Verifying Recursive Active Documents with Positive Data Tree Rewriting

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

[1] CNRS, IPAL UMI, joint with I2R-A*STAR-NUS, Singapore

[2] CNRS, IRISA UMR, joint with Universit Rennes I, France

[3] LaBRI, Université Bordeaux/CNRS, France

ABSTRACT. This paper considers a data tree-rewriting framework for modeling documents evolving through services. We focus on automatic verification of properties of documents that can contain data from an infinite domain. We establish the boundaries of decidability: while verifying general documents with arbitrary recursion is undecidable, we obtain decidability as soon as either documents are in the *positive-bounded* fragment (while service calls can be arbitrarily recursive), or when there is a bound on the number of service calls (bounded model-checking of arbitrary documents). In the later case, the complexity is NexpTime-complete. Our data tree-rewriting framework resembles Guarded Active XML, a platform used for handling XML repositories evolving through web services, expressed with data tree pattern guards and queries, while extending it by more flexible modification of documents, namely arbitrary renaming and deletion.

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 each of these topics has been studied extensively on its own, very few papers tackle decidability of algorithms when all aspects are present at the same time.

An interesting platform emerged recently for using 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 an AXML tree T, then from this point on the service cannot be disabled, and calling it can only extend T. In particular, information cannot be deleted and subsequent service calls return answers that extend previous answers. 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 *recursion-free GAXML* w.r.t Tree-LTL properties (an variant of LTL with data tree patterns as atomic formulas) was established in [5]. The crucial restriction needed for decidability there is a uniform bound on the number of possible service calls. For instance, deletion of data is not possible. Compared to [3], service invocation can terminate, and more importantly, negative guards can be used. Even more importantly, verification tasks are more complex and challenging because of the presence of unbounded data. However, the model relies on a rigid semantics (using a workspace etc) of what a service call can do, and how.

In this work, our aim is twofold. First, we aim at extending the GAXML model in a uniform way, by expressing the effect of embedded service calls in form of tree rewriting rules. Our model DTPRS (*data tree pattern rewriting systems*) is based on the same basic ingredients as GAXML, which are tree patterns for guards and queries. However, our formalism allows a user to describe several possible effects of a service call: materialization of implicit data like in (G)AXML, but also deletion and modification of existing document parts. This model is a simplified version of the TPRS model proposed in [16], but it can additionally handle data from infinite domains.

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, 14]. WSTSs are one example of 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, 14]. 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 other hand, we show that the verification of very simple Tree-LTL properties is undecidable even for *positive-bounded* DTPRS, although 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 for (*not necessarily positive bounded*) DTPRS, and show its Co-NexpTime-completeness. Finally, we show that *bounded* model-checking of (not necessarily positive-bounded) DTPRS is NexpTime-complete.

Related work: Verification of web services often ignores unbounded data (c.f. e.g. [18, 15]). On the other hand, several data-driven workflow process models have been proposed. Document-driven workflow was proposed in [21]. Artifact-based workflow was outlined in [17], 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 [12, 11, 13]. This approach has been recently extended to the artifact-based model [10].

On the verification side, there is a rich literature on the verification of well-structured infinite transition systems [1, 14], 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, define the DTPRS model. Then in Section 3, we show that DTPRS with recursive DTD or negated tree patterns are undecidable. In Section 4 we define positive-bounded DTPRS and prove our decidability results. Then in Section 5, we show that 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. Omitted proofs can be found in the appendix, as well as two complete examples.

2 Definitions and notations

In this paper, documents are labeled, unranked, unordered trees. We fix a finite alphabet Σ (with symbols $a, b, c, \dots$) 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 $T = \langle V, E, \text{root}, \ell \rangle$, 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 $\Sigma \cup \mathcal{X} \cup \{*\}$.

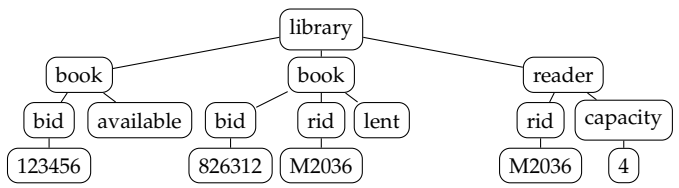

Figure 1: A document for a library system: The reader *M*2036 borrows a book with identifier 826312 from the library, and has capacity 4, namely he or she is able to borrow at most 4 other books.

We now define the different components used in our rewriting rules: *data tree patterns* to locate and specify a pattern of the document, *data constraints* to express equalities and inequalities among variables, *data tree pattern queries* to find (and format) all the valuations of variables satisfying some condition. A *data constraint* is a Boolean combination of relations $X = Y$, with* $X, Y \in \mathcal{X}$. A *data tree pattern* (DTP) $P = \langle V, E, \text{root}, \ell, \tau, \mathit{cond} \rangle$ is a (rooted) $\mathcal{T}$-labeled tree $\langle V, E, \text{root}, \ell \rangle$, together with an edge-labeling function $\tau : E \to \{|, ||\}$ and a data constraint *cond*. $|$-labeled edges denote child edges, and $||$-labeled edges denote descendant edges. 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 $X = Y$ ($X, Y \in \mathcal{X}$) means that

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

the corresponding leaves of P are 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.

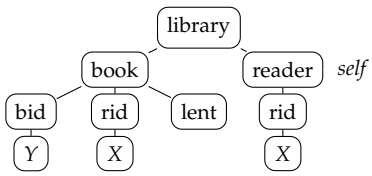

Figure 2: A relative DTP P finding a book Y borrowed by a particular reader designated by *self*.

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* 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 $\mathcal{X}$ (intuitively, *head* is not a constant tree).

Let T be a data tree and $Q = body \rightsquigarrow head$ 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. A *relative* DTPQ is defined like a DTPQ, except that its *body* is a relative pattern. A relative DTPQ Q is evaluated on a pair (T, v). The result of $Q(T, v)$ is defined as for DTPQ, except that matchings of the body must map *self* to v. For instance, the DTPQ $P \rightsquigarrow head$ (where P is given on Figure 2 and *head* is made of a unique node labeled by Y) will return a forest of one-node trees labeled by the books borrowed by the reader designated by *self*.

Our 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. We summarize all the operations related to a data tree rewriting rule into a notion of rule-context called *locator*, as defined in the following.

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 $\Sigma \cup \{*\}$ (Intuitively, information can only be added, as subforests, to Σ-labeled nodes, and a node labeled by a data value cannot be renamed). Label *del* can be attached to any node. Without loss of generality, 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.

A *data tree pattern rewriting rule* (DTP rule) 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,

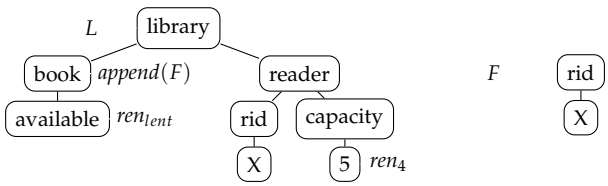

Figure 3: DTP rule "borrow": The reader identifier is added as a subtree to the node "book" whose state is renamed as "lent", and the capacity of the reader is decreased by renaming

- $\mathcal{F}$ 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 set $\mathcal{R}$ of DTP rules and a *static invariant* Δ, consisting of a *DTD* τ and a *data invariant inv*, i.e. a Boolean combination of DTPs. As usual for unordered trees, τ 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 a DTD such that for any $a, b \in \Sigma$, if there is a rule $a \to \psi$ such that b occurs in ψ, then there are no rules $b \to \psi'$ such that a occurs in ψ'.

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 $\mathcal{F} = \{F\}$. The complete library example can be found in the appendix.

We now define formally the semantics of a transition by DTP rules. Let $T = \langle V, E, \text{root}, \ell \rangle$ 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' can be 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) 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 $R \in \mathcal{R}$.

Remarks:

1. The injectivity of matching μ ensures that the outcome T' 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. However, mappings used for guards or queries are 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 wrt. the locator. But notice that this only concerns positive guards.

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[†] *Init*, given as the conjunction of a DTD and a Boolean combination of DTPs, is there some $T \in \mathcal{T}^*_{\mathcal{R}}(Init)$ 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).

Remarks: DTPRS resembles GAXML, while extends it by allowing more flexible modification of XML documents, namely arbitrary renaming and deletion. The appendix includes a detailed illustration of how GAXML can be simulated by DTPRS.

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 even 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

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

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 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.*

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* [16]. 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 [19], 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 decidability we need to consider only positive guards and positive data invariants.

However, from Theorem 2, we know that these restrictions alone do not suffice to achieve decidability. We 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 by adding to V the set of data values occurring in T, and adding to E the links between a leaf labeled by a data value and the node representing that value. Formally, $G(T) = (V', E')$, where $V' = V \cup \{\ell(v) \mid \ell(v) \in \mathcal{D}\}$ and $E' = E \cup \{\{v, d\} \mid \ell(v) = d \in \mathcal{D}\}$. 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 a *positive-bounded DTPRS* 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.

