$.

We first describe the semantics of DTP rules informally. First, the locator is mapped against the data tree in a non-deterministic way. Then, queries are evaluated, thus determining the information that will possibly enhance the tree using the *append*-labels in the locator. Deletion and renaming are performed as expected. The resulting data tree must satisfy the static invariant Δ.

We now define the semantics formally. Let $T = \langle V, E, \mathrm{root}, \ell \rangle$ be a data tree with $T \models \Delta$, and let $R = \langle L, G, \mathcal{Q}, \mathcal{F}, \chi \rangle$ be a DTP rule.

- Let μ be an *injective* matching from L to T. Let ν be the assignment of data values to variables in L such that $\nu(X) = \ell(\mu(v))$ for every v labeled by $X \in \mathcal{X}$ in L.
- For each variable $X \in \mathcal{X}$ we denote its evaluation as $X(T)$, with $X(T) = \nu(X)$ if defined, and $X(T)$ **a fresh data value otherwise**. Here a fresh data value is a data value which does not appear in T. Furthermore, it is required that all the *new* variables of R, i.e. variables occurring in $\mathcal{F}$, but not in L, should take **mutually distinct** fresh values. For each forest $F \in \mathcal{F}$, we denote its evaluation by $F(T)$, by replacing labels $Q \in \mathcal{Q}$ by $Q(T)$ and labels $X \in \mathcal{X}$ by $X(T)$. Recall that all queries $Q \in \mathcal{Q}$ are evaluated relatively to $\mu(self)$.
- A data tree T' is obtained from T by
  - deleting subtrees rooted at nodes $\mu(v)$ whenever v is labeled by *del* in L,
  - changing the tag of a node $\mu(v)$ to a whenever v is labeled by ren_a in L,
  - appending $F(T)$ as a subforest of nodes $\mu(v)$ whenever v is labeled by *append* in L and $\chi(v) = F$,
  - every other node of T keeps its tag or data.
- The rule R is enabled on data tree T if there exists an *injective* matching μ of L into T such that (1) the guard G is true on $(T, \mu(v))$ with v labeled by *self* in L, and (2) there is a data tree T', obtained from T and μ by the operations specified above, satisfying $T' \models \Delta$.

Let $T \xrightarrow{R} T'$ denote the transition from T to T' using DTP rule $R \in \mathcal{R}$.

▶ Remark. **1.** The injectivity of the matching μ ensures that the outcome of a rewriting step is well-defined. In particular, no two nodes with label *del* and *append* (or ren_a), resp., can be mapped to the same node in the data tree. Notice that mappings used for guards or queries may be non-injective.

2. For the new variables occurring in $\mathcal{F}$, but not in L, we choose mutually distinct fresh values. We could have chosen arbitrary values instead, and enforce that they are fresh and mutually distinct *a posteriori* using the data invariant *inv*. In this case, *inv* needs negation. The *inv* (or the locator) can be also used to enforce that the (arbitrarily) chosen values already occur in T. This kind of invariant would be positive.
3. In our definition of DTP rules, it might appear that guards are redundant w.r.t. the locator. However, this is not the case in general, e.g. in the situation that guards include disjunctions or negations of DTPs.

Given a DTPRS $(\mathcal{R}, \Delta)$, let $T \longrightarrow T'$ denote the union of $T \xrightarrow{R} T'$ for some $R \in \mathcal{R}$, and $T \xrightarrow{+} T'$ (or $T \xrightarrow{*} T'$) denote the transitive (or reflexive and transitive) closure of $T \longrightarrow T'$. Moreover, let $\mathcal{T}^*_{\mathcal{R}}(T)$ denote the set of trees that can be reached from a data tree T by rewriting with DTP rules from $\mathcal{R}$, i.e. $\mathcal{T}^*_{\mathcal{R}}(T) = \{T' \mid T \xrightarrow{*} T'\}$. For a set of data trees $\mathcal{I}$, let $\mathcal{T}^*_{\mathcal{R}}(\mathcal{I})$ be the union of $\mathcal{T}^*_{\mathcal{R}}(T)$, for $T \in \mathcal{I}$.

