Presburger Büchi Tree Automata with Applications to Logics with Expressive Counting

Bartosz Bednarczyk[1,2] and Oskar Fiuk[2]

[1] Computational Logic Group, Technische Universität Dresden, Germany
[2] Institute of Computer Science, University of Wrocław, Poland
bartosz.bednarczyk@cs.uni.wroc.pl, 307023@uwr.edu.pl

Abstract. We introduce two versions of Presburger Automata with the Büchi acceptance condition, working over infinite, finite-branching trees. These automata, in addition to the classical ones, allow nodes for checking linear inequalities over labels of their children. We establish tight NP and ExpTime bounds on the complexity of the non-emptiness problem for the presented machines. We demonstrate the usefulness of our automata models by polynomially encoding the two-variable guarded fragment extended with Presburger constraints, improving the existing triply-exponential upper bound to a single exponential.

1 Introduction

Description Logics (DLs) [3] are a robust family of first-order logic (FO) fragments designed for knowledge representation and reasoning. They serve as the core of the Web Ontology Languages (OWL) and are successfully used in the bio-medical domain, *e.g.* in Snomed CT [17]. Unfortunately, the expressiveness of DLs is quite limited: as description logics are fragments of FO, they are not able to express even simple statistical properties. Such information is crucial to reason about real-life scenarios as witnessed by the following example:

Example 1. Consider a knowledge base of Citizens voting for their new mayor. During the election, they must decide for each candidate whether they reject him or not. Moreover, each citizen can give at most two accepting votes. This situation can be modelled, in a DL-like language as follows with:

$$\text{Citizen} \sqsubseteq (\leq 2.\text{votedForAcc}).\top, \quad \text{Winner} \sqsubseteq (> 50\%)\text{votedForAcc}^{-}.\text{Citizen},$$

so the elected winner received more than 50% of accepting votes from Citizens.

The above-mentioned properties can be formalised in extensions of the standard description logic $\mathcal{ALC}$ with Presburger arithmetic, as was done *e.g.* in [1,15,19]. However, to model Example 1 one additionally requires the use of inverses of roles, making the logics from the cited papers not directly applicable in our setting. The decidability status of $\mathcal{ALC}$ with inverses and arithmetics was established only a few months ago in [5], but with a non-optimal 3NExpTime complexity. The aim of this short paper is to improve this complexity to a single exponential.

Our results. We follow a well-established automata-theoretic approach to decidability of description logics, as treated in several papers, see e.g. [2,11,12]. We present two models of tree automata, working over infinite finite-branching trees, with Büchi accepting conditions. The first of them, called SPBTA, assumes the powerset alphabet and has an ExpTime-complete non-emptiness problem. That makes SPBTA an attractive automata model for encoding modal and description logics with Presburger constraints. The other one, called PBTA, works over the usual alphabet and turns out to be NP-complete, complementing already existing results on finite tree automata with expressive counting from [22,23].
As an application of our results, we provide a translation from an extension of $\mathcal{ALCI}$ with expressive cardinality constraints, namely the two-variable guarded fragment with Presburger arithmetics [5], to our automata. This yields ExpTime-completeness of the mentioned logic.

Related automata models. We briefly discuss what distinguishes our automata models in comparison to the existing models. First of all, our automata work over unordered and unranked trees, hence all different automata models do not apply to our case. Second, our trees are infinite, hence the framework of [22,23] cannot simply be transferred here. Third, several tree automata models with arithmetics were proposed quite recently [9,7,6] but in their case the complexity of the non emptiness problem is either higher ExpTime or is nondeterministic, disallowing a reasonable translation of description logics into it. Finally, standard automata models used for description logics, e.g. the ones from [8] are not applicable to our case as they cannot impose Presburger constraints on successors.

2 Preliminaries

We assume that the reader is familiar with basics on formal languages, computability [24], and tree automata [14].

Let $\underline{n}$ denote the set $\{k \in \mathbb{N} \mid k < n\}$. By a *tree*, we mean a finite-branching tree, in which every branch is infinite. More formally, $\mathfrak{t}$ is a prefix-closed subset of $N\omega$, such that for every $x \in \mathfrak{t}$ we have $x0 \in \mathfrak{t}$ and there is $n \in \mathbb{N}$ so that no xn belongs to $\mathfrak{t}$. The elements of $\mathfrak{t}$ are called *nodes*. W.l.o.g. we assume that trees are *children-closed*, *i.e.* if $x \in \mathfrak{t}$ and $n \in \mathbb{N}$ then $xn \in \mathfrak{t}$ implies xk for all $k \in \underline{n}$. Given $x \in \mathfrak{t}$ we define the subtree of x (resp. children of x), denoted $\mathsf{Subt}(x)$ (resp. $\mathsf{Chld}(x)$), as the set $\{z \in \mathfrak{t} \mid z = xy\}$ (resp. $\{xn \in \mathfrak{t} \mid n \in \mathbb{N}\}$). The *degree* $\mathsf{d}(x)$ (resp. *height* $\mathsf{h}(x)$) of x is $|\mathsf{Chld}(x)|$ (resp. $|x|$). A *branch* of $\mathfrak{t}$ is any infinite sequence $v_0, v_1, v_2, \ldots$ of nodes of $\mathfrak{t}$, such that $v_0 = \varepsilon$ and for all i the node v_{i+1} is a child of v_i. A Σ-labelled tree is a pair $(\ell, \mathfrak{t})$, composed of a tree $\mathfrak{t}$ and a mapping $\ell : \mathfrak{t} \to \Sigma$. Sometimes we will also think about *finite* trees, defined in a natural way, with all the definitions adopted to them as the reader may expect.

By a *linear constraint* we mean an expression of the form

$$C + \Sigma_{i=0}^{n} a_i \cdot x_i \ \bowtie \ D + \Sigma_{i=0}^{m} b_i \cdot y_i,$$

where x_i, y_i are variables from some countably-infinite set $\mathcal{V}$, $\bowtie$ is one of $\leq, <, \equiv_k$ (the last one denotes equality modulo[3] k) and $a_i, b_i, k > 0, C, D$ are integers

[3] Constraints with $\equiv_k$ can be eliminated [4].

encoded in *binary*. We define *systems of constraints* and their *solutions over* $\mathbb{N}$ in an expected way. Given a solution $S : \mathcal{V} \to \mathbb{N}$ to a system E, the *support* $\mathsf{supp}(S)$ of S is defined as the set $\{v \in \mathcal{V} \mid S(v) \neq 0\}$.

Testing *solvability* (over $\mathbb{N}$) of systems of constraints is NP-complete [10]. Moreover, the following less-known lemma provides the existence of a "sparse and small" solution; consult: [16, Theorem 1], [1, Lemma 3] and [21].

Lemma 2. *Let $\mathcal{I}$ be a system of E linear constraints with integer coefficients absolutely bounded by C. Assume a solution S to $\mathcal{I}$ (over $\mathbb{N}$). Then there is a solution S' to $\mathcal{I}$ (over $\mathbb{N}$) with $|\mathsf{supp}(S')| \leq 2E \cdot \log(4EC)$ and $\mathsf{supp}(S') \subseteq \mathsf{supp}(S)$. Moreover, the maximal absolute value assigned to variables by S' is bounded by $V(E \cdot C)^{2E+1}$, where V is the total number of variables.*

3 Suitable tree automata models

Henceforth, we consider two tree automata models, starting from the more succinct and expressive one:

Definition 3. *A Succinct Presburger Büchi Tree Automata (SPBTA) $\mathcal{A}$ is a tuple $\mathcal{A} = (\Sigma, Q, F, Q_0, I)$, where all of its components are finite, Σ is an* alphabet*, Q is a set of* states*, $F \subseteq Q$ is a set of* final states*, $Q_0 \subseteq Q$ is an* initial configuration *and I is a finite set of* instructions *of the form:*

- $\pm q \to \pm q'$, $q \wedge q' \to q''$ *for* $q, q', q'' \in Q$,
- $q \to \pm a, \pm a \to q$ *for* $q \in Q$ *and* $a \in \Sigma$, *and*
- $q \to \mathcal{C}, \mathcal{C} \to q$, *where $\mathcal{C}$ is a linear constraint over $\mathcal{V} = \{x_S \mid S \subseteq Q\}$. For succinctness, only the mentioned variables from $\mathcal{V}$ are required to be explicitly written in $\mathcal{C}$.*

Our automata work over 2^Σ-labelled trees and labels each node with a subset of Q. Such a labelling is required to be compatible with Q_0, *i.e.* ε is labelled with Q_0. Moreover, it should be compatible with I, *i.e.* that the presence of a state in a node v implies the absence/presence of a state/letter in v, and that linear constraints over the labellings of v's children, where the value of the variable x_S is the total number of v's children labelled by some superset of S. Formally:

Definition 4. *A* run *of an SPBTA $\mathcal{A} = (\Sigma, Q, F, Q_0, I)$ over a 2^Σ-labelled tree $(\ell_\mathfrak{t}, \mathfrak{t})$ is a 2^Q-labelled tree $(\ell^\mathcal{A}_\mathfrak{t}, \mathfrak{t})$ s.t. $\ell^\mathcal{A}_\mathfrak{t}(\varepsilon) = Q_0$, and for all $v \in \mathfrak{t}$, $q \in \ell^\mathcal{A}_\mathfrak{t}(v)$:*