The library example illustrated in Figure 1 (c.f. appendix for the complete example) trivially 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 - lent - 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, 14], which has been applied to DTPRS *without* data in [16]. 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*[‡] *(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 $T_i \preceq T_j$.

Let $(\mathcal{R}, \Delta)$ be a positive-bounded DTPRS.

PROPOSITION 5. *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, $\longrightarrow$ is compatible wrt. $\preceq$ in $\mathcal{T}_{B,K}$, thus $(\mathcal{T}_{B,K}, \longrightarrow, \preceq)$ is a WSTS.

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

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 (see appendix for details). With these computability conditions, Theorem 4 then follows from the properties of WSTS (c.f. Theorem 3.6 and Theorem 4.6 in [14]).

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* [16].

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 $((\Sigma \cup \{\#\}) \times [B+1]) \cup \{\$\}$-labeled (where $[B+1] = \{0, 1, \cdots, B\}$) undirected graph (V', E', ℓ') defined as follows,

- $V' = V \cup \{\ell(v) \mid v \in V, \ell(v) \in \mathcal{D}\}$,
- $E' = E \cup \{\{v, d\} \mid v \in V, \ell(v) = d \in \mathcal{D}\}$,
- Let $v \in V$, if $\ell(v) \in \Sigma$, then $\ell'(v) = (\ell(v), i)$, otherwise, $\ell'(v) = (\#, i)$, where i is the depth of v in T. In addition, $\ell'(d) = \$$ for each $d \in V' \cap \mathcal{D}$.

Let Σ_G denote $((\Sigma \cup \{\#\}) \times [B+1]) \cup \{\$\}$.

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 $i, j : i \neq j$ (tree decompositions in this form are sometimes called ordered tree decompositions).

THEOREM 7. *([20, 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 $[K+1] = \{0, \dots, K\}$. 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 $\theta(u) = v_0 \dots v_K$, then $\eta(u) = (\ell(v_0) \dots \ell(v_K), \overline{\lambda})$, 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 $T_0, T_1, \dots$ be an infinite sequence of data trees from $\mathcal{T}_{B,K}$. Consider the infinite sequence of $\Sigma_{G,K}$-labeled trees $T'_0, T'_1, \dots$ obtained from the tree decompositions (with width K and depth at most K) of graphs $G_\ell(T_0), G_\ell(T_1), \dots$. 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[§]. 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:

[§]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.

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 even 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 TPRS ignoring the zero-tests, and describe them by a Tree-LTL formula $F\varphi$. The proof relies on the universal semantics of Tree-LTL, namely, every run of the TPRS satisfies the formula.

If the *existential* semantics of Tree-LTL formulas is used, i.e. 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 the negations are also forbidden, then the decidability can be achieved.

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

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

where P is a DTP.

6 Type-checking DTPRS

The purpose of this section is to show that it can be checked statically whether DTP rules preserve the static invariant $\Delta = (\tau, inv)$ 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[¶], 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 [9], 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 [‖], 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 show 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 holds in $\leq N$-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.

[¶] if the DTD is recursive, then the problem is undecidable, since the satisfiability of Boolean combinations of DTPs over a recursive DTD is undecidable [9].

[‖] the recursive DTD will quickly lead to undecidability, as argued for the type-checking problem.

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, 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, 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, 2008.

[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] M. Dam. CTL* and ECTL* as fragments of the modal mu-calculus. *Theor. Comput. Sci.*, 126(1):77–96, 1994.

[9] C. David. Complexity of data tree patterns over xml documents. In *MFCS '08*, pages 278–289, 2008.

[10] A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. Automatic verification of data-centric business processes. In *ICDT'09*, pages 252–267, 2009.

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

[12] A. Deutsch, L. Sui, V. Vianu, and D. Zhou. Verification of communicating data-driven web services. In *PODS'06*, pages 90–99, 2006.

[13] A. Deutsch and V. Vianu. WAVE: Automatic verification of data-driven web services. *IEEE Data Eng. Bull.*, 31(3):35–39, 2008.

[14] A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *TCS*, 256(1-2):63–92, 2001.

[15] 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.

[16] B. Genest, A. Muscholl, O. Serre, and M. Zeitoun. Tree pattern rewriting systems. In *ATVA'08*, pages 332–346, 2008.

[17] R. Hull. Artifact-centric business process models: Brief survey of research results and challenges. In *OTM'08*, pages 1152–1163, 2008.

[18] R. Hull, M. Benedikt, V. Christophides, and S. Jianwen. E-services: a look behind the curtain. In *PODS'03*, pages 1–14, 2003.

[19] R. Mayr. Undecidable problems in unreliable computations. *TCS*, 297(1-3):337–354, 2003.

[20] J. Nešetřil and P. O. de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Comb.*, 27(6):1022–1041, 2006.

[21] J. Wang and A. Kumar. A framework for document-driven workflow systems. In *BPM'05*, pages 285–301, 2005.

A The simulation of GAXML by DTPRS

DTPRS resembles Guarded Active XML (GAXML) [5], and extends GAXML by allowing more flexible modification of XML documents, namely arbitrary relabeling and deletion. To illustrate how GAXML is simulated by DTPRS, we first recall the salient features of GAXML in the following.

GAXML incorporates into XML documents embedded function calls, whose call and return are guarded by a Boolean combination of tree patterns. In GAXML, function calls can be internal or external. More specifically, a GAXML system is a tuple $(\Phi_{int}, \Phi_{ext}, \Delta)$, where Φ_{int}, Φ_{ext} are respectively a set of internal and external function calls, and Δ specifies the static constraints, including a DTD and a Boolean combination of DTPs (data invariant).

An internal function f includes four components,

- a call guard which is a Boolean combination of (relative) DTPs,
- an argument query which is a (relative) DTP query,
- a return guard which is a Boolean combination of DTPs rooted at a_f,
- the return query which is a DTP query rooted at a_f.

External functions are similar to internal functions, except that the return guard and the return query are missing, thus external functions may return an arbitrary forest that is consistent with the static invariant Δ. A (internal or external) function f can be continuous or non-continuous, in the sense that if it is non-continuous, a call to f is deleted once the result is returned, otherwise, the call is kept after the result is returned, so f can be called again.

The call of an internal function f at a node n (labeled by $!f$) in GAXML goes as follows:

> If the call guard G is satisfied, then the argument query Q is evaluated. The evaluation result of Q is a tree T_f with the root labeled by a_f, stored temporarily into a workspace, and the node n is relabeled into $?f$.

Here is how a call of an internal function f can be modeled in DTPRS: The associated DTP rule (see Figure 4) has the same guard G, the set of queries $\mathcal{Q} = \{Q\}$, $\mathcal{F} = \{F_1, F_2\}$, and χ maps the $!f$-node to F_2 and the WS-node to F_1. Applying the DTP rule amounts to evaluating Q to get the arguments of the call, writing them into the workspace WS, creating a fresh identifier X that it copied both below WS and below the node with the function call (aka return address for f), and renaming $!f$ by $?f$.

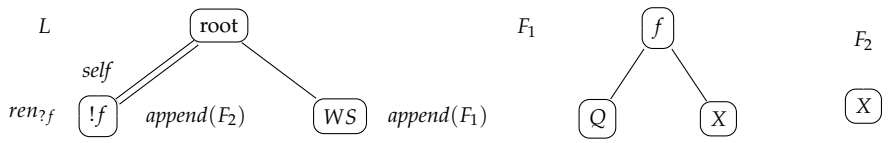

Figure 4: The DTP rule for the call of an internal function

After a function f is called, the function calls embedded in the tree T_f in the workspace can be called before the return of f.

The return of an internal function f at a node n (n is labeled by $?f$) in GAXML goes as follows:

> If the return guard G is satisfied over T_f in the workspace, then evaluate the return query Q of f on T_f, and add the resulting forest as a sibling of the node n,

remove T_f from the workspace, and relabel n by $!f$ if f is continuous, otherwise remove n.

Here is how a return of a continuous internal function f can be simulated in DTPRS.

The associated DTP rule (see Figure 5) has the same guard G, the set of queries $\mathcal{Q} = \{Q\}$, $\mathcal{F} = \{F\}$, and χ maps the node labeled *append* with F. This DTP rule locates where T_f is in the workspace WS using X, evaluates the return query Q to get the return of the call, puts the result of the Q as a sibling of $?f$, deletes the associated data in the workspace, as well as the identifier X, and renames $?f$ by $!f$.

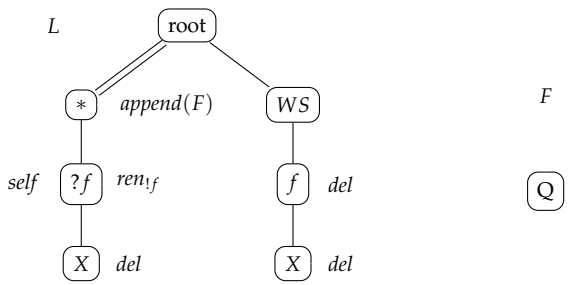

Figure 5: The DTP rule for the return of a continuous internal function

The rules for other kinds of functions are similar.