We are interested in the following questions, given a DTPRS $(\mathcal{R}, \Delta)$:

- *Pattern reachability*: Given a DTP P and a set of initial trees[2] *Init*, given as the conjunction of a DTD and a Boolean combination of DTPs, is there some $T \in \mathcal{T}^*_{\mathcal{R}}(\mathit{Init})$ such that P matches T?
- *Termination:* Given an initial data tree T_0, are all runs (rewriting paths) $T_0 \to T_1 \to \cdots$ starting from T_0 finite?

The reason for the fact that termination of DTPRS is defined above w.r.t. a single initial data tree is that termination from a set of initial trees is already undecidable without data (see Proposition 3).

3 Undecidability

As one might expect, the analysis of DTPRS is quickly undecidable – and sometimes already without using any unbounded data. The proof of the proposition below is obtained by a straightforward simulation of 2-counter machines.

▶ Proposition 1. Both pattern reachability and termination for DTPRS $(\mathcal{R}, \Delta)$ are undecidable whenever one of the following holds:

1. the DTD in Δ is recursive,
2. either guards in $\mathcal{R}$ or the invariant Δ contain negated DTPs.

The above result holds already without data.

The next result shows that with data, we can relax both conditions above and still get undecidability of DTPRS. The main idea is to use data for creating long horizontal paths (although trees are supposed to be unordered). Such horizontal paths can be obtained e.g. with a tree of depth 2, with each subtree (of the root) containing three nodes, a node plus its two children labeled respectively by data values d_i, d_{i+1}. Assuming all data values d_i are distinct (and distinguishing d_1), then a linear order on these subtrees is obtained.

▶ **Theorem 2.** *Both pattern reachability and termination are undecidable for DTPRS* $(\mathcal{R}, \Delta)$ *such that (1) the DTD in* Δ *is non-recursive and (2) all DTPs from guards in* $\mathcal{R}$ *and the invariant* Δ *are positive.*

[2] We require that every tree in *Init* satisfies Δ.

We end this section with a remark on the undecidability of termination from an *initial set of trees*. First we notice that – already without data – DTPRS can simulate so-called *reset Petri nets* [15]. These are Petri nets (or equivalently, multi-counter automata without zero test) with additional transitions that can reset places (equivalently, counters) to zero. They can be represented by trees of depth 2, where nodes at depth one represent places, and their respective number of children (leaves) is the number of tokens on that place. A DTPRS (without data) can easily simulate increments, decrements and resets (using deletion in DTPRS). It is known that so-called *structural termination* for reset Petri nets is undecidable [18], i.e., the question whether there are infinite computations from *any* initial configuration, is undecidable. This implies:

▶ Proposition 3. The following question is undecidable: Given a DTPRS $(\mathcal{R}, \Delta)$, is there some tree T_0 satisfying Δ and an infinite computation $T_0 \longrightarrow T_1 \longrightarrow \cdots$ in $(\mathcal{R}, \Delta)$? This holds already for non-recursive DTD in Δ and without data constraints in DTPs.

It follows from Proposition 3 that termination from an initial set of trees, namely to decide whether for every $T_0 \in \mathit{Init}$, all the runs starting from T_0 terminate, is undecidable.

4 Positive-bounded DTPRS

In this section we consider *positive-bounded DTPRS*, a fragment of DTPRS for which we show that pattern reachability and termination are decidable.

From Proposition 1, we know that in order to get decidability, the DTD in the static invariant Δ must be non-recursive. For a non-recursive DTD, there is some B such that every tree satisfying the DTD has depth bounded by B. In the following, we assume the existence of such a bound B. Also from Proposition 1, we know that for obtaining decidability we need to restrict ourselves to positive guards and positive data invariants.

