The Emptiness Problem for Tree Automata with Global Constraints

Luis Barguñó Carles Creus Guillem Godoy
Universitat Politècnica de Catalunya
Barcelona, Spain

Florent Jacquemard
INRIA Saclay
LSV-CNRS/ENS Cachan

Camille Vacher
France Telecom R&D
LSV-CNRS/ENS Cachan

***Abstract*—We define tree automata with global constraints (TAGC), generalizing the class of tree automata with global equality and disequality constraints [1] (TAGED). TAGC can test for equality and disequality between subterms whose positions are defined by the states reached during a computation. In particular, TAGC can check that all the subterms reaching a given state are distinct. This constraint is related to monadic key constraints for XML documents, meaning that every two distinct positions of a given type have different values.**

We prove decidability of the emptiness problem for TAGC. This solves, in particular, the open question of decidability of emptiness for TAGED. We further extend our result by allowing global arithmetic constraints for counting the number of occurrences of some state or the number of different subterms reaching some state during a computation. We also allow local equality and disequality tests between sibling positions and the extension to unranked ordered trees. As a consequence of our results for TAGC, we prove the decidability of a fragment of the monadic second order logic on trees extended with predicates for equality and disequality between subtrees, and cardinality.

I. Introduction

Tree automata techniques are widely used in several domains like automated deduction (see *e.g.* [2]), static analysis of programs [3] or protocols [4], [5], and XML processing [6]. A severe limitation of standard tree automata (TA) is however that they are not able to test for equality (isomorphism) or disequality between subtrees in an input tree. For instance, the language of trees described by a non-linear pattern of the form $f(x, x)$ is not regular (*i.e.* there exists no TA recognizing this language). Similar problems are also frequent in the context of XML documents processing. XML documents are commonly represented as labeled trees, and they can be constrained by XML schemas, which define both typing restrictions and integrity constraints. All the typing formalisms currently used for XML are based on finite tree automata. The key constraints for databases are common integrity constraints expressing that every two distinct positions of a given type have different values. This is typically the kind of constraints that can not be characterized by TA.

One first approach to overcome this limitation of TA consists in adding the possibility to make equality or disequality tests at each step of the computation of the automaton. The tests are performed *locally*, between subtrees at a bounded distance from the current computation position in the input tree. The emptiness problem, whether the language recognized by a given automaton is empty, is undecidable with such tests [7]. A decidable subclass is obtained by restricting the tests to sibling subtrees [8] (see [2] for a survey).

Another approach was proposed more recently in [9], [1] with the definition of tree automata with *global* equality and disequality tests (TAGED). The TAGED do not perform the tests during the computation steps but globally on the tree, at the end of the computation, at positions which are defined by the states reached during the computation. For instance, they can express that all the subtrees that reached a given state q are equal, or that every two subtrees that reached respectively the states q and q' are different. The emptiness has been shown decidable for several subclasses of TAGED [9], [1], but the decidability of emptiness for the whole class remained a challenging open question.

In this paper, we answer this question positively, even for a class of tree recognizers more general than TAGED. We define (in Section II) a class of tree automata with global constraints (TAGC) which, roughly, corresponds to TAGED extended with the possibility to express disequalities between subtrees that reached the same state (specifying key constraints, which are not expressible with TAGEDs), and with arbitrary Boolean combinations (including negation) of constraints. We show in Section III that emptiness is decidable for TAGC. The decision algorithm uses an involved pumping argument: every sufficiently large tree recognized by the given TAGC can be reduced by an operation of parallel pumping into a smaller tree which is still recognized. The existence of the bound is based on a particular well quasi-ordering.

In Section IV-A, we study the extension of TAGC with global counting constraints on the number $|q|$ of occurrences of a given state q in a computation, or the number $\|q\|$ of distinct subtrees reaching a given state q in a computation. We show that emptiness is decidable for this extension when

The first three authors were supported by Spanish Ministry of Education and Science by the FORMALISM project (TIN2007-66523). The third author was supported by Spanish Ministry of Education and Science by LOGICTOOLS-2 project (TIN2007-68093-C02-01).