B The library example

A library system consists of a collection of books and a collection of readers. Each book and each reader have a unique identifier. The system includes the following five functionalities.

1. The reader can borrow at most 5 books from the library.
2. The reader can return the borrowed books to the library.
3. A new book can be registered into the library.
4. A new reader can register in the library and a new account with a unique identifier will be created.
5. The account of a reader can be deleted, if all the books borrowed by a reader have been returned (namely, the capacity is 5).

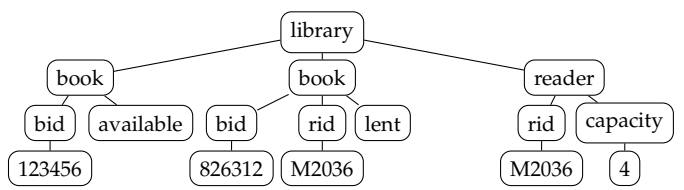

Figure 6: A document for the library system: The reader $M2036$ borrows a book with identifier 826312 from the library, and has capacity 4, namely he or she is able to borrow at most 4 other books.

Notice that here for simplicity, we do not include the other information (e.g. names) about books and readers, neither put a bound (e.g. one month) on the time between bor-

rowing and return of books. But the example presented here can be easily extended to take these into consideration.

The DTD of the library system is illustrated in Figure 6. Note that the capacity levels $\{0, 1, 2, 3, 4, 5\}$ are treated as the tags in Σ, instead of the data values in $\mathcal{D}$.

In the following, we present the DTPRS rules corresponding to the five functionalities.

1. There are five DTP rules borrow-i ($i = 1, 2, 3, 4, 5$) for a reader with capacity i to borrow a book from the library. The rule borrow-5 is illustrated in Figure 7, including the locator L and the $\mathcal{F} = \{F\}$: The state of the node "book" is renamed from "available" to "lent", a tree F containing the identifier of the reader is appended as its subtree, and the capacity of the reader is decreased by renaming 5 into 4. The other borrow rules can be defined similarly.

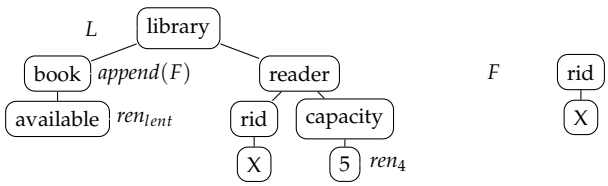

Figure 7: The rule borrow-5

2. There are five DTP rules return-i ($i = 0, 1, 2, 3, 4$) for a reader with capacity i to return a book to the library. The rule return-4 is illustrated in Figure 8: "lent" is renamed into "available", the *rid* subtree is deleted, and the capacity of the reader is increased. The other return rules can be defined similarly.

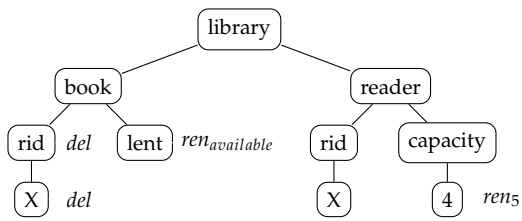

Figure 8: The rule return-4

3. The DTP rule new-book (see Figure 9) adds a new book into the library, generating a unique identifier for it.

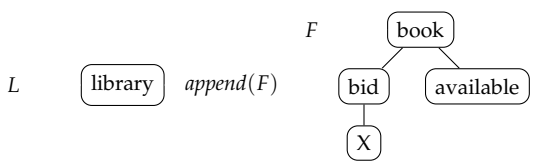

Figure 9: The rule new-book

4. The rule new-reader (see Figure 10) adds a new reader into the library, generating a unique identifier for it.

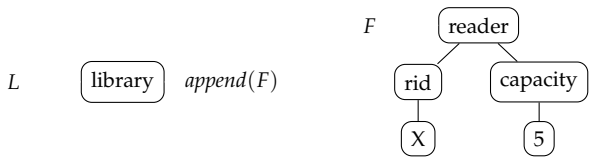

Figure 10: The rule new-reader

5. The rule del-reader (see Figure 11) checks whether the capacity of a reader is 5, if so, it deletes the account of the reader.

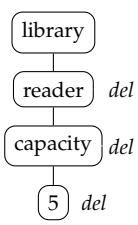

Figure 11: The rule del-reader

The initial document tree for the library system is a data tree containing no books and no readers. The books and readers can be added one by one into the library by DTP rules new-book and new-reader.

The library example is a positive bounded DTPRS with the length of simple paths bounded by 7: A longest path is for instance: library - book - rid - $M2036$ - rid - reader - capacity - 4. Moreover, this holds even when the restriction that each reader can borrow at most a bounded number of books is dropped.

C MailOrder example

This section includes an example to model a MailOrder system for *Play.com* in DTPRS. For simplicity, we present only what happens on the *Play.com* peer, although we could also model client peers, bank peers etc. The *Play.com* example can be compared with the *MailOrder* example in [5]. Syntactically, GAXML uses guards and queries. Most of the time, guards and queries are very simple and can be encoded in the locator of DTP rules. In this case, we omit the *self* label in our rules. Unlike *MailOrder* example in [5], we can express deletion with DTPRS, and thus model the selection of the products in the cart (adding and deleting products), and also handle an inventory (how many items of a product remain - each time an item is added to a cart, it is also deleted from the inventory). More importantly, compared with the recursion-free decidable restriction of GAXML, we are able to represent the process of many customers ordering many different products in our decidable fragment.

On the *Play.com* peer, there are a product catalog, a customer catalog, a set of carts and a set of orders. The inventory is encoded in the product catalog: If there are three items of a product in the inventory, then there are three tokens as children of the product. Each cart is associated with a customer (at first anonymous, and he can later login in as a regis-

tered member). The cart is first in the select mode, which allows the associated customer to add/delete products. Then the customer can check out, the cart gathers the different prices for the products into a bill, and goes into the payment mode. When the customer pays, a corresponding order is created with a receipt, the customer is disconnected and the cart is deleted.

A simple example of a document of *Play.com* is illustrated in Figure 12. We represent a data value only when it is used by at least two different nodes.

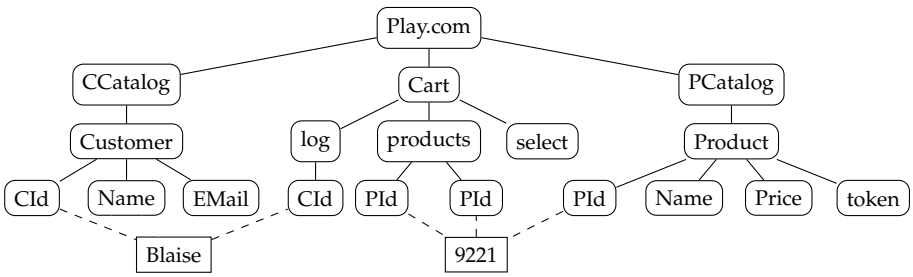

Figure 12: A document of the *Play.com* system, where the registered customer Blaise has two items of the same product in his cart, with PId 9221. There is one additional product with PId 9221 left (there remains one token).

Some key rewriting rules are described in the following. We only describe nontrivial components in these rules.

- An anonymous customer can connect to *Play.com* with the rule *create-cart* (Figure 13).

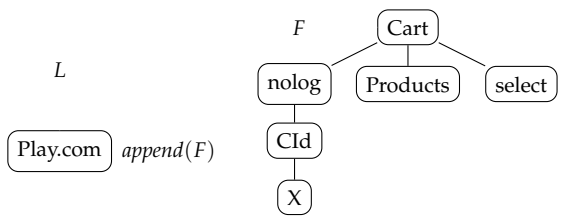

Figure 13: The rule create-cart

- An anonymous customer can login as a registered member with the rule *login* (Figure 14). As the customer peer is not modeled, we do not handle the check of a password, although it would not be a problem to do so in our framework.

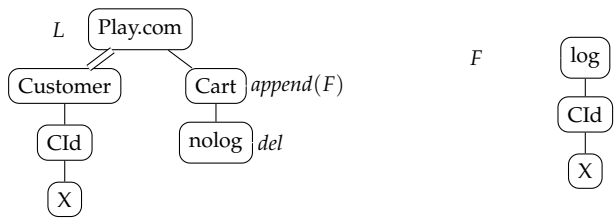

Figure 14: The rule login

- The rule *Add-Product* (Figure 15) checks the cart is in select mode and adds a new product into the cart (and deletes a token from the inventory).