- *if $(q \to q') \in I$ (resp. $(q \to -q') \in I$) with $q' \in Q$, then $q' \in \ell^\mathcal{A}_\mathfrak{t}(v)$ (resp. $q' \notin \ell^\mathcal{A}_\mathfrak{t}(v)$), and if $(q \wedge q') \to q'' \in I$ with $q', q'' \in Q$ then $q, q' \in \ell^\mathcal{A}_\mathfrak{t}(v)$ implies $q'' \in \ell^\mathcal{A}_\mathfrak{t}(v)$,*
- *if $(q \to \pm a) \in I$ (resp. $(q \to -a) \in I$) with $a \in \Sigma$, then $a \in \ell_\mathfrak{t}(v)$ (resp. $a \notin \ell_\mathfrak{t}(v)$),*
- *if $(q \to \mathcal{C}) \in I$ with $\mathcal{C}$ being a linear constraint, then $x_S \mapsto |\{u \in \mathsf{Chld}(v) : S \subseteq \ell^\mathcal{A}_\mathfrak{t}(u)\}|$ is a solution to $\mathcal{C}$.*

Omitted cases are symmetric. A run is successful*, if every branch of $\mathfrak{t}$ has inf. many elements v fulfilling the condition: "there is an accepting state $q_f \in F$*

such that $q_f \in \ell_{\mathcal{A}}(v)$". An SPBTA $\mathcal{A}$ accepts *a 2^{Σ}-labelled tree $(\ell_{\mathfrak{t}}, \mathfrak{t})$ if there is a successful run over $(\ell_{\mathfrak{t}}, \mathfrak{t})$. A language of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of all 2^{Σ}-labelled trees accepted by $\mathcal{A}$.*

The succinct representation of the alphabet and constraints in SPBTA will turn out to be especially useful for encoding models of modal and description logics with Presburger constraints (due to their tree-like model property). This is the biggest advantage of SPBTA in contrast to other automata models mentioned in the introduction. In order to translate logics into automata in the usual setting, the user must first provide a bound on the number of children that a node may have[4]. This essentially forces the user to prove some kind of a "small-branching tree model lemma" before you proceed with tree-encoding. Our automata model will do it automatically Lemma 16, so you can encode tree-models without the need of finding an appropriate bound (which sometimes can be difficult).

In what follows we present yet another model of tree automata, which is simpler due to the use of a standard alphabet. Within a run, nodes are labelled with exactly one state, and therefore, many of the presented instructions can be removed. While PBTA does not have any direct applications to logic, we find the drop of complexity interesting.

Definition 5. *A Presburger Büchi Tree Automata (PBTA) $\mathcal{A}$ is a tuple $\mathcal{A} = (\Sigma, Q, F, q_0, I)$, where all its components are finite, Σ is an* alphabet*, Q is a set of* states*, $F \subseteq Q$ is a set of* final states*, $q_0 \in Q$ is an* initial configuration *and I is a finite set of* instructions *of the form:*

- $(q, a) \to \mathcal{C}$*, where $q \in Q$, $a \in \Sigma$ and $\mathcal{C}$ is a linear constraint with variables $\mathcal{V} = \{x_q \mid q \in Q\}$.*

We define runs of PBTAs analogously to the case of SPBTAs.

Definition 6. *A* run *of a PBTA $\mathcal{A} = (\Sigma, Q, F, q_0, I)$ over a Σ-labelled tree $(\ell_{\mathfrak{t}}, \mathfrak{t})$ is a Q-labelled tree $(\ell^{\mathcal{A}}_{\mathfrak{t}}, \mathfrak{t})$ such that $\ell^{\mathcal{A}}_{\mathfrak{t}}(\varepsilon) = q_0$, and for all $v \in \mathfrak{t}$ with $q_v{=}\ell^{\mathcal{A}}_{\mathfrak{t}}(v)$ and $a_v{=}\ell_{\mathfrak{t}}(v)$:*

- *if $((q_v, a_v) \to \mathcal{C}) \in I$, then $x_q \mapsto |\{u \in \mathsf{Chld}(v) \mid q{=}\ell^{\mathcal{A}}_{\mathfrak{t}}(u)\}|$ is a solution to $\mathcal{C}$.*

A run is successful*, if every branch of $\mathfrak{t}$ has infinitely many elements v fulfilling the condition: "there is an accepting state $q_f \in F$ such that $q_f = \ell_{\mathcal{A}}(v)$".*

The notion of accepted trees and the *language of* PBTA are defined analogously to the case of SPBTA. Note that the languages accepted by (S)PBTA are no longer regular (in the sense of MSO-definability). Consider:

Example 7. A language $\{\mathfrak{t} \mid \forall v \in \mathfrak{t} \quad \mathsf{d}(v) \equiv_2 0\}$ is clearly not regular, but can be recognized by the PBTA $\mathcal{A} = (\{a\}, \{q\}, \{q\}, q, \{(q, a) \to x_q \equiv_2 0\})$.

[4] Of course we may always restrict ourselves to binary trees by employing standard tricks, but then the verification of Presburger constraints becomes a challenge.

The *non-emptiness* problem asks whether the language of a given (S)PBTA is non-empty. An immediate reduction from the solvability (over $\mathbb{N}$) of systems of linear equations yields NP-hardness for testing non-emptiness of PBTA.

Lemma 8. *The non-emptiness for PBTAs is* NP*-hard.*

Proof. Take a system of linear equations $\mathcal{I}$ and let $\Sigma = \{a_0\}, Q = F = \{q_v \mid v \in \mathcal{V}(\mathcal{I})\}$, $I = \{(q_v, a_0) \to \mathcal{C} \mid v \in \mathcal{V}(\mathcal{I}), \mathcal{C} \in \mathcal{I}\}$. Choose v_0 to be an arbitrary variable in $\mathcal{V}(\mathcal{I})$. Then the language recognized by automaton $\mathcal{A} = (\Sigma, Q, F, q_{v_0}, I)$ is non-empty iff $\mathcal{I}$ has a solution.

In the sequel, we provide a matching upper bound.

4 Non-emptiness problem for PBTA is in NP

Henceforth, we fix a PBTA $\mathcal{A} = (\Sigma, Q, F, q_0, I)$. By its size, denoted with $|\mathcal{A}|$, we mean the size of some reasonable succinct representation of $\mathcal{A}$, where the numbers in constraints are encoded in *binary*. We start with auxiliary definitions.

Definition 9. *Let $(\ell_{\mathcal{A}}, \mathfrak{t})$ be an accepting run of $\mathcal{A}$ on a tree $(\ell_{\mathfrak{t}}, \mathfrak{t})$. Then a state $q \in Q$ has* an occurrence in a node $v \in \mathfrak{t}$ *iff $\ell_{\mathcal{A}}(v) = q$. If there is $v \in \mathfrak{t}$, such that q occurs in v, then q has* an occurrence in an automaton run $(\ell_{\mathcal{A}}, \mathfrak{t})$. *An occurrence of $q \in Q$ in $v \in \mathfrak{t}$ is* safe *iff either $q \in F$ or on each branch $v_1, v_2, v_3, \ldots$ starting in v there exists an occurrence of some state from F before the next occurrence of q (if any),* i.e. $\inf\{i \mid \ell_{\mathcal{A}}(v_i) \in F\} < \inf\{i \mid \ell_{\mathcal{A}}(v_i) = q\}$. *A* safe subtree *of a safe occurrence $q \in Q$ in $v \in \mathfrak{t}$, denoted as $\mathsf{SSubt}(q, v)$, is a pair $(v, \mathfrak{t}')$ such that $\mathfrak{t}'$ is a singleton $\{\varepsilon\}$ whenever $q \in F$ or otherwise is a finite tree $\{x \mid vx \in \mathsf{Subt}(v), \forall st = x, t \neq \varepsilon,\ \ell_{\mathcal{A}}(vs) \notin F\}$,* i.e. *accepting states occur only in leaves.*

Lemma 10. *If $q \in Q$ has an occurrence in an accepting run $(\ell_{\mathcal{A}}, \mathfrak{t})$ of $\mathcal{A}$, then it has a safe one.*

Proof. Suppose that there is $q \in \ell_{\mathcal{A}}(\mathfrak{t})$, such that no occurrence is safe. Clearly, $q \notin F$. Then one can construct an infinite branch without any state from F, yielding a contradiction with the Büchi acceptance condition. Employ the following process. Start with any occurrence of q in some $v \in \mathfrak{t}$. Then by assumption it is not safe. Therefore, it contains a branch that reaches another occurrence of q in some other v' before reaching any state from F. Repeat from v'.

To get an NP-upper bound, we construct a *regular tree*, generated by a pair of functions and a state, that will serve as a witness for the non-emptiness test of a given automaton.