The last two authors were supported by the Future and Emerging Technologies (FET) program within the 7th Framework Program for Research of the European Commission, under the FET-Open grant agreement FOX number FP7-ICT-23359, and by the INRIA ARC 2010 project ACCESS.

counting constraints are only allowed to compare states to constants, like in $|q| \leq 5$ or $\|q\| + 2\|q'\| \geq 9$ (actually in this case, the counting constraints do not improve the expressiveness of TAGC). With counting constraints being able to compare state cardinalities (like in $|q| = |q'|$), emptiness becomes undecidable. We show that the emptiness decision algorithm can also be applied to the combination of TAGC with local tests between sibling subtrees a la [8] (Section IV-B), and to unranked ordered labeled trees (Section IV-C). This demonstrates the robustness of the method.

As an application of our results, in Section V we present a (strict) extension of the monadic second order logic on trees whose existential fragment corresponds exactly to TAGC. In particular, we conclude its decidability. The full version of this paper including all proofs can be found in [10].

Related Work.: The languages of TAGC and tree automata with local equality and disequality constraints are incomparable (see *e.g.* [11]). We show in Section IV-B that the local tests between sibling subtrees of [8] can be added to TAGC while preserving the decidability emptiness. The tree automata of [8] have been generalized from ranked trees to unranked ordered trees [12], [13]. The decidable generalization of TAGC to unranked ordered trees proposed in Section IV-C and the automata of [12], [13] are incomparable. A combination of both formalisms could be the object of a further study.

Another way to handle subtree equalities is to use automata computing on DAG representation of trees [14], [15]. This model is incomparable to TAGC whose constraints are conjunctions of equalities [11]. The decidable extension of TA with one tree shaped memory [16] can simulate TAGC with equality constraints only, providing that at most one state per run can be used to test equalities [9].

As explained in Section II-B, the TAGC strictly generalize the TAGEDs of [9], [1]. The latter have been introduced as a tool to decide a fragment of the spatial logic TQL [9]. Decidable subclasses of TAGEDs were also shown in correspondence with fragments of monadic second order logic on the tree extended with predicates for subtree (dis)equality tests. In Section V, we generalize this correspondence to TAGC and a more natural extension of MSO.

There have been several approaches to extend TA with arithmetic constraints on cardinalities $|q|$ described above: the constraints can be added to transitions in order to count between siblings [17], [18] (in this case we could call them *local* by analogy with equality tests) or they can be *global* [19]. We compare in Section IV-A the latter approach (closer to our settings) with our extension of TAGC, *wrt* emptiness decision. To our knowledge, this is the first time that arithmetic constraints on cardinalities of the form $\|q\|$ are studied.

II. Preliminaries

A. Terms, Positions, Tree Automata

We use the standard notations for terms and positions, see [20]. A *signature* Σ is a finite set of function symbols with arity. We sometimes denote Σ explicitly as $\{f_1 : a_1, \ldots, f_n : a_n\}$ where $f_1, \ldots, f_n$ are the function symbols, and $a_1, \ldots, a_n$ are the corresponding arities, or as $\{f_1, \ldots, f_n\}$ when the arities are omitted. We denote the subset of function symbols of Σ of arity m as Σ_m. The set of (ranked) *terms* over the signature Σ is defined recursively as $\mathcal{T}(\Sigma) := \{f \mid f : 0 \in \Sigma\} \cup \{f(t_1, \ldots, t_m) \mid f : m \in \Sigma, t_1, \ldots, t_m \in \mathcal{T}(\Sigma)\}$.