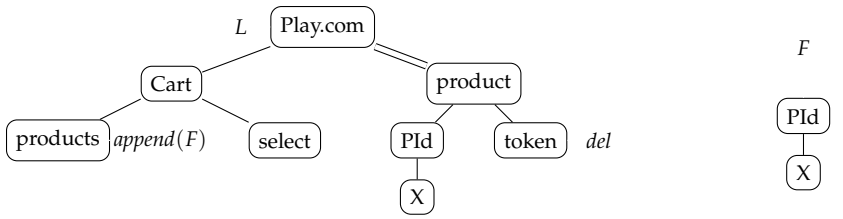

Figure 15: The rule Add-Product

- The rule *Delete-Product* (Figure 16) deletes a product from the cart (and puts the token back).

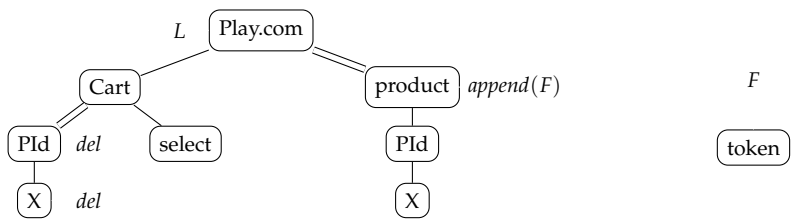

Figure 16: The rule Delete-Product

- The rule *Check-out*(Figure 17) checks whether the cart is nonempty and retrieves the prices of products in the cart into a bill through a query. It changes the mode of the cart from *select* to *payment*. Here $Q = body \rightsquigarrow Y$ with *body* illustrated in the figure.

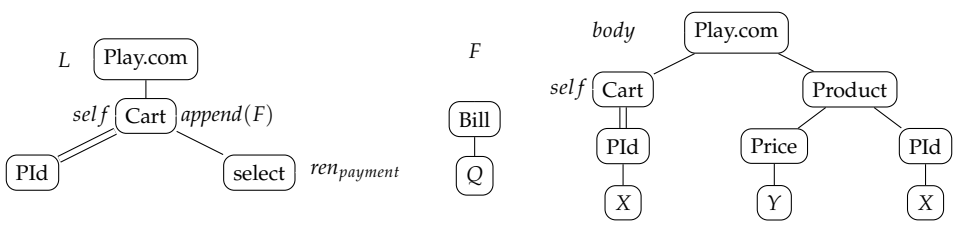

Figure 17: The rule Check-Out

- The customer can pay with the rule *Pay* (Figure 18), and a corresponding order is created. For simplicity, it disconnects the customer, and transforms the cart into an order. The order contains the customer ID, an order ID (a fresh unique identifier), and the total price (sum of the prices of each items). As we model prices by data values and we do not use any arithmetics, the total price is a fresh data value. The only important thing is that this data value *Total* is the same as the one registered in the bank account, which we could check for equality (although we do not explicitly model the bank here). The order does not recall the individual PId of products since there will be no more products to put back in the inventory (and anyway the product can be later removed from the catalog).
- The rule *Add-member* (Figure 19) allows a customer to register as a member.

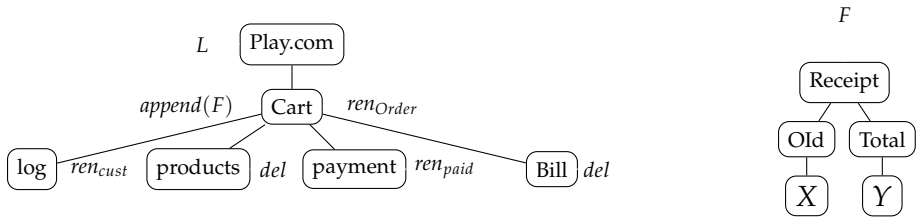

Figure 18: The rule Pay

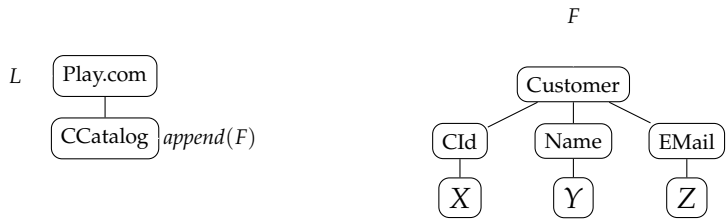

Figure 19: The rule Add-member

We do not specify the following rules here, which are easy to come up with: *shipped*, *delivered*, *add product to catalog* etc.

The *Play.com* example satisfies the first 2 conditions of the positive-bounded DTPRS. However, in general, the third condition is not satisfied. PIds can create a long path: a cart can be linked to a product, linked to another cart linked to another products etc. So the number of carts or the number of products needs to be bounded (unless a cart can contain at most one product). On the other hand, Name and Total are fresh data values, they cannot be used as links. At last, CId can be used in different carts and orders, but as a cart or order is associated to a unique customer, it cannot create long paths. More formally, if the system can handle only C active carts at a time (but the number of orders is unlimited), then the system has simple paths bounded by $12C + 7$. If there are at most D different products in the catalog, then the system has simple paths bounded by $12D + 7$. Finally, if each customer can have only one active cart at a time (but she can have many orders), and each cart has at most one product, then the system has simple paths bounded by 14.

D Proofs in Section 3

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 even without data.

PROOF. In both cases we encode a 2-counter machine with counters C_1, C_2. In the first one, a configuration $(q, n_1, n_2) \in Q \times \mathbb{N} \times \mathbb{N}$ is encoded by a tree with root labeled q and two subtrees, one of the form $a^{n_1}a_\$$, and the other of the form $b^{n_2}b_\$$. E.g. a zero test on the first counter corresponds to checking that the root has a child labeled $a_\$$. Decrementing

the first counter in state q (and going to state q') is done using the DTP rule (without data) illustrated in Figure 20.

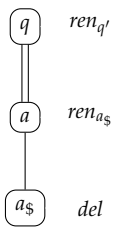

Figure 20: DTP rule for a transition in which the first counter is decremented

With non-recursive DTDs we can encode a configuration (q, n_1, n_2) by a tree of depth one, with root labeled by q, and n_1 (n_2, resp.) leaves labeled by a (b, resp.). The zero test is now done using a negative guard (e.g. "no a-leaf") or a negative invariant. In the latter case we split a transition in 2 steps: first we relabel the root by a transition from that state; second, we perform the corresponding rewriting as before. The invariant states that whenever the root is labeled by a transition corresponding to a zero test of counter C_1 (resp. C_2), the tree has no a-leaf (resp. b-leaf). ∎

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 in Δ are positive.*

PROOF. We reduce Post correspondence problem (PCP) first to pattern reachability. We may assume that our PCP instance $(u_i, v_i)_{1 \le i \le n}$ is such that the following holds for every non-empty sequence $i_1, \ldots, i_k$ of indices:

- Either $U = u_{i_1} \cdots u_{i_k}$, $V = v_{i_1} \cdots v_{i_k}$ are incomparable, or V is a prefix of U. In the latter case we call (U, V) a partial solution.
- If (U, V) and (Uu_i, Vv_i) are partial solutions and $U \neq V$, then either $Uu_i = Vv_i$ or Vv_i is a prefix of U.
- Every solution starts with with the pair (u_1, v_1) and ends with (u_n, v_n).

It is not hard to verify that the usual Turing machine reduction to PCP satisfies the restrictions above.

A partial solution (U, V) with $U = a_1 \cdots a_n$, $V = a_1 \cdots a_{m-1}$ will be represented by the data tree in Figure 21. In this data tree the leaves are labeled by data d_i, with $d_i \neq d_j$ for all $i \neq j$. Moreover, notice that the last position has the special marker \$, and the first position in $U \setminus V$ has the special marker #.

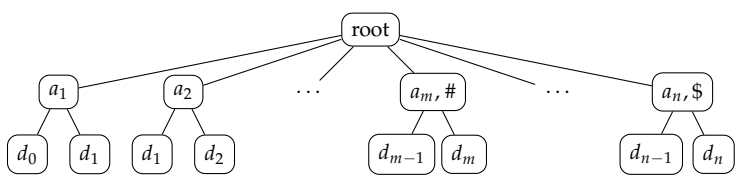

Figure 21: data tree for the reduction of PCP to DTPRS

With each PCP pair (u_i, v_i), $i < n$, we associate DTPRS rules $R_i = \langle L, G, \mathcal{Q}, \mathcal{F}, \chi \rangle$. For simplicity we describe below the locator L and the forest $\mathcal{F}$ for $(u_i, v_i) = (aba, bb)$ (the guard G and the query set $\mathcal{Q}$ are both empty):

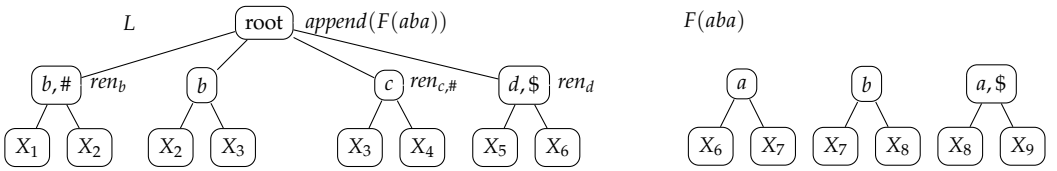

Figure 22: The rule R_i

We need a rule R_i (see Figure 22) for each pair of tags $c, d \in \Sigma$ (these are the tags at positions n and $m + 2$ in the example with $|v_i| = 2$). Notice that variable X_6 occurs in both L and $F(aba)$, whereas X_7, X_8, X_9 will take fresh (and mutually distinct) values.