Definition 11. *For functions $\alpha : Q \to \Sigma$, $\gamma : Q \to Q^+$, and an element $q_0 \in Q$, define inductively a family of sets:*

- $T_0 = \{(\varepsilon, q_0)\}$, *and* $T_{n+1} = \{(xk, \gamma(q)_k) \mid (x, q) \in T_n, k = 0, \ldots, |\gamma(q)| - 1\}$.

Let $\mathcal{T} = \bigcup_{n \geq 0} T_n$. Then

- $\mathfrak{t} = \{x \mid (x,q) \in \mathcal{T}\}$ *is a tree,*
- ℓ_Q *is taken as any* Q*-labelling of* $\mathfrak{t}$ *such that* $(x, \ell_Q(x)) \in \mathcal{T}$*,*
- ℓ_Σ *is taken as any* Σ*-labelling of* $\mathfrak{t}$ *such that* $\ell_\Sigma(x) = \alpha(\ell_Q(x))$*.*

The role of α is to name elements with letters and the role of γ is to create fresh state-labelled children of a given node. Note that we are working with unordered trees, thus w.l.o.g. we assume that γ is "sorted". Thus γ can represented as a function $\Gamma : Q \to \mathbb{N}^Q$ defined as:

$$\gamma(q) = \underbrace{q_1 \dots q_1}_{\Gamma(q)(q_1) \text{ times}} \quad \cdots \quad \underbrace{q_{|Q|} \cdots q_{|Q|}}_{\Gamma(q)(q_{|Q|}) \text{ times}} .$$

The role of Γ is to encode potential solutions to a system of inequalities from the I component of $\mathcal{A}$. Finally, we say that $(\ell_Q, \mathfrak{t})$ (resp. $(\ell_\Sigma, \mathfrak{t})$) is Q*-generated* (resp. Σ*-generated*) tree by a triple (q_0, α, Γ).

Lemma 12. *Let* $u = (q_0, \alpha, \Gamma)$ *generate a tree* $\mathfrak{t}$ *accepted by* $\mathcal{A}$*. Then there is a triple* $u' = (q_0, \alpha, \Gamma')$ *generating another tree* $\mathfrak{t}'$ *that is still accepted by* $\mathcal{A}$*, and which has small values in* Γ*,* i.e. $\log \max\{\Gamma(q)(q') \mid q, q' \in Q^2\} = O(poly(|\mathcal{A}|))$.

Proof. Since u generates $\mathfrak{t}$ accepted by $\mathcal{A}$, the values of $\Gamma(q)$ (after a renaming) are solutions to constraints $q \to \mathcal{C}$ given by $\mathcal{A}$. By Lemma 2 we get smaller solutions that can be encoded as Γ'. Then the tree $\mathfrak{t}'$ generated by (q_0, α, Γ') is clearly accepted by $\mathcal{A}$. Indeed, suppose that there is a branch with only finitely many accepting states. Then, by the inclusion of supports for the solutions and due to the way the tree $\mathfrak{t}'$ is generated, there will be a branch in the original tree $\mathfrak{t}$ with the same Q-labelling, yielding a contradiction.

The next crucial lemma shows how to generate a small regular tree, serving as a witness for our non-emptiness test.

Lemma 13. *If* $\mathcal{L}(\mathcal{A})$ *is nonempty, then there is a tree* $\mathfrak{t}$ *and a triple* $u = (q_0, \alpha, \Gamma)$ *satisfying: (1)* $(\ell_\Sigma, \mathfrak{t}) \in \mathcal{L}(\mathcal{A})$*, and is* Σ*-generated by* u*, (2)* $(\ell_Q, \mathfrak{t})$ *is an accepting run of* $\mathcal{A}$ *on the tree* $(\ell_\Sigma, \mathfrak{t})$*, and is* Q*-generated by* u*, and (3) the triple* u *has size polynomial in* $|\mathcal{A}|$*.*

Proof. Fix a tree $(\ell_{\mathfrak{t}'}, \mathfrak{t}') \in \mathcal{L}(\mathcal{A})$ and a corresponding accepting run $(\ell^{\mathcal{A}}_{\mathfrak{t}'}, \mathfrak{t}')$ of $\mathcal{A}$. Let $Q_0 = \{q_1, \dots, q_k\}$ be the set of states appearing in $\ell^{\mathcal{A}}_{\mathfrak{t}'}(\mathfrak{t}')$. For each $q_i \in Q_0$ fix a safe occurrence in some $v_i \in \mathfrak{t}'$ and let $(u_i, \mathfrak{t}_i) = \mathsf{SSubt}(q_i, v_i)$. We also fix some post-order enumeration of nodes from $\mathfrak{t}_i$ and let $\mathtt{PO}(\mathfrak{t}_i, v)$ denotes the position of $v \in \mathfrak{t}_i$ in such order.

A *solution* in a node v is a function $S_v : Q \to \mathbb{N}$ such that

$$S_v(q) = |\{u \in \mathsf{Chld}(v) : \ell^{\mathcal{A}}_{\mathfrak{t}'}(u) = q\}|.$$

We define a *witness* injection $j : Q_0 \to \mathbb{N}^2 \times \mathbb{N}^*$, that will be used to define α and Γ components of u, as follows:

$$j(q_i) = \min\{(r, s, x) \mid x \in \mathfrak{t}_r, q_i = \ell^{\mathcal{A}}_{\mathfrak{t}'}(u_r x), \mathtt{PO}(\mathfrak{t}_r, x) = s\},$$

where the minimum is taken with respect to the lexicographic order. The intuition behind the function j is that it assigns, for each state, the deepest leftmost occurrence of this state in the safe subtree with the lowest possible index. Therefore, if the node is not a leaf, any outgoing edge leads to a child with a strictly smaller value of j. Since all leaves are labelled with accepting states, any branch in the resulting tree that visits the same node twice must pass through the accepting state.

Suppose that $j(q_i) = (r_i, s_i, x_i)$. Then we simply put $\alpha(q_i) = \ell_{\mathfrak{t}'}(u_{r_i} x_i)$ and $\Gamma(q_i) = S_{u_{r_i} x_i}$ for each $q_i \in Q_0$, and let $\alpha(q) = \bot$ and $\Gamma(q) = \bot$ for each $q \in Q \setminus Q_0$. One can show (see Appendix A) that $(\ell_Q, \mathfrak{t})$ is indeed an accepting run of $\mathcal{A}$ on the tree $(\ell_\Sigma, \mathfrak{t})$ with respect to the given definition of (q_0, α, Γ). Thus, by applying Lemma 12 we conclude that u has a polynomial size description.

Lemma 14. *Given $\mathcal{A}$ and a tuple $u = (q_0, \alpha, \Gamma)$, it is in* PTIME *to test if u generates a tree from $\mathcal{L}(\mathcal{A})$.*

Proof. We can clearly verify all constraints from the I component of $\mathcal{A}$, since Γ encodes all the required solutions. For the Büchi acceptance condition build a directed graph on the vertices Q and edges $E{=}\{(q, q') \in Q^2 \mid q \notin F, \Gamma(q)(q') > 0\}$. Then if the graph (Q, E) is acyclic, output `Yes`, and `No` otherwise.

Thus as a corollary of Lemmas 8, 13, 14 we conclude:

Theorem 15. *The non-emptiness problem for Presburger Büchi Tree Automata is* NP*-complete.*

5 Non-emptiness for SPBTA

Fix an SPBTA $\mathcal{A} = (\Sigma, Q, F, Q_0, I)$. W.l.o.g. we can assume that Σ is empty; if not, then we simply introduce fresh states q_a per each letter $a \in \Sigma$ and replace all occurrences of a in I by q_a. Take E be the number of constraints in I and let C be the minimal absolute bound on the coefficients from I.

We start with a "small accepted tree lemma":

Lemma 16. *If there is an accepting run $(\ell^{\mathcal{A}}_{\mathfrak{t}}, \mathfrak{t})$ of $\mathcal{A}$ on some tree $(\ell_{\mathfrak{t}}, \mathfrak{t})$, then there is a tree $(\ell_{\mathfrak{t}'}, \mathfrak{t}')$ with an accepting run $(\ell^{\mathcal{A}}_{\mathfrak{t}'}, \mathfrak{t}')$ of $\mathcal{A}$ satisfying the following conditions:*

- *for all nodes $v \in \mathfrak{t}'$ we have $|\mathsf{Chld}(v)| \leq 2^{|Q|}(E \cdot C)^{2E+1}$ and $|\{\ell^{\mathcal{A}}_{\mathfrak{t}'}(u) \mid u \in \mathsf{Chld}(v)\}| \leq 2E \cdot \log(4CE)$.*
- *On any path $v_1, v_2, \ldots, v_n$ of length $n > 2^{|Q|}$ in $\mathfrak{t}'$ there is a node v such that $\ell^{\mathcal{A}}_{\mathfrak{t}'}(v)$ contains a final state.*