Positions in terms are denoted by sequences of natural numbers. With Λ we denote the empty sequence (root position), and $p.p'$ denotes the concatenation of positions p and p'. The set of positions of a term t is defined recursively as $Pos\big(f(t_1, \ldots, t_m)\big) = \{\Lambda\} \cup \{i.p \mid i \in \{1, \ldots, m\} \wedge p \in Pos(t_i)\}$. A term $t \in \mathcal{T}(\Sigma)$ can be seen as a function from its set of positions $Pos(t)$ into Σ. For this reason, the symbol labeling the position p in t shall be denoted by $t(p)$. By $p < p'$ and $p \leq p'$ we denote that p is a proper prefix of p', and that p is a prefix of p', respectively. In this cases, p' is necessarily of the form $p.p''$, and we define $p' - p$ as p''. Two positions p_1, p_2 incomparable with respect to the prefix ordering are called *parallel*, and it is denoted by $p_1 \parallel p_2$. The *subterm* of t at position p, denoted $t|_p$, is defined recursively as $t|_\Lambda = t$ and $f(t_1, \ldots, t_m)|_{i.p} = t_i|_p$. The replacement in t of the subterm at position p by s, denoted $t[s]_p$ is defined recursively as $t[s]_\Lambda = s$ and $f(t_1, \ldots, t_{i-1}, t_i, t_{i+1}, \ldots, t_m)[s]_{i.p} = f(t_1, \ldots, t_{i-1}, t_i[s]_p, t_{i+1}, \ldots, t_m)$. The fact $t = t[s]_p$ may also be used to emphasis that $t|_p$ is s. The *height* of a term t, denoted $h(t)$, is the maximal length of a position of $Pos(t)$. In particular, the length of Λ is 0.

A *tree automaton* (**TA**, see *e.g.* [2]) is a tuple $\mathcal{A} = \langle Q, \Sigma, F, \Delta \rangle$ where Q is a finite set of *states*, Σ is a signature, $F \subset Q$ is the subset of final states and Δ is a set of *transitions* rules of the form $f(q_1, \ldots, q_m) \to q$ where $f : m \in \Sigma$, $q_1, \ldots, q_m, q \in Q$. Sometimes, we shall refer to $\mathcal{A}$ as a subscript of its components, like in $Q_\mathcal{A}$ to indicate that Q is the state set of $\mathcal{A}$.

A *run* of $\mathcal{A}$ is a pair $r = \langle t, M \rangle$ where t is a term in $\mathcal{T}(\Sigma)$ and $M : Pos(t) \to Q_\mathcal{A}$ is a mapping satisfying, for all $p \in Pos(t)$, that the rule $t(p)\big(M(p.1), \ldots, M(p.m)\big) \to M(p)$ is in $\Delta_\mathcal{A}$, where m is the arity of the symbol $t(p)$ in Σ. By abuse of notation we write $r(p)$ for $M(p)$, and say that r is a run of $\mathcal{A}$ on t. Moreover, by $\texttt{term}(r)$ we refer to t, and by $\texttt{symbol}(r)$ we refer to $t(\Lambda)$. The run r is called *successful* (or *accepting*) if $r(\Lambda)$ is in $F_\mathcal{A}$. The language $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is the set of terms t for which there exists a successful run of $\mathcal{A}$. A language L is called *regular* if there exists a TA $\mathcal{A}$ satisfying $L = \mathcal{L}(\mathcal{A})$. For facility of explanations, we shall use term-like notations for runs defined as follows in the natural way. For a run $r = \langle t, M \rangle$, by $Pos(r)$ we denote $Pos(t)$, and by $h(r)$ we denote $h(t)$. Similarly, by $r|_p$ we denote the run $\langle t|_p, M|_p \rangle$, where $M|_p$ is defined as $M|_p(p') = M(p.p')$ for each p' in $Pos(t|_p)$, and say that $r|_p$ is a subrun of r. Moreover, for a run $r' = \langle t', M' \rangle$, by $r[r']_p$ we denote the run $\langle t[t']_p, M[M']_p \rangle$, where $M[M']_p$ is defined as $M[M']_p(p.p') = M'(p')$ for each p' in $Pos(t')$, and as $M[M']_p(p') = M(p')$ for each p' with $p \not\leq p'$.

A well quasi-ordering [21] $\leq$ on a set S is a reflexive and transitive relation such that any infinite sequence of elements $e_1, e_2, \ldots$ of S contains an increasing pair $e_i \leq e_j$ with $i < j$.

B. Tree Automata with Global Constraints

In this subsection, we define a class of tree automata with global constraints which generalizes the class of TAGEDs [1].