The pair (u_n, v_n) has similar rules, except that we will not append any forest to the root, but rename the root with a special marker $\surd$. The initial tree T_0 is defined as expected, from (u_1, v_1). The PCP instance has a solution iff we can reach a data tree with root label $\surd$. Notice that all guards and the invariant are empty.

For termination we can modify the above proof in order to ensure that executions that do not correspond to partial solutions, are infinite. More precisely, if U, V as above is a partial solution, but Uu_i, Vv_i is not, then we use a DTP rule associated with (u_i, v_i) that forces an infinite execution. In this way, termination will hold iff the PCP instance has a solution. ∎

E Proofs in Section 4

E.1 Proof of Proposition 5

Proposition 5. *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 $T_2 \preceq T_2'$.*

PROOF. Let $R = \langle L, G, \mathcal{Q}, \mathcal{F}, \chi \rangle$. Taking an injective mapping $\phi : T_1 \to T_1'$ preserving the root, parent-child relation, tag, and data (in)equality relation, and an injective matching $\psi : L \to T_1$ satisfying the data constraint *cond* of L, we have an injective matching $\phi \circ \psi : L \to T_1'$ which respects the parent-child, tags and data (in)equality relation. Hence *cond* is satisfied by $\phi \circ \psi$ too. As G is positive, if G is true at T_1 wrt. ϕ, then it is true at T_1' wrt. $\phi \circ \psi$ as well. Applying the rule R to T_1' wrt. $\phi \circ \psi$, we get a tree T_2' such that $T_2 \preceq T_2'$. As both the DTD and the data invariant in Δ are positive and T_2 fulfills Δ, so does T_2'. Thus $T_1' \xrightarrow{R} T_2'$. ∎

E.2 Additional computability conditions for $(\mathcal{T}_{B,K}, \longrightarrow, \preceq)$

First consider pattern reachability. To get the decidability of this problem, from Theorem 3.6 in [14], we need to show that $(\mathcal{T}_{B,K}, \longrightarrow, \preceq)$ has effective pred-basis. A WSTS $(S, \longrightarrow, \preceq)$ has *effective pred-basis* if there exists an algorithm that computes for any state $s \in S$ the finite

basis $pb(s)$ of the upward closed set $\uparrow Pred(\uparrow s)$. Here, $\uparrow I = \{s' \in S \mid \exists s \in I \text{ s.t. } s \preceq s'\}$ denotes the upward closure of I wrt. $\preceq$, and $Pred(I) = \{s \in S \mid \exists t \in I, s \longrightarrow t\}$ the set of immediate predecessors of states in I. A basis of an upward-closed set I is a minimal set I^b such that $I = \bigcup_{x \in I^b} \uparrow x$. Recall that whenever $\preceq$ is a wqo, the basis I^b of an upward closed set I is finite.

PROPOSITION 14. *$(\mathcal{T}_{B,K}, \longrightarrow, \preceq)$ has effective pred-basis.*

PROOF. It is sufficient to consider $\min(Pred(\uparrow T))$, the set of minimal elements wrt. $\preceq$ in $Pred(\uparrow T)$, for each tree $T \in \mathcal{T}_{B,K}$.

Fix a rule $R = \langle L, G, \mathcal{Q}, \mathcal{F}, \chi \rangle$ with

$$L = \langle V_L, E_L, root_L, \ell_L, \tau_L, cond_L, \ell'_L \rangle$$

such that ℓ'_L attaches additional labels $\{append, ren_a, del\}$, $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$ ($Q_i = body_i \rightsquigarrow head_i$), and $\mathcal{F} = \{F_1, \ldots, F_n\}$.

Let $T_1 = \langle V_1, E_1, root_1, \ell_1 \rangle \in \min(Pred(\uparrow T))$, then

$\exists T'$ such that $T_1 \xrightarrow{R} T'$ with respect to some $\phi : L \to T_1$ and $T \preceq T'$ via some $\psi : T \to T'$. $(*)$

In the following, we show that the size of T_1 (number of nodes) is bounded by the following constant (we actually show even more, by exhibiting T_1 satisfying Δ):

$$B^2 \left((|\Sigma| + 1) \max(\Delta)\right)^B \left(|L| + |G| + |T| \max_i |Q_i|\right),$$

where $\max(\Delta)$ is the maximum integer used in the definition of DTD in Δ, and $\max_i |Q_i|$ is the maximum size (the sum of the size of the body and the head) of DTPQs in $\mathcal{Q}$.

Thus a finite basis, which is a finite subset of $\min(Pred(\uparrow T))$, is computable.

Let $T' = \langle V', E', root', \ell' \rangle$. Then V' consists of four disjoint subsets,

- $V'_1 = \{\phi(v) \mid v \in V_L, v \text{ not labeled by } del\}$,
- $V'_2 = node^{-1}(V'_1)$, where $node^{-1}(V'_1)$ is the set of nodes $w \in V_1 \setminus \phi(V_L)$ such that the lowest ancestor of w in $\phi(V_L)$ is in V'_1.
- V'_3 contains distinct copies of F_j, excluding the leaves labeled by those Q_i,
- V'_4 contains distinct copies of the nodes of the forest $Q_i(T_1)$, one for each node labeled by Q_i in each copy of F_j.

The node set of T_1 consists of V'_1, V'_2, $V_3 = \{\phi(v) \mid v \in V_L, v \text{ labeled by } del\}$, and $V_4 = node^{-1}(V_3)$.

Now we consider an upper bound on the size of T_1 that are sufficient to allow T_1 satisfying $(*)$,

- To guarantee the matching ϕ from L to T_1:
  The nodes in $V'_1 \cup V_3 = \phi(V_L)$ and all their ancestors in T_1 are sufficient.
  Note that in L, ancestor relations $||$ may occur, so the inclusion of the ancestors of nodes in $V'_1 \cup V_3 = \phi(V_L)$ is necessary.
  Size: $B|\phi(V_L)| = B|L|$;
- To witness that G is satisfied over T_1 wrt. ϕ:
  G is a positive Boolean combination of DTPs. To witness the satisfaction of each DTP

P_i in G, we need keep a matching ϕ_i from P_i to T_1 and all the ancestors of nodes of $\phi_i(P_i)$ in T_1.
Size: $B|G|$;

- To guarantee that $T \preceq T'$:
  - Keep $(V_1' \cup V_2') \cap \psi(V_T)$ and all their ancestors in T_1,
  - At most $|T|$ instantiations of $head_i$ on T_1 by matchings from $body_i$ to T_1 wrt. ϕ are sufficient. The ancestors of all the nodes of T_1 in these instantiations should be preserved as well.

  Size: $B|T| + B|T||body_i| \leq B|T| \max_i |Q_i|$.
- Finally, to satisfy the DTD in Δ, T_1 should be completed into a data tree of size at most (c.f. [5])

$$B \cdot (|(\Sigma| + 1) \max(\Delta))^B |T_1|.$$

Thus a sufficient upper bound for the size of T_1 is

$$B^2 ((|\Sigma| + 1) \max(\Delta))^B \left(|L| + |G| + |T| \max_i |Q_i| \right).$$

∎

A solution for reachability of a given DTP P from an initial set of data trees *Init* is obtained by backward exploration: we start with I^0 as the set of data trees matching P and satisfying Δ. Then compute iteratively the upward closed sets $I^{n+1} = I^n \cup (Pred(I^n) \cap \Delta)$ by representing each set through its finite basis. Since the sequence I^n is increasing by construction, and since $\preceq$ is a wqo, the sequence must be finite and termination can be effectively tested (c.f. Lemma 2.4 in [14]). If $I^n = I^{n+1}$ then it suffices to check whether $I^n \cap Init = \emptyset$. Notice that we did not impose any restriction on the set *Init* of the initial trees. We need to test the existence of a data tree from $\mathcal{T}_{B,K}$ satisfying an (arbitrary) Boolean combination on DTPs *and* an (arbitrary) DTD. This problem is in general undecidable [9], but becomes decidable in the special case where trees are of bounded depth [9]. Here we need to talk in addition about trees from $\mathcal{T}_{B,K}$, but we can apply the same proof ideas as in [9] in order to infer decidability.

Now consider the termination problem. From Theorem 4.6 in [14], to show the decidability of termination problem from a single initial tree T_0, it is sufficient to show that $(\mathcal{T}^*_{\mathcal{R}}(T_0), \longrightarrow, \preceq)$ has effective *Succ*, i.e. for each $T \in \mathcal{T}^*_{\mathcal{R}}(T_0)$, the set $Succ(T) := \{T' \mid T \longrightarrow T'\}$ is computable. Then we can compute the *finite reachability tree* starting with T_0: we compute trees T s.t. $T_0 \xrightarrow{*} T$ and we stop whenever we find $T \preceq T'$ along some branch.