Proof. We construct $(\ell^{\mathcal{A}}_{\mathfrak{t}'}, \ell_{\mathfrak{t}'}, \mathfrak{t}')$ recursively, starting from $(\ell^{\mathcal{A}}_{\mathfrak{t}}, \ell_{\mathfrak{t}}, \mathfrak{t})$. The second condition can be established simply by taking any finite path $v_1, v_2, \ldots, v_n$ of length $n > 2^{|Q|}$ violating it and by observing, by the pigeonhole principle, that two nodes v_i, v_j $(i < j)$ have the same value of $\ell^{\mathcal{A}}_{\mathfrak{t}'}$. Hence, we obtain a new tree by replacing the subtree of v_i by a subtree of v_j. We repeat the process indefinitely. For the first condition, assume that there is a node v violating the condition. Per

each subset $X \subseteq Q$ let v_X be any fixed child of v with $\ell^{\mathcal{A}}_{\mathfrak{t}'}(v_X) = X$ (if it exists). Let $\mathcal{I} = \{\mathcal{C} \mid q \in \ell^{\mathcal{A}}_{\mathfrak{t}'}(v), (q \to \mathcal{C}) \in I\}$ be a system of inequalities. Note that in $\mathcal{I}$ we have unknowns of the form x_X per each $X \subseteq Q$, which by the semantics of $\mathcal{A}$ counts nodes labelled by some superset of X. But $\mathcal{I}$ can be rewritten into a system $\mathcal{I}'$ (by applying the inclusion–exclusion principle) so that the unknowns x_X indicates the total number of children labelled by *precisely the set* X. Thus, the assignment $S : X \mapsto |\{u \in \mathsf{Chld}(v) \mid \ell^{\mathcal{A}}_{\mathfrak{t}'}(u) = X\}|$ is the solution to $\mathcal{I}'$. By employing Lemma 2 to $\mathcal{I}'$ we infer the existence of a "sparse and small" solution S' to $\mathcal{I}'$ with $\mathsf{supp}(S') \subseteq \mathsf{supp}(S)$. We conclude by modifying $\mathfrak{t}'$ as follows: we remove all descendants of v and for any subset $X \subseteq Q$ with $S'(x_X) \neq 0$ we assign as children of v precisely $S'(x_X)$ copies of the previously selected elements v_X. The labellings $\ell^{\mathcal{A}}_{\mathfrak{t}'}, \ell_{\mathfrak{t}'}$ are as indicated by the "copies". The resulting tree is accepted by $\mathcal{A}$: systems of constraints are satisfied (again by applying the inclusion-exclusion principle on S') and the Büchi acceptance condition still holds, since the "copied" subtrees were present in the original run.

We next describe APSpace (= ExpTime by [13]) procedure for testing non-emptiness of $\mathcal{L}$. It will be tableaux-like: we start by guessing a node, then we guess its children, verify the consistency of the guess with the transition function of $\mathcal{A}$ and repeat the process from a universally selected child. In the pseudocode below we identify a node with a pair $(q, \ell) \in 2^Q \times 2^\Sigma$. We employ two counters, stored in binary. The counter `FCnt` is responsible for counting how many times we encountered an accepting state (if $\texttt{FCnt} > 2^{|Q|}$ we know that we ended up in the same accepting configuration twice and hence, we can build the tree by making precisely the same choices as we did in the past). The counter `QCnt` is responsible for counting how many configurations we have visited without entering into an accepting state. If $\texttt{QCnt} > 2^{|Q|}$ then some non-final state was surely repeated twice, violating the second condition of Lemma 16.

Speaking informally, the role of counters is to prevent us from mixing-up the numbers of already visited non-accepting and accepting elements on a branch: we want to make sure that there is a repetition of an accepting state but the path between two accepting nodes can be long (more than 2^Q), which potentially spoils the counting. The second counter ensures us that we count correctly.

1 **Guess** the root node $v = (q, \ell)$ and set $\texttt{FCnt} = 0$.
2 **If** $\texttt{FCnt} > 2^{|Q|}$ **then** *Accept* **else** set $\texttt{QCnt} = 0$.
3 **Guess** $\leq 2E \cdot \log(4CE)$ nodes (q_i, ℓ_i) that will be repeated **guessed** $n_i \leq 2^{|Q|}(E \cdot C)^{2E+1}$ times.
4 Check if the guessed children are consistent with the transition function of $\mathcal{A}$.
5 **Universally choose** (q_i, ℓ_i) as $v = (q, \ell)$.
6 **If** q contains a final state **then** `FCnt++` and **goto** 2.
7 `QCnt++`. **If** $\texttt{QCnt} > 2^{|Q|}$ **then** *Reject* **else goto** 3.

A proof of correctness can be found in Appendix B. We conclude:

Theorem 17. *The non-emptiness problem for SPBTAs is decidable in* ExpTime.

6 Applications in Logic

We assume familiarity basics on (description) logics and model theory [3,20]. The two-variable guarded fragment of first-order logic, GF^2, is a relevant fragment of FO that captures many standard description logics up to $\mathcal{ALCIHb}^{\mathsf{self}}$ [3,18]. Here we consider its expressive extension with *local Presburger quantifiers*, GP^2, introduced in [5]. For brevity we introduce it already in a suitable, polynomially computable, normal form [5, p. 10]. A formula φ is in GP^2 if it has the shape:

$$\forall x\, \gamma(x) \wedge \bigwedge_{i=1}^{n} \Big(\forall x \forall y\, \mathrm{E}_i(x,y) \to \alpha_i(x,y)\Big) \wedge \bigwedge_{i=1}^{m} \forall x A_i(x) \to \Big(\sum_{j=1}^{m_i} \lambda_{i,j} \cdot \#_y^{\mathrm{R}_{i,j}}[\top] \circledast_i \delta_i\Big),$$

where $\gamma(x)$, $\alpha_i(x,y)$ are quantifier-free formulae in NNF, each $e_i, A_i, r_{i,j}$ are relational symbols, all $\lambda_{i,j}$'s, δ_i's are integers, and $\circledast_i$ is one of $=, \neq, \leq, \geq, \equiv_d$ or $\not\equiv_d$, where $d \in \mathbb{N}_+$. Here $\equiv_d$ denotes the congruence modulo d. W.l.o.g. assume that there is a symbol D so that any element is required to a D-successor.

We restrict ourselves to finite-branching structures (the logic trivialises if we allow for infinite branching). The semantics of GP^2 is like in FO, except for the expressions $\#_y^{\mathrm{R}_i}[\top]$, which evaluate to the total number of R_i-successors y of x.

$$\mathfrak{A}, x/a \;\models\; \sum_{i=1}^{n} \lambda_i \cdot \#_y^{\mathrm{R}_i}[\top] \circledast \delta \quad \text{iff} \quad \sum_{i=1}^{n} \lambda_i \cdot |\{b \mid \mathfrak{A}, x/a, y/b \models \mathrm{R}_i(x,y)\}| \;\circledast\; \delta$$

A GP^2*-knowledge-base* (kb for brevity) is a pair $\mathcal{K} = (\mathbb{D}, \mathcal{T})$, where $\mathbb{D}$ is a *database*, *i.e.* the set of ground facts emplying indv. names and unary/binary predicates, and $\mathcal{T}$ is s set of GP^2 formulae. A structure $\mathfrak{A}$ satisfies $\mathcal{K}$ (written $\mathfrak{A} \models \mathcal{K}$) if $\mathfrak{A} \models \bigwedge \mathcal{T} \wedge \bigwedge \mathbb{D}$. In the kb-satisfiability problem we ask whether an input GP^2-kb has a (finite branching, possibly infinite) model. Note that this generalises the usual formula satisfiability. Fix a GP^2-kb $\mathcal{K} = (\mathbb{D}, \mathcal{T})$. The main goal is to construct an SPBTA $\mathcal{A}$ from $\mathcal{K}$ such that $\mathcal{L}(\mathcal{A})$ is non-empty iff $\mathcal{K}$ has a model.

We first argue that we can restrict ourselves to tree-like models (*i.e.* models whose underlying structure is a tree). Let $\mathfrak{A}$ be a (finite branching) model of $\mathcal{K}$. Let $G = (V, E)$ be a Gaifman graph of $\mathfrak{A}$, *i.e.* we have domain elements of $\mathfrak{A}$ as nodes and undirected edges between any two elements linked by some binary relation. Let $\mathsf{names}(\mathcal{K})$ be the set of all individual names appearing in $\mathcal{K}$. By a $\mathcal{K}$-named element in $\mathfrak{A}$ we mean any element $\mathrm{a}^{\mathfrak{A}}$ for $\mathrm{a} \in \mathsf{names}(\mathcal{K})$. By a $\mathcal{K}$*-path* in $\mathfrak{A}$ we mean any word $\mathrm{v}_1, \mathrm{v}_2, \ldots, \mathrm{v}_n$ from A^+ such that: (i) v_1 is $\mathcal{K}$-named in $\mathfrak{A}$, (ii) v_2 is not $\mathcal{K}$-named, (iii) any two v_i, v_{i+1} are connected by an edge in G, and (iv) any three consecutive $\mathrm{v}_i, \mathrm{v}_{i+1}, \mathrm{v}_{i+2}$ are pairwise different.