However, from Theorem 2, we know that these restrictions alone do not suffice to achieve decidability. We also need to disallow long linear orders created by data. For this, we introduce a last restriction, called *simple-path bounded*, which is defined in the following.

Let $T = \langle V, E, \mathit{root}, \ell \rangle$ be a data tree. *The graph* $G(T)$ *associated with* T is the undirected graph obtained from T by merging all the nodes labeled by the same $d \in \mathcal{D}$ into a single node d. Formally, $G(T) = (V', E')$, where $V' = \{v \in V \mid \ell(v) \in \Sigma\} \cup \{\ell(v) \mid v \in V, \ell(v) \in \mathcal{D}\}$ and $E' = \{\{v, w\} \mid \ell(v), \ell(w) \in \Sigma, (v, w) \in E\} \cup \{\{v, d\} \mid \ell(v) \in \Sigma, \exists w \text{ s.t. } (v, w) \in E, \ell(w) = d\}$. A *simple path* of T is a simple path in $G(T)$, i.e. a sequence of vertices $v_1, \ldots, v_n$ in $G(T)$ such that for all $i \neq j$, $\{v_i, v_{i+1}\} \in E'$ and $v_i \neq v_j$. The length of a path $v_1, \ldots, v_n$ is $n - 1$.

Formally, a DTPRS $(\mathcal{R}, \Delta)$ is *positive-bounded* with set of initial trees *Init*, if:

- **non-recursive-DTD:** the DTD in the static invariant Δ is non-recursive. In particular, trees satisfying the DTD have depth bounded by some $B > 0$.
- **positive:** all guards in $\mathcal{R}$ and the data invariant in Δ are positive Boolean combinations of DTPs. The DTD in Δ is positive as well.
- **simple-path bounded:** there exists $K > 0$ such that for any $T_0 \in \mathit{Init}$, the length of any simple path in any $T \in \mathcal{T}^*_{\mathcal{R}}(T_0)$, is bounded by K.

Notice that the third condition above implies that all data trees have depth bounded by K. So we always assume that $B \leq K$. Notice also that in positive-bounded DTPRS, the data value inequality is *allowed* in DTPs, that is, we can state that two data values are different. Moreover, fresh data values (for the new variables in DTP rules) can be used to model some conceptually negative data constraints, like for instance key properties. Notice also that there is no restriction on the DTD of the initial set *Init*. However, since the DTD in the invariant Δ is positive, “at most”-constraints must be ensured via the rewriting rules.

The library example illustrated in Figure 1 includes DTP rules for book borrowing (Figure 3) and returning, the registration of new books and new readers (where fresh data values can be used to guarantee that the *rid* and *bid* are "keys"), and the deletion of reader accounts. It is easy to notice that the library example satisfies the first 2 conditions above. It is also the case for the third condition. Indeed, all simple paths are bounded by 7: A longest path is for instance: library - book - rid - $M2036$ - rid - reader - capacity - 4. Notice that the bound still holds even if the capacity of a reader is unbounded.

The rest of the section is devoted to the proof of the following result:

▶ **Theorem 4.** *Given a positive-bounded DTPRS* $(\mathcal{R}, \Delta)$*, pattern reachability and termination are both decidable.*

We prove Theorem 4 by using the framework of *well-structured transition systems (WSTS)* [1, 13], which has been applied to DTPRS *without* data in [15]. We recall briefly some definitions. A WSTS is a triple $(S, \longrightarrow, \preceq)$ such that S is an (infinite) state space, $\preceq$ is a *well-quasi-ordering*[3] *(wqo for short)* on S, and $\longrightarrow$ is the transition relation on S. It is required that $\longrightarrow$ is *compatible w.r.t.* $\preceq$: for any $s, t, s' \in S$ with $s \longrightarrow t$ and $s \preceq s'$, there exists $t' \in S$ such that $s' \longrightarrow t'$ and $t \preceq t'$.