It is not hard to see that $Succ(T)$ contains only a finite number of equivalence classes induced by the quasi-order $\preceq$. Since the DTPRS $(\mathcal{R}, \Delta)$ is not able to distinguish between two distinct data trees belonging to the same equivalence class, by selecting one data tree from each equivalence class, we can get a finite representation of $Succ(T)$, therefore, $(\mathcal{T}^*_{\mathcal{R}}(T_0), \longrightarrow, \preceq)$ has effective *Succ*.

E.3 Tree decompositions

Theorem 7. ([20, 6]) *If $G \in \mathcal{G}_K$, then G has a tree decomposition with both depth and width bounded by K.*

PROOF. Let $G = (V, E, \ell) \in \mathcal{G}_K$ and $T = \langle V, E_T, r\rangle$ be a depth-first-search tree of G with $r \in V$ as the root. Then T is of depth at most K. For each $v \in V$, let $\theta(v)$ be the union of $\{v\}$ and the set of all ancestors of v in T, then $\langle V, E_T, r, \theta\rangle$ is a tree decomposition of G of depth at most K and width at most K. ∎

As a matter of fact, the converse of Theorem 7 holds as well.

PROPOSITION 15. *If G has a tree decomposition of width $\leq A$ and depth $\leq B$, then the length of any simple path of G is bounded by $(A+2)^B + \sum_{1\leq i\leq B}(A+2)^i$.*

PROOF. Let $G = (V, E)$ and $T = \langle W, F, r, \theta\rangle$ be a tree decomposition of G of width at most A and depth at most B.

Let $P = v_1 \cdots v_n$ be a path in G, and $w_1 \cdots w_n$ be a trace of P in T such that $v_i \in \theta(w_i)$, $w_i = w_{i+1}$ or there is a path in T from w_i to w_{i+1} such that for each $w \neq w_{i+1}$ on the path, $v_i \in \theta(w)$.

Because all bags are of size at most $A+1$, each bag can only occur at most $A+1$ times on the sequence $w_1 \cdots w_n$.

Let B_0 be the minimal depth of w_i's. Then there is only one bag at depth B_0, say w, occurring on the sequence $w_1 \cdots w_n$.

Let $w_{i_1}, \cdots, w_{i_l}$ ($l \leq A+1$, $i_j < i_{j+1}$) be all the occurrences of w on the sequence $w_1 \cdots w_n$. Then all the bags on each sub sequence $w_{i_j+1} w_{i_j+2} \cdots w_{i_{j+1}-1}$ is at depth no less than $B_0 + 1$. By induction hypothesis, each subsequence $w_{i_j+1} w_{i_j+2} \cdots w_{i_{j+1}-1}$ is of length at most

$$(A+2)^{B-B_0-1} + \sum_{1\leq i\leq B-B_0-1}(A+2)^i,$$

thus

$$\begin{aligned} n &\leq l + (l+1)\left((A+2)^{B-B_0-1} + \sum_{1\leq i\leq B-B_0-1}(A+2)^i\right) \\ &\leq (A+2)\left(1 + (A+2)^{B-1} + \sum_{1\leq i\leq B-1}(A+2)^i\right) \\ &= (A+2)^B + \sum_{1\leq i\leq B}(A+2)^i. \end{aligned}$$

∎

So generally speaking, for a class of graphs, all simple paths are length-bounded for each graph in the class iff there is a tree decomposition of bounded depth and width for each graph in the class.

E.4 Well-quasi-ordering of data trees

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 $G_1 \sqsubseteq G_2$.*

PROOF. Let $G_i = (V_i, E_i, \ell_i)$, $T_i = \langle U_i, F_i, r_i, \theta_i\rangle$ and $T_i' = \langle U_i, F_i, r_i, \eta_i\rangle (i = 1, 2)$. Suppose that $T_1' \leq T_2'$. Then there is an injective mapping ϕ from U_1 to U_2 preserving the root, the parent-child relation and the node-labels.

Define an injective mapping $\pi : V_1 \to V_2$ as follows:

For $v \in V_1$, select some $u \in U_1$ such that $\theta_1(u) = v_0 \dots v_K$ and $v = v_i$ for some i. Writing $\theta_2(\phi(u)) = v'_0 \dots v'_K$, we let $\pi(v) = v'_i$.

First we show that π does not depend upon the choice of i such that $v = v_i$, neither on the choice of $u \in U_1$ such that $v \in \theta_1(u)$. The former holds because $\eta_1(u) = \eta_2(\phi(u))$ (and in particular the component λ_1 is preserved), hence if $v_i = v_j$, then we also have $v'_i = v'_j$.

For the latter, notice that the λ_3 component of $\eta_1(u) = \eta_2(\phi(u))$ is preserved, hence the choice of u or of its father is irrelevant. Now, the set $\{u \in U_1 \mid v \in \theta_1(u)\}$ is a connected subgraph of T_1 by definition of tree decomposition, hence π does not depend upon the choice of $u \in U_1$.

Now we show that π is injective. Let v_2 be a vertex of G_2. Because of the preservation of λ_1, no two different vertices v, v' of G_1 with $v, v' \in \theta(u)$ can satisfy $\pi(v) = \pi(v') = v_2$. Because of the preservation of λ_3, no two different vertices v, v' with $v \in \theta(u)$ and $v' \in \theta(u')$ with u father of u' can satisfy $\pi(v) = \pi(v') = v_2$. Again, as the set $\{u \in U_1 \mid v_2 \in \theta(u)\}$ is a connected subgraph of T_2, it means that π is injective.

We finish the proof by showing that π preserves the node-labels and edge relations.

Node-label preservation: Suppose $\pi(v) = v'$. Then there exists some $u \in U_1$ such that $\theta_1(u) = v_0 \cdots v_k$, $v = v_i$ for some i, $\theta_2(\phi(u)) = v'_0 \cdots v'_k$, and $v' = v'_i$. Since $\eta_1(u) = \eta_2(\phi(u))$, $\ell_1(v_0) \dots \ell_1(v_k) = \ell_2(v'_0) \dots \ell_2(v'_k)$, it follows that $\ell_1(v) = \ell_1(v_i) = \ell_2(v'_i) = \ell_2(v')$.

Edge relation preservation: We show that $\{v, w\} \in E_1$ iff $\{\pi(v), \pi(w)\} \in E_2$ for any $v, w \in V_1$.

If $\{v, w\} \in E_1$, there exists $u \in U_1$ such that $\theta_1(u) = v_0 \cdots v_k$, $v = v_i$ and $w = v_j$ for some i, j. So $(i, j) \in \lambda_2(u)$ in T'_1. Then $(i, j) \in \lambda_2(\phi(u))$. Let $\theta_2(\phi(u)) = v'_0 \cdots v'_k$, then $\{v'_i, v'_j\} \in E_2$. Consequently $\{\pi(v), \pi(w)\} = \{v'_i, v'_j\} \in E_2$.

If $\{\pi(v), \pi(w)\} \in E_2$, then there exists $u' \in U_2$ such that $\pi(v), \pi(w) \in \theta_2(u')$. Without loss of generality, we can choose u' at minimal depth such that $\pi(v), \pi(w) \in \theta_2(u')$. It means that for instance, the father u'' of u' satisfies $\pi(v) \notin \theta_2(u'')$. Since $U'_2 = \{u''' \in U_2 \mid \pi(v) \in \theta_2(u''')\}$ is connected, it means that U'_2 is entirely contained in the subtree rooted at u'. By contradiction, if there does not exist $u \in U_1$ such that $\phi(u) = u'$, then $\phi(U_1) \cap U'_2 = \emptyset$. On the other hand, according to the definition of π, there is $u \in U_1$ such that $v \in \theta_1(u)$ and $\pi(v) \in \theta_2(\phi(u))$. So $\phi(u) \in \phi(U_1) \cap U'_2$, a contradiction. Thus there is $u \in U_1$ such that $u' = \phi(u)$. Let $\theta_1(u) = v_0 \dots v_k$ and $\theta_2(u') = v'_0 \dots v'_k$, by injectivity of π, we have $\pi(v) = v'_i$, $\pi(w) = v'_j$, $v = v_i$, $w = w_j$ for some i, j. Then $(i, j) \in \lambda_2(u') = \lambda_2(u)$, which proves that $\{v, w\}$ is an edge of G_1. ∎

F Proofs in Section 5

At first, we briefly recall the syntax and semantics of Tree-LTL defined in [5]. In fact we will only consider a fragment of Tree-LTL defined in [5], namely Tree-LTL formulas without free variables, defined by the following rules,

$$\varphi := P \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi_1 \mid X\varphi_1 \mid \varphi_1 U \varphi_2,$$

where P is a DTP.