Definition 18. *The* $\mathcal{K}$-unravelling *of* $\mathfrak{A}$ *is a structure* $\mathfrak{B}$*, which domain is composed of all* $\mathcal{K}$*-paths in* $\mathfrak{A}$ *and the relational symbols appearing in* $\mathcal{K}$ *are interpreted as follows for all words* $\mathrm{wd} \in A^*A$ *and* $\mathrm{wde} \in A^*AA$*:*

- $a^{\mathfrak{B}} = \mathrm{a}^{\mathfrak{A}}$ *for all* $\mathrm{a} \in \mathsf{names}(\mathcal{K})$*, and* $\mathrm{wd} \in \mathrm{P}^{\mathfrak{B}}$ *iff* $d \in \mathrm{P}^{\mathfrak{A}}$ *for all unary* P*,*
- *For any* $x, y \in B$ *we put* $(x,y) \in \mathrm{R}^{\mathfrak{B}}$ *if and only if:*
  (i) $x = \mathrm{a}^{\mathfrak{B}}, y = \mathrm{b}^{\mathfrak{B}}$ *and* $(\mathrm{a}^{\mathfrak{A}}, \mathrm{b}^{\mathfrak{A}}) \in \mathrm{R}^{\mathfrak{A}}$*, (ii)* x=y=wd *and* $(\mathrm{d}, \mathrm{d}) \in \mathrm{R}^{\mathfrak{A}}$*, (iii)* $x = \mathrm{wd}, y = \mathrm{wde}$ *and* $(\mathrm{d}, \mathrm{e}) \in \mathrm{R}^{\mathfrak{A}}$*, or (iv)* $x = \mathrm{wde}, y = \mathrm{wd}$ *and* $(\mathrm{e}, \mathrm{d}) \in \mathrm{R}^{\mathfrak{A}}$*.*

The following fact justifies the use of unravellings and follows directly from the semantics of GP^2 (after a trivial renaming of structures).

Fact 19 *$\mathfrak{B} \models \mathcal{K}$ and for any $\mathrm{a} \in \mathsf{names}(\mathcal{K})$ the substructure of $\mathfrak{B}$ restricted to all $\mathcal{K}$-paths starting from $\mathrm{a}^{\mathfrak{A}}$ constitute a tree.*

Call a database *simple*, if it has the form $\{\mathrm{U(a)}\}$ for some indv. name a and a unary predicate symbol U. It is a standard trick in DLs that one can reduce the kb-satisfiability problem to polynomially many subtasks of kb-satisfiability for instances having simple databases only. Indeed, observe first that the unravelled model $\mathfrak{B}$ is just a collection of trees rooted at named elements, were the roots can be interlinked by some binary relations. As the number of individual names appearing in $\mathsf{names}(\mathcal{K})$ is clearly linear in $|\mathcal{K}|$, one can simply guess the substructure $\mathfrak{C} := \mathfrak{B} \restriction \{\mathrm{a}^{\mathfrak{B}} \mid \mathrm{a} \in \mathsf{names}(\mathcal{K})\} \models \mathbb{D}$ and after verifying its consistency (doable in polynomial time), encode connection between a given root and the other roots as follows. Let us introduce fresh unary predicates $\mathrm{DB}_{\pm \mathrm{R}(a,b)}, \mathrm{DB}_{\pm \mathrm{U(a)}}, \mathrm{Root}_a$ per each binary relation symbol R and names $\mathrm{a}, \mathrm{b} \in \mathsf{names}(\mathcal{K})$, and a binary predicate symbol insideDB. Fix a name $\mathrm{a} \in \mathsf{names}(\mathcal{K})$; we shall write a reduction for an $\mathrm{a}^{\mathfrak{B}}$ point of view. The reduction for other individual names is analogous. We append to the $\mathcal{T}$-component of $\mathcal{K}$ the following formulae: (i) whenever Root_a is satisfied at x then for each $\mathrm{b} \neq \mathrm{a} \in \mathsf{names}(\mathcal{K})$ we have that x has precisely one insideDB-successor y labelled with $\mathrm{Root}_{\mathrm{b}}$; moreover the pair (x, y) should satisfy precisely the same unary and binary relations as indicated in $\mathfrak{C}$. (ii) Whenever x satisfies $\mathrm{Root}_{\mathrm{a}}$, it encodes the full structure $\mathfrak{C}$ with the help of unary predicate symbols $\mathrm{DB}_{\pm \mathrm{R(a,b)}}, \mathrm{DB}_{\pm \mathrm{U(a)}}$, *e.g.* if $(\mathrm{c}^{\mathfrak{B}}, \mathrm{d}^{\mathfrak{B}}) \notin \mathrm{S}^{\mathfrak{B}}$ then the satisfaction of $\mathrm{Root}_{\mathrm{a}}$ implies the satisfaction of $\mathrm{DB}_{-\mathrm{S(c,d)}}$. Note that we do not impose any constraints on successors of the elements labelled with $\mathrm{Root}_{\mathrm{b}}$ for $\mathrm{b} \neq a$. This will be taken into account in the reduction for other individual names (here the role of such elements is only to make the counting constraints of $\mathcal{K}$ work). This yields the desired reduction. Take $\mathcal{K}_{\mathrm{a}}^{\mathfrak{C}} = (\{\mathrm{Root}_a(a)\}, \varphi_{\mathrm{a}}^{\mathfrak{C}} := \bigwedge_{\varphi \in \mathcal{T}} \varphi \wedge \psi_{\mathrm{a}}^{\mathfrak{C}})$, where $\psi_{\mathrm{a}}^{\mathfrak{C}}$ simply contains the extra formulae that we discussed. It is easy to show that:

Lemma 20. *A GP^2-kb $\mathcal{K}$ is satisfiable iff there is a structure $\mathfrak{C} \models \mathbb{D}$ (with at most $|\mathsf{names}(\mathcal{K})|$ elements) such that all $\mathcal{K}_a^{\mathfrak{C}}$ (defined above) are satisfiable.*

Proof. ($\Rightarrow$). Take the unravelled model $\mathfrak{B}$ of $\mathcal{K} = (\mathbb{D}, \mathcal{T})$, and let $\mathfrak{C}$ to be the restriction of $\mathfrak{B}$ to the roots of $\mathfrak{B}$. Make $\mathrm{a}^{\mathfrak{B}}$ to be the only elements satisfying the symbol $\mathrm{Root}_{\mathrm{a}}$ and interpret predicate symbols $\mathrm{DB}_{\pm \mathrm{R(a,b)}}$, $\mathrm{DB}_{\pm \mathrm{U(a)}}$ in a above-described way. Finally let $\mathrm{insideDB}^{\mathfrak{B}}$ be the full (non-reflexive) binary relation on $\mathfrak{C}$. Take $\mathfrak{B}_{\mathrm{a}}$ as the substructure of $\mathfrak{B}$ obtained by removing all pairs having the form $(\mathrm{b}^{\mathfrak{B}}, \mathrm{c}^{\mathfrak{B}})$ for $\mathrm{b} \neq \mathrm{a}$ from all binary relations. It follows that $\mathfrak{B}_{\mathrm{a}} \models \mathcal{K}_{\mathrm{a}}$. ($\Leftarrow$). Let $\mathfrak{B}_{\mathrm{a}}$ be tree-models of $\mathcal{K}_{\mathrm{a}}^{\mathfrak{C}}$. Thus, by definition of $\mathrm{DB}_{\pm \mathrm{R(a,b)}}$, $\mathrm{DB}_{\pm \mathrm{U(a)}}$ predicates, there is unique $\mathfrak{C} \models \mathbb{D}$ described inside $\mathcal{K}_{\mathrm{a}}^{\mathfrak{C}}$. Let $\mathfrak{C}_{\mathrm{a}}$ be a substructure of $\mathfrak{B}_{\mathrm{a}}$ obtained by removing elements that are descendants of insideDB-successors of $\mathrm{a}^{\mathfrak{B}_{\mathrm{a}}}$ labelled with $\mathrm{Root}_{\mathrm{b}}$ for $\mathrm{b} \in \mathsf{names}(\mathcal{K})$. Take $\mathfrak{D}$ to be the disjoint union of all $\mathfrak{C}_{\mathrm{a}}$ with $\mathrm{a}^{\mathfrak{D}}$ interpreted as the root of $\mathfrak{C}_{\mathrm{a}}$. Finally, make the restriction of $\mathfrak{D}$ to the roots of $\mathfrak{C}_{\mathrm{a}}$ isomorphic to $\mathfrak{C}$. The resulting structure $\mathfrak{D}$ is a model of $\mathcal{K}$.

After the above reduction, we are going to explain how we will encode tree-like models inside trees. For the rest of the paper fix the kb $\mathcal{K}_{\mathrm{a}}^{\mathfrak{C}} = (\{\mathrm{Root}_{\mathrm{a}}(a)\}, \varphi_{\mathrm{a}}^{\mathfrak{C}})$. We first prepare the alphabet for our automaton. Let $\mathcal{S}_0$ be the set of all possible binary and unary atoms for predicate names appearing in $\varphi_a^{\mathfrak{C}}$, involving only the variables x and y. Take $\mathcal{S}_1$ to be composed of all subformulae of α_i and γ from $\varphi_{\mathrm{a}}^{\mathfrak{C}}$ (consult the normal form if needed). Finally take[5] $\mathcal{S} := \mathcal{S}_0 \cup \mathcal{S}_1$ and close it under a single negation. Note that $|\mathcal{S}|$ is of size polynomial in $|\mathcal{K}_{\mathrm{a}}^{\mathfrak{C}}|$.