Definition II.1 *A* tree automaton with global constraints ***(TAGC)*** *over a signature Σ is a tuple $\mathcal{A} = \langle Q, \Sigma, F, C, \Delta \rangle$ such that $\langle Q, \Sigma, F, \Delta \rangle$ is a TA, denoted $ta(\mathcal{A})$, and C is a Boolean combination of atomic constraints of the form $q \approx q'$ or $q \not\approx q'$, where $q, q' \in Q$. A TAGC $\mathcal{A}$ is called* positive *if $C_{\mathcal{A}}$ is a disjunction of conjunctions of atomic constraints. A TAGC $\mathcal{A}$ is called* positive conjunctive *if $C_{\mathcal{A}}$ is a conjunction of atomic constraints. The subclasses of positive and positive conjunctive TAGC are denoted by* ***PTAGC*** *and* ***PCTAGC****, respectively.*

A run *r of the TAGC $\mathcal{A}$ is a run of $ta(\mathcal{A})$ such that r satisfies $C_{\mathcal{A}}$, denoted $r \models C_{\mathcal{A}}$, where the satisfiability of constraints is defined as follows, where t is* $\mathtt{term}(r)$*. For atomic constraints, $r \models q \approx q'$ holds (respectively $r \models q \not\approx q'$) if and only if for all different positions $p, p' \in Pos(t)$ such that $r(p) = q$ and $r(p') = q'$, $t|_p = t|_{p'}$ holds (respectively $t|_p \neq t|_{p'}$ holds). This notion of satisfiability is extended to Boolean combinations as usual. As for TAs, we say that r is a run of $\mathcal{A}$ on t.*

A run of $\mathcal{A}$ on $t \in \mathcal{T}(\Sigma)$ is successful *if $r(\Lambda) \in F_{\mathcal{A}}$. The language $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is the set of terms t for which there exists a successful run of $\mathcal{A}$.* $\diamondsuit$

It is important to note that the semantics of $\neg(q \approx q')$ and $q \not\approx q'$ differ, as well as the semantics of $\neg(q \not\approx q')$ and $q \approx q'$. This is because we have a "for all" quantifier in both definitions.

We use below the notation TAGC$[\tau]$, where τ is either $\approx$ or $\not\approx$, to characterise the subclass of TAGC with only atomic constraints of type τ (and same for PTAGC and PCTAGC).

The class of regular languages is strictly included in the class of TAGC languages due to the constraints.

Moreover, the TAGEDs of [1] are also a particular case of TAGC, since they can be redefined in our setting as restricted PCTAGC. In particular q and q' are required to be distinct in $q \not\approx q'$ for TAGEDs. Reflexive disequality constraints such as $q \not\approx q$ correspond to monadic *key constraints* for XML documents, meaning that every two distinct positions of type q have different values. A state q of a TAGC can be used for instance to characterize unique identifiers as in the following example, which presents a TAGC whose language cannot be recognized by a TAGED.

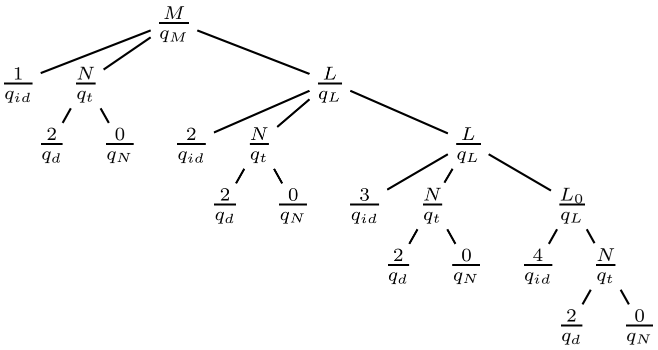

Figure 1. Term and successful run (Ex. II.2).