Intuitively, Tree-LTL is just the extension of LTL to data trees by replacing atomic propositions with DTPs.

The semantics of Tree-LTL formulas are defined universally: Given a DTPRS $(\mathcal{R}, \Delta, Init)$ and a Tree-LTL formula φ, $(\mathcal{R}, \Delta, Init)$ is said to satisfy φ iff every run of $(\mathcal{R}, \Delta, Init)$ satisfies φ.

Symmetrically, the semantics of Tree-LTL can be defined existentially, namely, a DTPRS satisfies a Tree-LTL formula iff there exists a run of the DTPRS satisfying the formula. We call the Tree-LTL with existential semantics as existential Tree-LTL.

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 even without data.*

PROOF.

We show this by a reduction from the halting problem of two-counter machines.

Let $M = (Q, \Sigma, \Delta, q_0, H)$ be a two-counter machine such that

- $Q = \{q_0, \ldots, q_m\}$,
- Σ: input alphabet,
- q_0: the initial state,
- $H \subseteq Q$: the halting states,
- $\Delta \subseteq Q \times \Sigma \times \{=0, \neq 0, *\} \times \{=0, \neq 0, *\} \times Q \times \{+1, -1, \cdot\} \times \{+1, -1, \cdot\}$, where $*$ means the condition is unspecified, and $\cdot$ means the action is unspecified.

Without loss of generality, it can be assumed that M is deterministic in the following sense,

> for any configuration $(q, \ldots)$which is reachable from the initial configuration, $(q_0, \ldots)$, it holds that either $q \in H$, or else there is a *unique* transition in Δ which can be applied to yield another configuration.

The general idea is to simulate M by a *positive bounded* DTPRS ignoring the zero-tests, and describe the zero-tests in a Tree-LTL formula $F\varphi$.

Recall that the semantics of Tree-LTL defined in [5] is universal, namely, a DTPRS satisfies a Tree-LTL formula ψ if all the runs in the DTPRS satisfy ψ.

Since the halting problem for the two-counter machines with the empty input string is undecidable, the input can be ignored in the reduction and it can be assumed that the transitions Δ of M is of the form

$$\Delta \subseteq Q \times \{=0, \neq 0, *\} \times \{=0, \neq 0, *\} \times Q \times \{+1, -1, \cdot\} \times \{+1, -1, \cdot\}.$$

Moreover, let $\Delta = \{\delta_1, \cdots, \delta_n\}$.

Now we define a positive bounded TPRS $(DTD, \mathcal{R}, Init)$ and a Tree-LTL formula $F\varphi$ such that

$$(DTD, \mathcal{R}, Init) \models F\varphi \text{ iff } M \text{ halts on the empty string } \varepsilon.$$

The DTD is illustrated in Figure 23.

Recall that DTD is required to be positive in positive bounded DTPRS, so what we can do is to guarantee that there is at least one child of the root labeled by q, etc. It follows that the tree in which there are two children of the root labeled by q, q' respectively satisfies also the DTD.

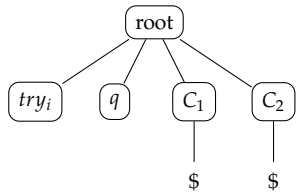

Figure 23: DTD

The *Init* is a Boolean combination of tree patterns describing a tree corresponding to the state q_0 and the initial values for the two counters in M. Recall that in the definition of positive bounded TPRS, it is *not* required that *Init* is positive.

Take the transition $\delta_i = (q, = 0, \neq 0, q', +1, -1)$ as an example, we illustrate how to simulate the transitions of M by TPRS rules in $\mathcal{R}$.

The transition $\delta_i = (q, = 0, \neq 0, q', +1, -1)$ is simulated by two rules, Try_i verifying the conditions of δ_i, and $Trans_i$ enforcing the actions of δ_i.

Try_i is illustrated in Figure 24, where F is the tree with a single node labeled by try_i, and the node labeled by q is renamed as q_{try} in order to avoid repeated application of Try_i. Note that the zero test for the first counter is forgotten in the definition of Try_i.

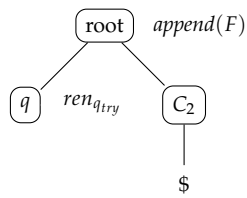

Figure 24: Rule Try_i

$Trans_i$ is illustrated in Figure 25, the node try_i is deleted, q_{try} is renamed as q', C_1 is increased (add a tree F, a single node labeled by \$), and C_2 is decreased (remove one \$).

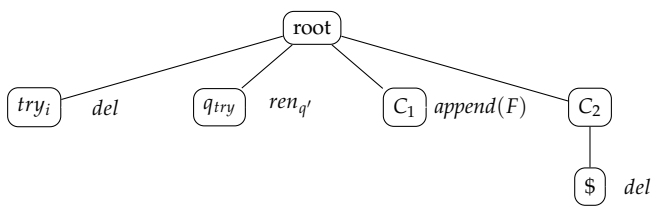

Figure 25: Rule $Trans_i$

Note that the DTPRS defined above does not use data, thus in fact a TPRS.

For each reachable configuration of M, before halting, there is a unique transition enabled. However, for the TPRS $(DTD, \mathcal{R}, Init)$ defined above, there may be several rules enabled at the same time. For instance, if there are one rule $\delta_i = (q, = 0, \neq 0, \dots)$ and another rule $\delta_j = (q, \neq 0, \neq 0, \dots)$, and there is at least one child (\$) for both node C_1 and node C_2 in the current data tree, then both the rule Try_i and Try_j are enabled according to

the definition. However, only one of the transitions (Try_j in the example) corresponds to the valid transition in M, all the others (Try_i in the example) are cheating.

The Tree-LTL formula $F\varphi$ satisfies that φ is a disjunction of tree patterns P_{q_h} ($q_h \in H$) and P_{cheat_i}, where P_{q_h} is a tree pattern specifying that the state q_h is reached, and P_{cheat_i} (see Figure 26) is a tree pattern specifying that the TPRS is cheating, in the sense that the rule Try_i has been applied (so the transition $\delta_i = (q, = 0, \neq 0, q', +1, -1)$ of M is simulated), but the node C_1 has at least one child (the counter C_1 has nonzero value). P_{cheat_j} can be defined similarly for the other transitions δ_j in M.

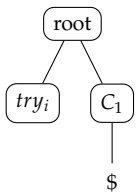

Figure 26: Pattern P_{cheat_i}

A run in TPRS $(DTD, \mathcal{R}, Init)$ is either a cheating run or a valid run corresponding to the unique run of M.

Recall that the two-counter machine M is assumed to either halt, i.e. reach a state in H, or always have a unique transition enabled, thus have an unique infinite run. Since the TPRS $(DTD, \mathcal{R}, Init)$ simulates M by ignoring zero tests, it follows that the TPRS satisfies $F\varphi$ iff each run of the TPRS is either cheating or a valid halting run (i.e. reaching in a state $q_h \in H$). Consequently, the TPRS $(DTD, \mathcal{R}, Init)$ satisfies $F\varphi$ iff M halts. ∎

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

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

where P is a DTP.

PROOF.

Let $(\mathcal{R}, \Delta)$ be a given positive-bounded DTPRS.

It has been shown in Section 4 that $(\mathcal{T}_{B,K}, \longrightarrow, \preceq)$ is a well-structured transition system.

The proof goes as follows:

- A fragment of modal μ-calculus with DTPs as atomic propositions, called existential positive modal tree μ-calculus (denoted as Tree-$L_\mu^{\exists,+}$), is first defined.
- Then the existential positive Tree-LTL is shown subsumed by this fragment.
- Finally, for each Tree-$L_\mu^{\exists,+}$ sentence φ, the set of data trees satisfying φ is shown upward-closed and a finite basis can be effectively computed.

The existential positive modal tree μ-calculus (Tree-$L_\mu^{\exists,+}$) formulas are defined by the following rules,

$$\varphi := true \mid false \mid P \mid Z \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \diamond\varphi_1 \mid \mu Z.\varphi_1(Z),$$

where P is a data tree pattern and $\varphi_1(Z)$ is a Tree-$L_\mu^{\exists,+}$ formula containing the free variable Z.

A Tree-$L_\mu^{\exists,+}$ sentence is a Tree-$L_\mu^{\exists,+}$ formula containing no free variables. Let $\mathcal{T}(\varphi)$ be the set of data trees in $\mathcal{T}_{B,K}$ satisfying φ.

Similar to the translation of LTL with existential semantics to existential positive modal μ-calculus (c.f. [8]), the following result can be shown.

LEMMA 16. *From each positive Tree-LTL formula with existential semantics, an equivalent Tree-$L_\mu^{\exists,+}$ sentence can be constructed effectively.*

LEMMA 17. *Let $\varphi(Y_1, \cdots, Y_n)$ be a Tree-$L_\mu^{\exists,+}$ formula containing the set of free variables $Y_1, \ldots, Y_n$, and $\mathcal{T}_1, \ldots, \mathcal{T}_n \subseteq \mathcal{T}_{B,K}$ be a sequence of upward-closed sets of data trees with finite basis resp. $F_1, \ldots, F_n$. Then $\mathcal{T}(\varphi(\mathcal{T}_1, \ldots, \mathcal{T}_n))$ is upward-closed and a finite basis of it can be computed effectively from $F_1, \ldots, F_n$.*

PROOF. The proof is by a induction of the structure of Tree-$L_\mu^{\exists,+}$ formulas.