Let the alphabet Σ be equal to

$$\Sigma := \{\sigma_\psi \mid \psi \in \mathcal{S}\} \cup \{\mathrm{Root}_a, \mathrm{Parent}, \mathrm{Self}, \mathrm{Child}\}. \tag{1}$$

The intuition behind Σ is as follows. Roughly speaking, whenever a symbol σ_ψ will label a node in a tree-encoding of a tree-like structure $\mathfrak{A}$, this will indicate that the corresponding element in $\mathfrak{A}$ satisfies ψ. To speak about formulae satisfied at a node, we always think that the variable y is evaluated at the current node, while the variable x corresponds to its parent. In order to verify cardinality constraints, note that SPBTA can count only node's children, while the expressions $\#_y^{\mathrm{R}_{i,j}}[\top]$ can also count the node itself as a witness as well as its parent. Thus a naive translation to automata may destroy counting constraints. To handle this issue, every node in a tree-encoding of $\mathfrak{A}$ will have two extra children, being "copies" of itself and its parent. This, after the introduction of some simple consistency rules, will make cardinality constraints work.

The formal definition of the encoding is given next. This is going to be *very* technical but we are required such a notion in order capture all necessary properties. The first two items imply the satisfaction of $\mathcal{K}_{\mathrm{a}}^{\mathfrak{C}}$, while the others are responsible for the correct evaluation of formulae and for propagating atoms from parent to child as well as to different copies.

Definition 21. *A* tree encoding *of $\mathcal{K}$ is a Σ-labelled tree* $\mathfrak{t}$ *satisfying:*

- *The root of* $\mathfrak{t}$ *is the unique node in* $\mathfrak{t}$ *labelled with* Root_a.
- *If* v *is a* Child*-labelled node or the root of* $\mathfrak{t}$ *then it is labelled with σ_γ and if* v *is labelled by $\sigma_{\mathrm{A}_i(y)}$ then the total number of its children labelled by $\sigma_{\mathrm{R}_{i,j}(x,y)}$ satisfies the cardinality constraint $\circledast_i\, \delta_i$ (*cf. *normal form).*
- *The letters* Parent*,* Child*,* Self *and* $\mathrm{Root}_{\mathrm{a}}$ *label disjoint sets of nodes in* $\mathfrak{t}$*. Every* Child*-labelled node has a unique child labelled with* Parent *and a unique child labelled with* Self*. The root of* $\mathfrak{t}$ *has a unique* Self*-labelled child, but it does not have any* Parent*-labelled children.*
- *If the root of* $\mathfrak{t}$ *is labelled by some σ_ψ then ψ uses the variable y only.*
- *For every unary (resp. binary) symbol* U *(resp.* B*) from $\mathcal{K}_{\mathrm{a}}^{\mathfrak{C}}$ and every node v from* $\mathfrak{t}$*, v is labelled by precisely one of $\sigma_{\pm \mathrm{U}(y)}$ (resp. by one of $\sigma_{\pm \mathrm{B}(y,y)}$). If v is neither the root of* $\mathfrak{t}$ *nor the* Self*-labelled child of the root, then we also take all other atomic formulae involving x and/or y into account.*
- *Labelling of nodes is consistent,* i.e. *a node v cannot be labelled by both σ_λ and $\sigma_{\neg\lambda}$ and that if v is labelled by $\sigma_{\lambda\wedge\lambda'}$ (resp. $\sigma_{\lambda\vee\lambda'}$) then v is labelled by both (resp. one of) σ_λ, $\sigma_{\lambda'}$ (the converse also holds for $\sigma_{\lambda\vee\lambda'}$, $\sigma_{\lambda\wedge\lambda'}$ from Σ).*

[5] In the literature $\mathcal{S}$ appears under the name of a syntactic closure.

– *If v is a* Child*-labelled node or the root of* $\mathfrak{t}$*, then (i) if* v *is labelled by some letter* $\sigma_{\pm\lambda(y)}$ *then every* Child*-labelled child of* v *is labelled by* $\sigma_{\pm\lambda(x)}$*, the* Self*-labelled child of v is labelled with both* $\sigma_{\pm\lambda(y)}$ *and* $\sigma_{\pm\lambda(x)}$ *as well as by letters obtained by replacing binary relations* $\mathrm{B}(y,y)$ *with* $\mathrm{B}(x,y)$*; (ii) if v is labelled by some letter* $\sigma_{\pm\lambda(x,y)}$ *then its* Parent*-labelled child is labelled by* $\sigma_{\pm\lambda'(x,y)}$*, where λ' is obtained by replacing every occurrence of the variable x with y and vice-versa.*

A crucial lemma, linking tree encodings with tree models is:

Lemma 22. *There exists a tree encoding of $\mathcal{K}$ iff $\mathcal{K}_a^{\mathfrak{C}}$ is satisfiable.*

Proof. ($\Leftarrow$). Let $\mathfrak{t}$ be the promised tree encoding. First, remove all Parent-labelled and Self-labelled children from $\mathfrak{t}$ together with their subtrees. Then, we interpret unary symbols U so that its interpretation contains precisely the nodes labelled with $\sigma_{\mathrm{U}(y)}$. For binary relations R we take the union of two cases: (i) we include all pairs (v,v) for v labelled with $\sigma_{\mathrm{R}(y,y)}$, (ii) we include all parent-child pairs (v,w) (resp. child-parent pairs (w,v)) for which we have that v is labelled with $\sigma_{\mathrm{R}(x,y)}$ (resp. $\sigma_{\mathrm{R}(y,x)}$). Let $\mathfrak{A}$ be the obtained structure. It is immediate to see by construction that $\mathfrak{A}, x/\mathrm{v}, y/\mathrm{w} \models \pm\lambda$ whenever w is labelled with $\sigma_{\pm\lambda}$ where λ ranges over subformulae from $\mathcal{S}$ and v is a parent of w. The constraints from $\mathcal{K}_{\mathrm{a}}^{\mathfrak{C}}$ are fulfilled due to our trick with extra nodes (we took parents and the elements themselves into account when counting). Hence, $\mathfrak{A} \models \mathcal{K}_{\mathrm{a}}^{\mathfrak{C}}$.

($\Rightarrow$). For the opposite side we will take a tree model $\mathfrak{A} \models \mathcal{K}_{\mathrm{a}}^{\mathfrak{C}}$ and "convert it" into a tree representation. As the prelimiary step, label all the elements of $\mathfrak{A}$ but root with a Child predicate symbol. After the first step, we select any Child-labelled element v, create a fresh copy of its parent w and insert a copy of w as a child of v. We maintain all the relations between v and the copy of w as they were between v and w. Finally, we make the Parent predicate to be satisfied at the copy of w. After the second step, for every element v in $\mathfrak{A}$ we copy v and make the copy a child of v and make the predicate Self to be satisfied at the copy. Then each of these copies is connected to their original elements with all binary R for which v has an R-loop. We repeat the process infinitely often in order to provide a child per each node. Finally, take any element v and label it with all predicates from $\{\sigma_{\pm\lambda(y)} \mid \mathfrak{A}, \mathrm{v} \models \lambda, \lambda \in \mathcal{S}\}$, and whenever v has a Child-labelled child w then we additionally make w to be labelled with $\{\sigma_{\pm\lambda(x)}, \sigma_{\pm\lambda'(x,y)} \mid \mathfrak{A}, x/\mathrm{v}, y/\mathrm{w} \models \lambda', \mathfrak{A}, x/\mathrm{v} \models \lambda, \lambda, \lambda' \in \mathcal{S}\}$. It is immediate to check that the structure $(\ell_{\mathfrak{t}}, \mathfrak{t} = A)$ with $\ell_{\mathfrak{t}}$ inherited from $\mathfrak{A}$ is a tree-encoding.

Relying on Definition 21, the construction of a polynomial-size SPBTA $\mathcal{A}$ accepting Σ-labelled tree encoding is straightforward. Thus, we delegate it to Appendix C. Since the satisfiability of GP^2 is known to be ExpTime-hard, we conclude the last result of the paper:

Theorem 23. *The knowledge-base satisfiability problem for* GP^2 *over finite-branching models is* ExpTime*-complete. Hence, the non-emptiness problem for SPBTA is* ExpTime*-hard.*

Acknowledgements

This work was supported by the Polish Ministry of Science and Higher Education program "Diamentowy Grant" no. DI2017 006447. We are grateful to Tim Lyon and Reijo Jaakkola for proofreading and language improvements.

Results from this paper will appear in the BSc thesis of Oskar Fiuk, written under informal supervision of Bartosz Bednarczyk at the University of Wrocław.