Let $\mathcal{T}_{B,K}$ denote the set of data trees whose depths are bounded by B and lengths of simple paths are bounded by K. From the definition of positive-bounded DTPRS, we know that $\mathcal{T}^*_{\mathcal{R}}(\mathit{Init}) \subseteq \mathcal{T}_{B,K}$. In the following, we prove Theorem 4 by defining a binary relation $\preceq$ on $\mathcal{T}_{B,K}$ and showing that $(\mathcal{T}_{B,K}, \longrightarrow, \preceq)$ is a WSTS.

4.1 Well-structure of positive-bounded DTPRS

We define a binary relation $\preceq$ on $\mathcal{T}_{B,K}$ as follows. Let $T_1 = \langle V_1, E_1, \mathrm{root}_1, \ell_1 \rangle, T_2 = \langle V_2, E_2, \mathrm{root}_2, \ell_2 \rangle \in \mathcal{T}_{B,K}$, then $T_1 \preceq T_2$ if there is an injective mapping ϕ from V_1 to V_2 such that

- root preservation: $\phi(\mathrm{root}_1) = \mathrm{root}_2$,
- parent-child relation preservation: $(v_1, v_2) \in E_1$ iff $(\phi(v_1), \phi(v_2)) \in E_2$,
- tag preservation: If $\ell_1(v) \in \Sigma$, then $\ell_1(v) = \ell_2(\phi(v))$,
- data value (in)equality preservation: If $v_1, v_2 \in V_1$ and $\ell_1(v_1), \ell_1(v_2) \in \mathcal{D}$, then $\ell_2(\phi(v_1)), \ell_2(\phi(v_2)) \in \mathcal{D}$, and $\ell_1(v_1) = \ell_1(v_2)$ iff $\ell_2(\phi(v_1)) = \ell_2(\phi(v_2))$.

It is easy to see that $\preceq$ is reflexive and transitive, so it is a quasi-order. In the following, we first assume that $\preceq$ is a wqo on $\mathcal{T}_{B,K}$ and show that $\longrightarrow$ is compatible with $\preceq$, in order to prove Theorem 4. We show in Section 4.2 that $\preceq$ is indeed a wqo: for any infinite sequence of data trees $T_0, T_1, \ldots \in \mathcal{T}_{B,K}$, there are $i < j$ such that $T_i \preceq T_j$.

▶ Proposition 5. Let $(\mathcal{R}, \Delta)$ be a positive-bounded DTPRS. Let $T_1, T'_1, T_2 \in \mathcal{T}_{B,K}$, $T_1 \xrightarrow{R} T_2$ for some $R \in \mathcal{R}$, and $T_1 \preceq T'_1$. Then there exists $T'_2 \in \mathcal{T}_{B,K}$ such that $T'_1 \xrightarrow{R} T'_2$ and $T_2 \preceq T'_2$.

Consequently, in the positive-bounded fragment $\longrightarrow$ is compatible w.r.t. $\preceq$ in $\mathcal{T}_{B,K}$, thus $(\mathcal{T}_{B,K}, \longrightarrow, \preceq)$ is a WSTS.

In addition, it can be shown that $(\mathcal{T}_{B,K}, \longrightarrow, \preceq)$ satisfies some additional computability conditions which are needed to show the decidability of pattern reachability and termination, namely, *effectiveness of pred-basis* for pattern reachability and *effectiveness of successor relation* for termination. With these computability conditions, Theorem 4 then follows from the properties of WSTS (c.f. Theorem 3.6 and Theorem 4.6 in [13]).

[3] A wqo $\preceq$ is a reflexive, transitive and well-founded relation with no infinite antichain.