Example II.2 *The TAGC of our running example accepts (in state q_M) lists of dishes called menus, where every dish is associated with one identifier (state q_{id}) and the time needed to cook it (state q_t). We have other states accepting digits (q_d), numbers (q_N) and lists of dishes (q_L).*

The TAGC $\mathcal{A} = \langle Q, \Sigma, F, C, \Delta \rangle$ is defined as follows: $\Sigma = \{0, \ldots, 9 : 0, N, L_0 : 2, L, M : 3\}$, $Q = \{q_d, q_N, q_{id}, q_t, q_L, q_M\}$, $F = \{q_M\}$, and $\Delta = \{i \to q_d \mid q_N \mid q_{id} \mid q_t \mid 0 \leq i \leq 9\} \cup \{N(q_d, q_N) \to q_N \mid q_{id} \mid q_t, L_0(q_{id}, q_t) \to q_L, L(q_{id}, q_t, q_L) \to q_L, M(q_{id}, q_t, q_L) \to q_M\}$.

The constraint C ensures that all the identifiers of the dishes in a menu are pairwise distinct (i.e. that q_{id} is a key) and that the time to cook is the same for all dish: $C = q_{id} \not\approx q_{id} \wedge q_t \approx q_t$. Key constraints such as $q_{id} \not\approx q_{id}$ cannot be simulated by TAGEDs (see [10]).

A term in $\mathcal{L}(\mathcal{A})$ together with an associated successful run are depicted in Figure 1. $\diamondsuit$

Similarly to TAGED (see [1]) TAGC can be proved to be closed under union and intersection, but not under complementation. The membership problem (given a term t and a TAGC $\mathcal{A}$, do we have $t \in \mathcal{L}(\mathcal{A})$?) is NP-complete while universality (given a TAGC $\mathcal{A}$ over Σ, do we have $\mathcal{L}(\mathcal{A}) = \mathcal{T}(\Sigma)$?) is undecidable for PCTAGC$[\approx]$ [11]. The following consequence is a new result for TAGEDs.

Proposition II.3 *It is undecidable whether the language of a given PTAGC$[\approx]$ is regular.*

Proof: (Sketch) A reduction from universality to regularity can be easily described as follows by using a new function symbol f with arity 2, and any non-regular language L which is recognizable by a PTAGC[$\approx$].

Let $\mathcal{A}$ be an input of universality for PTAGC[$\approx$]. It is not difficult to compute a new PTAGC[$\approx$] $\mathcal{A}'$ recognizing the language $\{f(t_1, t_2) \mid t_1 \in \mathcal{T}(\Sigma) \wedge t_2 \in L\} \cup \{f(t_1, t_2) \mid t_1 \in \mathcal{L}(\mathcal{A}) \wedge t_2 \in \mathcal{T}(\Sigma)\}$. Then we can show that $\mathcal{L}(\mathcal{A}) = \mathcal{T}(\Sigma)$ if and only if $\mathcal{L}(\mathcal{A}')$ is regular. ■

The *emptiness* is the problem to decide, given a TAGC $\mathcal{A}$, whether $\mathcal{L}(\mathcal{A}) = \emptyset$? The proof that it is decidable for TAGC is rather involved and is presented in Section III.

III. Emptiness Decision Algorithm

In this section we prove the decidability of the *emptiness* problem for TAGC. We start by stating that it suffices to prove this result for PCTAGC.

Lemma III.1 *Given a TAGC $\mathcal{A}$, one can effectively construct a PCTAGC recognizing $\mathcal{L}(\mathcal{A})$.*

The proof of this lemma, given in [10], is technical but also straightforward. It is based on the fact that negative literals $\neg(q_1 \not\approx q_2)$ or $\neg(q_1 \approx q_2)$ can be encoded with the addition of new states and positive literals.

The decidability of emptiness for PCTAGC is proved in three steps. In Subsection III-A, we present a new notion of pumping which allows to transform a run into a smaller run under certain conditions. In Subsection III-B, we define a well quasi-ordering $\leq$ on a certain set S. In Subsection III-C, we connect the two previous subsections by describing how to compute, for each run r with height $h = h(r)$, a certain sequence $e_h, \ldots, e_0$ of elements of S satisfying the following fact: there exists a pumping on r if and only if $e_i \leq e_j$ for some $h \geq i > j \geq 0$. Finally, all of these constructions are used as follows. Suppose the existence of an accepting run r. If r is "too high", the fact that $\leq$ is a well quasi-ordering and the property of the sequence imply the existence of such i, j. Thus, it follows the existence of a pumping providing a smaller accepting run r'. We conclude the existence of a computational bound for the height of an accepting run, and hence, decidability of emptiness.