References

1. Baader, F.: A new description logic with set constraints and cardinality constraints on role successors. In: FroCoS 2017 (2017)
2. Baader, F., Hladik, J., Peñaloza, R.: Automata can show pspace results for description logics. Inf. Comput. (2008)
3. Baader, F., Horrocks, I., Lutz, C., Sattler, U.: An Introduction to Description Logic (2017)
4. Bednarczyk, B.: One-variable logic meets presburger arithmetic. Theor. Comput. Sci. (2020)
5. Bednarczyk, B., Orlowska, M., Pacanowska, A., Tan, T.: On Classical Decidable Logics Extended with Percentage Quantifiers and Arithmetics. In: FSTTCS 2021 (2021)
6. Boiret, A., Hugot, V., Niehren, J., Treinen, R.: Automata for unordered trees. Inf. Comput. (2017)
7. Boiret, A., Hugot, V., Niehren, J., Treinen, R.: Logics for unordered trees with data constraints. J. Comput. Syst. Sci. (2019)
8. Bonatti, P.A., Lutz, C., Murano, A., Vardi, M.Y.: The complexity of enriched mu-calculi. Log. Methods Comput. Sci. (2008)
9. Boneva, I., Talbot, J.: Automata and logics for unranked and unordered trees. In: Giesl, J. (ed.) RTA 2005
10. Borosh, I., Flahive, M., Treybig, B.: Small solutions of linear diophantine equations. Discret. Math. (1986)
11. Calvanese, D., Carbotta, D., Ortiz, M.: A practical automata-based technique for reasoning in expressive description logics. In: Walsh, T. (ed.) IJCAI 2011 (2011)
12. Calvanese, D., Eiter, T., Ortiz, M.: Answering regular path queries in expressive description logics via alternating tree-automata. Inf. Comput. (2014)
13. Chandra, A.K., Stockmeyer, L.J.: Alternation. In: FOCS 1976 (1976)
14. Comon, H.: Tree automata techniques and applications (1997)
15. Demri, S., Lugiez, D.: Complexity of modal logics with presburger constraints. J. Appl. Log. (2010)
16. Eisenbrand, F., Shmonin, G.: Carathéodory bounds for integer cones. Oper. Res. Lett. (2006)
17. El-Sappagh, S., Franda, F., Ali, F., Kwak, K.S.: SNOMED CT standard ontology based on the ontology for general medical science. BMC medical informatics and decision making (2018)
18. Grädel, E.: Description logics and guarded fragments of first order logic. In: DL (1998)
19. Kupke, C., Pattinson, D.: On modal logics of linear inequalities. In: Beklemishev, L.D., Goranko, V., Shehtman, V.B. (eds.) AIML 2010 (2010)
20. Libkin, L.: Elements of Finite Model Theory (2004)
21. Papadimitriou, C.H.: On the complexity of integer programming. (JACM) (1981)

22. Schwentick, T.: Trees, automata and XML. In: Beeri, C., Deutsch, A. (eds.) PODS 2004 (2004)
23. Seidl, H., Schwentick, T., Muscholl, A.: Counting in trees. In: Flum, J., Grädel, E., Wilke, T. (eds.) Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas] (2008)
24. Sipser, M.: Introduction to the theory of computation. ACM Sigact News (1996)

A Missing details from the proof of Lemma 13

Now we will show formally, that indeed $(\ell_Q, \mathfrak{t})$ is an accepting run of $\mathcal{A}$ on the tree $(\ell_\Sigma, \mathfrak{t})$ with respect to the given definition of (q_0, α, Γ).

- Since α and Γ are partial functions, it is necessary to verify that the tree $\mathfrak{t}$ is a well-defined, i.e. there is no $v \in \mathfrak{t}$ such that $\Gamma(\ell_Q(v)) = \bot$. But observe that from construction of Γ we have $\{q' \in Q \mid q \in Q_0, \Gamma(q)(q') > 0\} \subseteq Q_0 = dom(\Gamma) = dom(\alpha)$.
- Clearly, the initial configuration is $l_Q(\varepsilon) = q_0$.
- All linear constraints from the I component are still satisfied, since the quantitative Q-labelling of children for each node in $\mathfrak{t}$ corresponds to the quantitative Q-labelling of children for some node in the original tree $\mathfrak{t}'$ with the same Σ-label. Here, by quantitative Q-labelling we mean that we care only for the number of children labelled with a given state.
- Suppose that, there is a branch $v_1, v_2, \ldots$ with only a finite number of states from F. Let v_t be the last node with accepting state. Consider a sequence $\{j(\ell_Q(v_{t+i}))\}_{i\in\mathbb{N}}$. Note that $j(\ell_Q(v_{t+i+1})) <_{lex} j(\ell_Q(v_{t+i}))$, unless $\ell_Q(v_{t+i}) \in F$, since all leaves in every $\mathsf{SSubt}(q_i, v_i)$ are labelled by the states from F. But $<_{lex}$ on $\mathbb{N}^2 \times \mathbb{N}^*$ is a well-founded order. Therefore, there must be another accepting state.

B Correctness proof of the algorithm from Section 5

Let $\mathcal{A}$ be a SPBTA given as input to the algorithm. We will show that $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff the algorithm accepts on $\mathcal{A}$.

- Suppose that $\mathcal{L}(\mathcal{A})$ is non-empty. Then let $(\ell_\mathfrak{t}, \mathfrak{t})$ be a tree with accepting run $(\ell^\mathcal{A}_\mathfrak{t}, \mathfrak{t})$ satisfying Lemma 16. It is clear that the algorithm at each node can guess the children with a labelling consistent with $\ell_\mathfrak{t}$ and $\ell^\mathcal{A}_\mathfrak{t}$. Since on each branch final states are appearing infinitely many times and the length of any path between the node and a predecessor such that both endpoints are labelled with final states is at most $2^{|Q|}$, the `QCnt` is cleared more often than every $\mathcal{O}(2^{|Q|})$ steps. Hence, the algorithm will never enter into the *Reject* state. On the other hand, each time, whenever the algorithm sees an accepting state, the `FCnt` is incremented. Therefore, the algorithm will enter into the *Accept* state after at most $\mathcal{O}(2^{2|Q|})$ steps.
- In the other direction, suppose that the algorithm accepts. Then one can construct a tree from $\mathcal{L}(\mathcal{A})$. Let $(\ell_\mathfrak{t}, \mathfrak{t})$ and $(\ell^\mathcal{A}_\mathfrak{t}, \mathfrak{t})$ be the finite trees mirroring the choices of the algorithm in the natural way. Since the `FCnt` $> 2^{|Q|}$, on each path there is a final configuration that appeares at least twice. Trim the tree in a way that each leaf is labelled with a such final configuration. Formally, let
$$\mathfrak{t}_0 = \{x \in \mathfrak{t} \mid \exists y \quad xy \in \mathcal{W}\},$$
where $\mathcal{W} = \{x \in \mathfrak{t} \mid \exists st = x, t \neq \varepsilon, \ell^\mathcal{A}_\mathfrak{t}(s) = \ell^\mathcal{A}_\mathfrak{t}(x), \ell^\mathcal{A}_\mathfrak{t}(x) \cap F \neq \emptyset\}$, and let $(\ell_{\mathfrak{t}_0}, \mathfrak{t}_0)$ and $(\ell^\mathcal{A}_{\mathfrak{t}_0}, \mathfrak{t}_0)$ be the appropriate labellings.
The desired tree can be constructed by employing the following process indefinitely. For each leaf v of $\mathfrak{t}_0$ let p_v be a node for which exists $x \neq \varepsilon$ such

that $p_v x = v$ and $\ell^{\mathcal{A}}_{\mathfrak{t}_0}(p_v) = \ell^{\mathcal{A}}_{\mathfrak{t}_0}(v)$, i.e. a predecessor labelled with the same final configuration. Then replace each leaf v with a whole subtree of p_v.
It is clear that all linear constraints from the I component of $\mathcal{A}$ are still satisfied and on each branch an accepting state occurs infinitely many times. Thus, a resulting tree is in $\mathcal{L}(\mathcal{A})$.

The last thing is to ensure that the procedure indeed can be implemented in a polynomial space. But before doing so, we need to make assumptions on encoding of SPBTA. We assume that $|Q|, |\Sigma|, |\mathcal{I}| \in \mathcal{O}(|A|)$, where $|\mathcal{I}|$ is the size of an encoding of I with all the coefficients encoded in binary. As stated in Section 2 not all the variables from $\mathcal{V}(\mathcal{I})$ have to be mentioned explicitly. And in Section 5 we defined N to be the number of constraints in the I, hence $N = \mathcal{O}(|A|)$, and C to be the minimal absolute bound on the coefficients from I, thus $\log C = \mathcal{O}(|A|)$.