4.2 Well-quasi-ordering for data trees

In order to prove that $\preceq$ is a wqo over $\mathcal{T}_{B,K}$, we first represent a data tree T as a (labeled) undirected graph $G_\ell(T)$, then we encode $G_\ell(T)$ into a tree (without data) of *bounded depth* using the concept of tree decompositions. Define a binary relation $\leq$ on labeled trees (without data) of bounded depth as follows: $T_1 \leq T_2$ if there is an injective mapping from T_1 to T_2 preserving the root, the tags, and the parent-child relation. It is known that $\leq$ is a wqo on labeled trees of bounded depth *without data* [15].

Let $\mathcal{G}_K$ be the set of labeled graphs with the lengths of all simple paths bounded by K. In the following, we show that $\preceq$ on $\mathcal{T}_{B,K}$ corresponds to the induced subgraph relation (formally defined later) on $\mathcal{G}_K$, and the fact that $\leq$ is a wqo for labeled trees of bounded depth implies that the induced subgraph relation is a wqo on $\mathcal{G}_K$.

Given a data tree $T = \langle V, E, root, \ell\rangle \in \mathcal{T}_{B,K}$, *the labeled undirected graph representation* $G_\ell(T)$ *of* T is obtained from $G(T)$, the graph associated to T, by adding labels encoding information of data tree nodes (tag, depth ...). Formally, $G_\ell(T)$, is a $((\Sigma \times [B+1]) \cup \{\$\})$-labeled (where $[B+1] = \{0, 1, \cdots, B\}$) undirected graph (V', E', ℓ') defined as follows,

- $V' = \{v \in V \mid \ell(v) \in \Sigma\} \cup \{\ell(v) \mid v \in V, \ell(v) \in \mathcal{D}\}$,
- $E' = \{\{v, w\} \mid \ell(v), \ell(w) \in \Sigma, (v, w) \in E\} \cup \{\{v, d\} \mid \ell(v) \in \Sigma, \exists w, (v, w) \in E, \ell(w) = d\}$,
- Let $v \in V$ such that $\ell(v) \in \Sigma$, then $\ell'(v) = (\ell(v), i)$. In addition, $\ell'(d) = \$$ for each $d \in V' \cap \mathcal{D}$.

Let Σ_G denote $(\Sigma \times [B+1]) \cup \{\$\}$. For Σ_G-labeled graphs, we define the induced subgraph relation as follows. Let $G_1 = (V_1, E_1, \ell_1), G_2 = (V_2, E_2, \ell_2)$ be two Σ_G-labeled graphs, then G_1 is an *induced subgraph* of G_2 (denoted $G_1 \sqsubseteq G_2$) iff there is an injective mapping ϕ from V_1 to V_2 such that

- label preservation: $\ell_1(v_1) = \ell_2(\phi(v_1))$ for any $v_1 \in V_1$,
- edge preservation: let $v_1, v_1' \in V_1$, then $\{v_1, v_1'\} \in E_1$ iff $\{\phi(v_1), \phi(v_1')\} \in E_2$.

From the definition of the labeled graph representation of data trees, it is not hard to show that the induced subgraph relation $\sqsubseteq$ corresponds to the relation $\preceq$ on data trees.

▶ Proposition 6. Let $T_1, T_2 \in \mathcal{T}_{B,K}$, then $T_1 \preceq T_2$ iff $G_\ell(T_1) \sqsubseteq G_\ell(T_2)$.

Now we show how to encode any Σ_G-labeled graph belonging to $\mathcal{G}_K$ into a labeled tree of bounded depth by using tree decompositions.

Let $G = (V, E, \ell)$ be a connected Σ_G-labeled graph, then a *tree decomposition* of G is a quadruple $T = \langle U, F, r, \theta\rangle$ such that:

- (U, F, r) is a tree with the domain U, the parent-child relation F, and the root $r \in U$,
- $\theta : U \to 2^V$ is a labeling function attaching each node $u \in U$ a set of vertices of G,
- For each edge $\{v, w\} \in E$, there is a node $u \in U$ such that $\{v, w\} \subseteq \theta(u)$,
- For each vertex $v \in V$, the set of nodes $u \in U$ such that $v \in \theta(u)$ constitutes a connected subgraph of T.