It is easy to show this for atomic formulas.

The cases of disjunctions and conjunctions follow from the fact that upward-closed sets are closed under union and intersection.

Suppose $\varphi = \diamond\varphi_1$. Then $\mathcal{T}(\varphi) = \mathit{Pred}(\mathcal{T}(\varphi_1))$. By induction hypothesis, $\mathcal{T}(\varphi_1)$ is upward-closed and a finite basis of it has been effectively computed. The upward-closeness of $\mathcal{T}(\varphi)$ and the effectiveness of its finite basis then follow from the fact that $(\mathcal{T}_{B,K}, \longrightarrow, \preceq)$ is a well-structured transition system.

Now consider the situation that $\varphi = \mu Z.\varphi_1(Z, Y_1, \ldots, Y_n)$.

First set $Z_0 = \emptyset$, then compute a finite basis of $Z_1 = \varphi_1(Z_0, \mathcal{T}_1, \ldots, \mathcal{T}_n)$, which can be done according to the induction hypothesis.

Similarly, compute a finite basis of $Z_2 = \varphi(Z_1, \mathcal{T}_1, \ldots, \mathcal{T}_n)$, and so on.

Because $Z_0 \subseteq Z_1 \subseteq Z_2 \subseteq \cdots$, and each Z_i is upward-closed, it follows that there is i such that $Z_i = Z_{i+1}$, since $\preceq$ is a wqo over $\mathcal{T}_{B,K}$ (c.f. Lemma 2.4 in [14]). Such a Z_i is exactly $\mathcal{T}(\mu Z.\varphi(Z, \mathcal{T}_1, \ldots, \mathcal{T}_n))$. ∎

From the above two lemmas, it follows that given a positive Tree-LTL formula φ (with existential semantics), the set of data trees in $\mathcal{T}_{B,K}$ satisfying φ is upward-closed and a finite basis of it can be effectively constructed.

The decidability of verification of Tree-LTL formula φ over the postive-bounded DTPRS $(\mathcal{R}, \Delta, \mathit{Init})$ then follows from the decidability of the satisfiability of Boolean combinations of DTPs over (unordered) data trees of bounded depth with respect to a DTD ([9]). ∎

G Proofs in Section 6

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

We first show how to decide in NexpTime the existence of data trees T, T' and DTP rule R such that $T \models \Delta$, $T \xrightarrow{R} T'$, but $T' \not\models \Delta$.

Recall that the DTD τ is defined by rules $a \to \psi$, where ψ is a Boolean combination of the inequalities of the form $|b| \geq k$. Let B be maximum depth of trees satisfying τ and $\max(\Delta)$ be the maximal integer k occurring in the inequalities $b \geq k$ of τ.

At first, it is not hard to show the following result.

LEMMA 18. *[Completion wrt. τ] Let T, T_1 be data trees such that $T \models \Delta$, $T_1 \preceq T$ and $T_1 \models$ inv. Then there exists T_1' with $T_1 \preceq T_1' \preceq T$, $|T_1'| = O(|T_1|(\max(\Delta)|\Sigma|)^{B+1})$ and $T_1' \models \Delta$.*

LEMMA 19. *Assume that there exist a data tree $T \models \Delta$, a DTP rule $R = \langle L, G, \mathcal{Q}, \mathcal{F}, \chi \rangle \in \mathcal{R}$ and a data tree T' such that $T \xrightarrow{R} T'$ and $T' \not\models \tau$. Then there exists such a tree T of size at most*

$$B\left(|\text{inv}| + |L| + |G| + |L|(\max(\Delta)+1)|\Sigma| \max_{F \in \mathcal{F}} |F| \max_{Q \in \mathcal{Q}} |Q|\right)(\max(\Delta)|\Sigma|)^{B+1}.$$

PROOF. We show how to get $T_1 \preceq T$ and $T_1' \preceq T'$ with $T_1 \xrightarrow{R} T_1'$, $T_1' \not\models \tau$ and $T_1 \models \Delta$ has size bounded as in the statement of the lemma. The tree T_1 is defined in such a way that it satisfies the same DTPs from *inv* and G, as T does. Let us denote this set of DTPs by $\mathcal{P}$. For each $P \in \mathcal{P}$, the satisfaction of P in T is witnessed by a matching φ_P from P to T.

- Witness $T \models inv$ and $T \models G$: The set of nodes in the image of φ_P and their ancestors in T, size $O(B(|inv| + |G|))$.
- Witness of the (injective) matching φ_L from L to T: The set of nodes in the image of φ_L and their ancestors in T, size $O(B|L|)$.
- Witness of the evaluation of a query on T: suppose $Q = body \rightsquigarrow head$ is a DTPQ labeling some node in a forest of $\mathcal{F}$. Note that at most $(\max(\Delta)+1)|\Sigma|$ copies of *head* are needed to witness $T' \not\models \tau$.
  There are at most $|L| \max_{F \in \mathcal{F}} |F|$ possible queries that might be evaluated when applying R to T. Each evaluation requires at most $(\max(\Delta)+1)|\Sigma|$ copies of *head*-trees. Thus the overall size of witnesses in T is

$$O(|L|(\max(\Delta)+1)|\Sigma|(\max_{F \in \mathcal{F}} |F|)(B \max_{Q \in \mathcal{Q}} |Q|)).$$

The sum of the sizes of all those witnesses is at most

$$B\left(|inv| + |L| + |G| + |L|(\max(\Delta)+1)|\Sigma| \max_{F \in \mathcal{F}} |F| \max_{Q \in \mathcal{Q}} |Q|\right).$$

Finally, the witness-tree should be completed with respect to T and τ, in order to satisfy the DTD, which gives by Lemma 18 the claimed bound. ∎

LEMMA 20. *Assume that there exists a data tree $T \models \Delta$, a DTP rule $R = \langle L, G, \mathcal{Q}, \mathcal{F}, \chi \rangle \in \mathcal{R}$ and $T \xrightarrow{R} T'$ such that $T' \not\models$ inv. Then there exists such a tree T of size at most*

$$B\left(2|\text{inv}| + |L| + |G| + |\text{inv}| \cdot |L| \max_{F \in \mathcal{F}} |F| \max_{Q \in \mathcal{Q}} |Q|\right)(\max(\Delta)|\Sigma|)^{B+1}.$$

PROOF. The proof is similar to Lemma 19. We look for some $T_1 \preceq T$, and $T_1' \preceq T'$, such that $T_1 \models \Delta$, but $T_1' \not\models \mathit{inv}$. The only difference is that we need to witness in T' some DTPs for the negation of *inv*. This yields the claimed upper bound. ∎

We show the lower bound now. From [9], we know that depth-bounded satisfiability of Boolean combinations of DTPs with respect to a DTD (with depth as part of the input) is NexpTime-hard.

Let φ be a Boolean combination of DTPs and τ a DTD. The DTPRS $(\mathcal{R}, \Delta)$ is defined as follows: Δ consists of τ and the static data invariant φ. There is a single rule R which relabels the root of the data tree to a label r such that r does not occur in τ. Then φ is satisfiable wrt. τ iff there are T, T' such that $T \models \Delta$, $T \xrightarrow{R} T'$ and $T' \not\models \Delta$.

H Proofs in Section 7

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

The proof is similar to the proof that checking Tree-LTL properties over recursion-free GAXML systems is Co-2NexpTime-complete [5].

First consider upper bound.

The upper bound for model-checking of recursion-free GAXML in [5] was shown by establishing a double exponential small model property.

By a similar argument, an exponential small model property can be obtained for the bounded model-checking over DTPRS. The exponentiation, instead of double-exponentiation, follows from the fact that in bounded model-checking, the length bound N is given by unary encoding, thus of polynomial size, while for the recursion-free GAXML, the length of runs is exponential over the size of the GAXML system.

The lower bound is adapted from the Co-2NexpTime lower bound proof for model-checking of recursion-free GAXML. We only recall the rough idea here.

The main ingredient of the proof is to create/check lists of length 2^n. This is done using data values, similarly as in the proof of Theorem 2. A "list" of length k corresponds to a tree of depth 2, where each node at depth one has 2 children, with distinct data values d_i, d_{i+1}. If each data value d_i occurs twice (except for d_1 and d_{k+1}, which occur only once) we get a linear order, i.e. a list. Using n queries we can compute n steps of transitive closure and thus verify that $k = 2^n$. Obviously, this suffices for encoding a $(2^n \times 2^n)$ tableau representing a computation of a 2^n-time bounded TM. Details are fairly easy to complete. ∎