The algorithm needs the $2|Q|$ bits to keep the binary counters `FCnt` and `QCnt`. For the node $v = (q, \ell)$ the $\mathcal{O}(|Q| \log |Q| + |\Sigma| \log |\Sigma|) = \mathcal{O}(|A| \log |A|)$ bits is enough (a trivial encoding as a list of elements from each set). In the *Step 3.* the algorithm needs to guess simultaneously the $\leq 2N \cdot \log(4CN) = \mathcal{O}(|A|^2)$ triples (q_i, ℓ_i, n_i), where $n_i \leq 2^{|Q|}(N \cdot C)^{2N+1}$, i.e. $\log n_i \leq \log n_{\max} = \mathcal{O}(|A|^2)$. To verify the consistency of guesses with the transition function of $\mathcal{A}$ each constraint E from I can be checked independently and one by one. This can be done by computing the values $x_S = \sum_{q_i \supseteq S} n_i$ for all $x_S \in \mathcal{V}(E)$, and plugging them into the expression, and verifying the result. Hence, the space needed for each x_S is $\log x_S \leq \log x_{S_{\max}} = \mathcal{O}(|A|^2)$, and for the whole E:

$$(\log C + \log x_{S_{\max}}) \cdot |E| \leq \mathcal{O}(|A|^3),$$

where $|E|$ is the number of coefficients in E.

Therefore, the total space can be bounded by

$$2|Q| + (2N \cdot \log(4CN) + 1) \cdot (|Q| \log |Q| + |\Sigma| \log |\Sigma| + \log n_{\max}),$$

which is at most $\mathcal{O}(|A|^4)$. Thus the described procedure is in APSPACE.

C Construction of a polynomial-size SPBTA

Below we give the translation of the Definition 21 into a polynomial-time procedure constructing an SPBTA $\mathcal{A} = (\Sigma^{\mathcal{A}}, Q^{\mathcal{A}}, F^{\mathcal{A}}, Q_0^{\mathcal{A}}, I^{\mathcal{A}})$ such that $\mathcal{L}(\mathcal{A})$ consists of exactly those trees which corresponds to the correctly encoded tree-like models.

For the $\Sigma^{\mathcal{A}}$ component just take the Σ from Equation (1). Let at the beginning $Q^{\mathcal{A}} = \emptyset$ and $I^{\mathcal{A}} = \emptyset$, and then apply imperatively the following steps:

- Create a "trivial" state $q_\top \in Q^{\mathcal{A}}$ and add instr. $-q_\top \to q_\top \in I^{\mathcal{A}}$.
- Create a state q_σ for each $\sigma \in \Sigma$ and add instr. $\sigma \to q_\sigma, q_\sigma \to \sigma \in I^{\mathcal{A}}$; *i.e. reading input symbols into automaton states.*
- Add instr. $q_\top \to x_{q_{\mathrm{Root}_a}} = 0 \in I^{\mathcal{A}}$; *i.e. ensure that only true tree root can be labelled with* Root_a.
- Add instr. $q_a \to q_{\sigma_{\gamma(y)}} \in I^{\mathcal{A}}$ for $a \in \{\mathrm{Root}, \mathrm{Child}\}$; *i.e. ensure the universal component of the normal form.*

- Add instr. $q_{\sigma_{A_i(y)}} \to \left(\sum_{j=1}^{m_i} \lambda_{i,j} \cdot x_{q_{\sigma_{r_{i,j}(x,y)}}} \circledast_i \delta_i \right) \in I^{\mathcal{A}}$ for each $\forall x\ A_i(x) \to \left(\sum_{j=1}^{m_i} \lambda_{i,j} \cdot \#_y^{r_{i,j}}[\top] \circledast_i \delta_i \right)$; *i.e. ensure linear constraints on successors.*
- Add instr. $q_a \to -q_b \in I^{\mathcal{A}}$ for each $a, b \in \{\text{Parent}, \text{Child}, \text{Self}, \text{Root}_a\}, a \neq b$; *i.e. ensure that* Parent*,* Child*,* Self *and* Root_a *label disjoint set of nodes.*
- Add instr. $q_{\text{Child}} \to x_{q_a} = 1 \in I^{\mathcal{A}}$ for $a \in \{\text{Parent}, \text{Self}\}$; *i.e. ensure that every* Child*-labelled node has exactly one child with label* Parent*, and exactly one child with label* Self*.*
- Add instr. $q_{\text{Root}_a} \to x_{q_{\text{Self}}} = 1, q_{\text{Root}_a} \to x_{q_{\text{Parent}}} = 0 \in I^{\mathcal{A}}$; *i.e. ensure that* Root_a*-labelled node has exactly one child with label* Self*, and no children with label* Parent*.*
- Add instr. $q_{\text{Root}_a} \to -q_{\sigma_{\psi(x,y)}}, q_{\text{Root}_a} \to -q_{\sigma_{\psi(x)}} \in I^{\mathcal{A}}$; *i.e. ensure that root uses only formulae with y variable.*
- Add instr. $\pm q_{\sigma_a} \to \mp q_{\sigma_{-a}} \in I^{\mathcal{A}}$ for each $a \in \{\text{U}(y) : \text{U unary symbol}\} \cup \{\text{B}(y,y) : \text{B binary symbol}\}$; *i.e. ensure that every node is labelled by precisely one of* $q_{\sigma_{\pm \text{U}(y)}}$ *and* $q_{\sigma_{\pm \text{B}(y,y)}}$*.*
- Introduce helper states $q_{-\text{Root}_a}, q_{-\text{Self}}, q_{\pm\text{RootOrSelf}} \in Q^{\mathcal{A}}$ with a natural semantics, i.e. $\pm q_a \to \mp q_{-a} \in I^{\mathcal{A}}$ for each $a \in \{\text{Root}_a, \text{Self}, \text{RootOrSelf}\}$, and $q_{\text{Root}_a} \to q_{\text{RootOrSelf}}, q_{\text{Self}} \to q_{\text{RootOrSelf}}, q_{-\text{Root}_a} \wedge q_{-\text{Self}} \to q_{-\text{RootOrSelf}} \in I^{\mathcal{A}}$. Then for each

  $$a \in \{U(x) : \text{U unary symbol}\} \cup \{B(x,x), B(x,y), B(y,x) : \text{B binary symbol}\}$$

  create states $q_{\text{Missing}\pm a}$ and add instr. $-q_{\sigma_{\pm a}} \to q_{\text{Missing}\pm a}, q_{\text{Missing}+a} \wedge q_{\text{Missing}-a} \to q_{\text{RootOrSelf}} \in I^{\mathcal{A}}$; *i.e. extends previous point with all other atomic formulae, unless node is labelled with* Root_a *or* Self*.*
- For each pair of states $q_{\sigma_\lambda}, q_{\sigma_{\neg\lambda}}$ add instr. $q_{\sigma_{\pm\lambda}} \to -q_{\sigma_{\mp\lambda}} \in I^{\mathcal{A}}$; similarly for each triple of states $q_{\sigma_{\lambda\wedge\lambda'}}, q_{\sigma_\lambda}, q_{\sigma_{\lambda'}}$ (resp. $q_{\sigma_{\lambda\vee\lambda'}}, q_{\sigma_\lambda}, q_{\sigma_{\lambda'}}$) add instr. $q_{\sigma_\lambda} \wedge q_{\sigma_{\lambda'}} \to q_{\sigma_{\lambda\wedge\lambda'}}, q_{\sigma_{\lambda\wedge\lambda'}} \to q_{\sigma_\lambda}, q_{\sigma_{\lambda\wedge\lambda'}} \to q_{\sigma_{\lambda'}} \in I^{\mathcal{A}}$ (resp. $q_{\sigma_{\neg\lambda}} \wedge q_{\sigma_{\neg\lambda'}} \to q_{\sigma_{\neg(\lambda\vee\lambda')}}, q_{\sigma_\lambda} \to q_{\sigma_{\lambda\vee\lambda'}}, q_{\sigma_{\lambda'}} \to q_{\sigma_{\lambda\vee\lambda'}} \in I^{\mathcal{A}}$); *i.e. consistency checking.*
- Add instr. $q_l \wedge q_{\sigma_{\pm\lambda(y)}} \to x_{\{q_{\sigma_{\pm\lambda(x)}}, q_{l'}\}} = x_{q_{l'}}$ for each $l \in \{\text{Root}_a, \text{Child}\}$, and $l' \in \{\text{Child}, \text{Self}\}$; *i.e. copy information about unary predicates to all children and a self-copy.*
- Add instr. $q_l \wedge q_{\sigma_{\pm\Lambda}} \to x_{\{q_{\sigma_{\pm\Lambda'}}, q_{\text{Self}}\}} = x_{q_{\text{Self}}}$ for each $l \in \{\text{Root}_a, \text{Child}\}$ and $(\Lambda, \Lambda') \in \{(\lambda(y), \lambda(y)), (\lambda(y,y), \lambda(x,y))\}$; *i.e. copy information about unary predicates and unravelled loops to a self-copy.*
- Add instr. $q_{\text{Child}} \wedge q_{\sigma_{\pm\lambda(x,y)}} \to x_{\{q_{\sigma_{\pm\lambda(y,x)}}, q_{\text{Parent}}\}} = x_{q_{\text{Parent}}}$; *i.e. copy information to parent-copy, but with swapped variables x and y.*

Finally, set initial configuration $Q_0 = \{q_{\text{Root}_a}\}$ and accepting states $F = \{q_\top\}$.