The sets $\theta(v)$ are called the *bags* of the tree decomposition. The *depth* of a tree decomposition $T = \langle U, F, r, \theta\rangle$ is the depth of the tree (U, F, r) and the *width* of T is defined as $\max\{|\theta(u)| - 1 \mid u \in U\}$. The *tree-width* of a graph $G = (V, E)$ is the minimum width of tree decompositions of G. For a tree decomposition of width K of a graph G, without loss of generality, we assume that each bag is given by a sequence of vertices of length $K+1$, $v_0 \dots v_K$, with possible repetitions, i.e. possibly $v_i = v_j$ for some $i \neq j$ (tree decompositions in this form are sometimes called ordered tree decompositions).

▶ **Theorem 7.** *([19, 6]) If $G \in \mathcal{G}_K$, then G has a tree decomposition with both depth and width bounded by K.*

Now we describe how to encode labeled graphs by trees using tree decompositions.

Let $G = (V, E, \ell) \in \mathcal{G}_K$ be a Σ_G-labeled graph, and $T = \langle U, F, r, \theta \rangle$ be a tree decomposition of G with width K and depth at most K. Remember that each $\theta(u)$ is represented as a sequence of exactly $K + 1$ vertices, and $[K + 1] = \{0, \ldots, K\}$. Define

$$\Sigma_{G,K} := (\Sigma_G)^{K+1} \times 2^{[K+1]^2} \times 2^{[K+1]^2} \times 2^{[K+1]^2}.$$

We transform $T = \langle U, F, r, \theta \rangle$ into a $\Sigma_{G,K}$-labeled tree $T' = (U, F, r, \eta)$, which encodes in a uniform way the information about G (including edge relations and vertex labels). $\eta : U \to \Sigma_{G,K}$ is defined as follows. Let $\theta(u) = v_0 \ldots v_K$, then $\eta(u) = (\ell(v_0) \ldots \ell(v_K), \overline{\lambda})$, where $\overline{\lambda} = (\lambda_1, \lambda_2, \lambda_3)$,

- $\lambda_1 = \{(i, j) \mid 0 \leq i, j \leq K, v_i = v_j\}$,
- $\lambda_2 = \{(i, j) \mid 0 \leq i, j \leq K, \{v_i, v_j\} \in E\}$,
- If $u = r$, then $\lambda_3 = \emptyset$, otherwise let u' be the parent of u in T and $\theta(u') = v'_0 \cdots v'_K$, then $\lambda_3 = \{(i, j) \mid 0 \leq i, j \leq K, v'_i = v_j\}$.

The encoding of labeled graphs into labeled trees establishes a connection between the wqo $\leq$ of labeled trees and the induced subgraph relation ($\sqsubseteq$) of labeled graphs.

▶ Proposition 8. Let G_1, G_2 be two Σ_G-labeled graphs with tree-width bounded by K, and T_1, T_2 be two tree decompositions of width K of resp. G_1, G_2, then the two $\Sigma_{G,K}$-labeled trees T'_1, T'_2 obtained from T_1, T_2 satisfy that: If $T'_1 \leq T'_2$, then $G_1 \sqsubseteq G_2$.

Now we are ready to show that $\preceq$ is a wqo for $\mathcal{T}_{B,K}$. Let $T_0, T_1, \ldots$ be an infinite sequence of data trees from $\mathcal{T}_{B,K}$. Consider the infinite sequence of $\Sigma_{G,K}$-labeled trees $T'_0, T'_1, \ldots$ obtained from the tree decompositions (with width K and depth at most K) of graphs $G_\ell(T_0), G_\ell(T_1), \ldots$. Then there are $i, j : i < j$ such that $T'_i \leq T'_j$, because $\leq$ is a wqo for labeled trees of depth at most K. So $G_\ell(T_i) \sqsubseteq G_\ell(T_j)$ from Proposition 8, and $T_i \preceq T_j$ from Proposition 6. We thus prove following theorem.

▶ **Theorem 9.** *$\preceq$ is a well-quasi-ordering over $\mathcal{T}_{B,K}$.*

5 Verification of temporal properties

Until now we considered only two properties for static analysis: pattern reachability and termination. (Non-)reachability of a DTP can be expressed easily in Tree-LTL [5], which corresponds roughly to linear time temporal logics where atomic propositions are DTPs[4]. We show in this section that allowing for runs of unbounded length makes the validation of (even very simple) Tree-LTL properties undecidable, even without data:

▶ **Theorem 10.** *It is undecidable whether a positive-bounded DTPRS satisfies a Tree-LTL formula $F\varphi$, where φ is a positive Boolean combination of DTPs. This holds already without data.*

The proof of Theorem 10 is by a reduction from the halting problem of two-counter machines. The idea is to simulate a two-counter machine by a positive bounded DTPRS ignoring the zero-tests, and describe them by a Tree-LTL formula $F\varphi$. The proof relies on the universal semantics of Tree-LTL, requiring that every run of the DTPRS satisfies the formula.

[4] Such formulas use actually free variables in patterns, which are then quantified universally. This is consistent with the approach of testing whether a model satisfies the negation of a formula.

If the *existential* semantics of Tree-LTL formulas is used instead, i.e. requiring that there is a run of the DTPRS satisfying a given Tree-LTL formula, then the problem is still undecidable if *negations* are available, since the negation of $F\varphi$ in the universal semantics is $G\neg\varphi$ in the existential semantics. If negations are also forbidden, then we get decidability:

▶ Proposition 11. It is decidable whether a positive-bounded DTPRS satisfies a given positive existential Tree-LTL formula defined by the following rules,

$$\varphi ::= \mathit{true} \mid \mathit{false} \mid P \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid X\varphi_1 \mid \varphi_1 U \varphi_2,$$

where P is a DTP.

6 Type-checking DTPRS

This section shows that it can be checked statically whether DTP rules preserve the static invariant $\Delta = (\tau, \mathit{inv})$ consisting of a DTD τ and a data invariant inv.

Recall that in the definition of DTPRS, if $T \xrightarrow{R} T'$, then it is required that $T' \models \Delta$. Here we drop this requirement and consider the following type-checking problem.

DTPRS Type-checking: Given a DTPRS $(\mathcal{R}, \Delta)$ with a non-recursive DTD[5], decide whether for each T, T' and each DTP rule R such that $T \models \Delta$ and $T \xrightarrow{R} T'$, it holds that $T' \models \Delta$.

▶ **Theorem 12.** *DTPRS type-checking is* Co-NexpTime*-complete.*

The upper bound of Theorem 12 is shown by a small model argument. The lower bound follows from [8], that shows that satisfiability of DTPs on depth-bounded data trees relative to a DTD is NexpTime-hard.

7 Bounded model-checking DTPRS

In this section we consider *bounded model-checking* for DTPRS: Given a DTPRS $(\mathcal{R}, \Delta)$ with a non-recursive DTD [6], a set of initial trees *Init*, a DTP P and a bound N (encoded in unary) we ask whether there is some T_0 satisfying *Init* and some T s.t. P matches T and $T_0 \xrightarrow{\leq N} T$. We have the following result:

▶ **Theorem 13.** *Bounded model-checking for DTPRS is* NexpTime*-complete.*

Theorem 13 can be extended to bounded model-checking Tree-LTL properties. Bounded model-checking of a Tree-LTL formula φ with a bound N is the problem checking whether a counter-example for φ can be obtained in at most N rewriting steps. For instance, bounded model-checking for $G\neg P$ with a bound N is to check whether the DTP P can be reached in $\leq N$ steps.

The proof of Theorem 13 follows the similar line as the proof of Co-2NexpTime completeness of model-checking Tree-LTL properties over recursion-free GAXML ([5]): The upper-bound is shown by a (exponential) small-model property, and the lower-bound is shown by a simulation of the computations of NexpTime-Turing machines. The gap between

[5] If the DTD is recursive, then the problem is undecidable, since the satisfiability of Boolean combinations of DTPs over a recursive DTD is undecidable [8].
[6] The recursive DTD will quickly lead to undecidability, as argued for the type-checking problem.

the NexpTime complexity above and the Co-2NexpTime complexity in [5] is essentially due to the unary encoding of the bound N for bounded model checking.

Acknowledgements We would like to thank the participants of the ANR Docflow and the CREATE Activedoc projects for the discussion on the verification of AXML systems, and in particular Serge Abiteboul and Albert Benveniste.

References

1. P. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *LICS'96*, pages 313–321. IEEE, 1996.
2. P. A. Abdulla, A. Bouajjani, J. Cederberg, F. Haziza, and A. Rezine. Monotonic abstraction for programs with dynamic memory heaps. In *CAV'08*, volume 5123 of *LNCS*, pages 341–354. Springer, 2008.
3. S. Abiteboul, O. Benjelloun, and T. Milo. Positive Active XML. In *PODS'04*, pages 35–45. ACM, 2004.
4. S. Abiteboul, O. Benjelloun, and T. Milo. The Active XML project: an overview. *VLDB Journal*, 17(5):1019–1040, 2008.
5. S. Abiteboul, L. Segoufin, and V. Vianu. Static analysis of active XML systems. In *PODS'08*, pages 221–230. ACM, 2008. Journal version in *ACM Trans. Database Syst.* 34(4): 2009.
6. A. Blumensath and B. Courcelle. On the monadic second-order transduction hierarchy. HAL Archive, 2009. http://hal.archives-ouvertes.fr/hal-00287223/fr.
7. P. Bouyer, N. Markey, J. Ouaknine, P. Schnoebelen, and J. Worrell. On termination for faulty channel machines. In *STACS'08*, pages 121–132, 2008.
8. C. David. Complexity of data tree patterns over XML documents. In *MFCS '08*, pages 278–289, 2008.
9. A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. Automatic verification of data-centric business processes. In *ICDT'09*, pages 252–267, 2009.
10. A. Deutsch, L. Sui, and V. Vianu. Specification and verification of data-driven web applications. *JCSS*, 73(3):442–474, 2007.
11. A. Deutsch, L. Sui, V. Vianu, and D. Zhou. Verification of communicating data-driven web services. In *PODS'06*, pages 90–99, 2006.
12. A. Deutsch and V. Vianu. WAVE: Automatic verification of data-driven web services. *IEEE Data Eng. Bull.*, 31(3):35–39, 2008.
13. A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *TCS*, 256(1-2):63–92, 2001.
14. X. Fu, T. Bultan, and J. Su. Conversation protocols: a formalism for specification and verification of reactive electronic services. *TCS*, 328(1-2):19–37, 2004.
15. B. Genest, A. Muscholl, O. Serre, and M. Zeitoun. Tree pattern rewriting systems. In *ATVA'08*, pages 332–346. Springer, 2008.
16. R. Hull. Artifact-centric business process models: Brief survey of research results and challenges. In *OTM'08*, pages 1152–1163, 2008.
17. R. Hull, M. Benedikt, V. Christophides, and S. Jianwen. E-services: a look behind the curtain. In *PODS'03*, pages 1–14, 2003.
18. R. Mayr. Undecidable problems in unreliable computations. *TCS*, 297(1-3):337–354, 2003.
19. J. Nešetřil and P. O. de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Comb.*, 27(6):1022–1041, 2006.
20. J. Wang and A. Kumar. A framework for document-driven workflow systems. In *BPM'05*, pages 285–301, 2005.