A. Global Pumpings

Pumping is a traditional concept in automata theory, and in particular, they are very useful to reason about tree automata. The basic idea is to convert a given run r into another run by replacing a subrun at a certain position p in r by a run r', thus obtaining a run $r[r']_p$. Pumpings are useful for deciding emptiness: if a "big" run can always be reduced by a pumping, then decision of emptiness is obtained by a search of an accepting "small" run.

For plain tree automata, a necessary and sufficient condition to ensure that $r[r']_p$ is a run is that the resulting states of $r|_p$ and r' coincide, since the correct application of a rule at a certain position depends only on the resulting states of the subruns of the direct children. In this case, an accepting run with height bounded by the number of states exists, whenever the accepted language is not empty.

i	H_i	$\check{H}_i$
5	$\{\Lambda\}$	$\emptyset$
4	$\{3\}$	$\{1, 2\}$
3	$\{3.3\}$	$\{1, 2, 3.1, 3.2\}$
2	$\{3.3.3\}$	$\{1, 2, 3.1, 3.2, 3.3.1, 3.3.2\}$
1	$\{2, 3.2, 3.3.2, 3.3.3.2\}$	$\{1, 3.1, 3.3.1, 3.3.3.1\}$
0	$\{1, 2.1, 2.2, 3.1, 3.2.1,$ $3.2.2, 3.3.1, 3.3.2.1, 3.3.2.2,$ $3.3.3.1, 3.3.3.2.1, 3.3.3.2.2\}$	$\emptyset$

Figure 2. H_i and $\check{H}_i$ (Example III.3).

When the tree automaton has global equality and disequality constraints, the constraints may be falsified when replacing a subrun by a new run. For PCTAGC, we will define a notion of pumping ensuring that the constraints are satisfied. This notion of pumping requires to perform several replacements in parallel. We first define the sets of positions involved in such a kind of pumping.

Definition III.2 *Let $\mathcal{A}$ be a PCTAGC. Let r be a run of $\mathcal{A}$. Let i be an integer between 0 and $h(r)$. We define H_i as $\{p \in Pos(r) \mid h(r|_p) = i\}$ and $\check{H}_i$ as $\{p.j \in Pos(r) \mid h(r|_{p.j}) < i \wedge h(r|_p) > i\}$.* ♢

Example III.3 *According to Definition III.2, for our running example (Example II.2), we have the H_i and $\check{H}_i$ presented in Figure 2.* ♢

The following lemma is rather straightforward from the previous definition.

Lemma III.4 *Let $\mathcal{A}$ be a PCTAGC. Let r be a run of $\mathcal{A}$. Let i be an integer between 0 and $h(r)$. Then, any two different positions in $H_i \cup \check{H}_i$ are parallel, and for any arbitrary position p in $Pos(r)$ there is a position $\bar{p}$ in $H_i \cup \check{H}_i$ such that, either p is a prefix of $\bar{p}$, or $\bar{p}$ is a prefix of p.*

Definition III.5 *Let $\mathcal{A}$ be a PCTAGC. Let r be a run of $\mathcal{A}$. Let i, j be integers satisfying $0 \leq j < i \leq h(r)$. A pump-injection $I : (H_i \cup \check{H}_i) \rightarrow (H_j \cup \check{H}_j)$ is an injection function such that the following conditions hold:*

(C_1) *$I(H_i) \subseteq H_j$ and $I(\check{H}_i) \subseteq \check{H}_j$.*

(C_2) *For each $\bar{p}$ in $H_i \cup \check{H}_i$, $r(\bar{p}) = r(I(\bar{p}))$.*

(C_3) *For each $\bar{p}_1, \bar{p}_2$ in $H_i \cup \check{H}_i$, $(\mathtt{term}(r|_{\bar{p}_1}) = \mathtt{term}(r|_{\bar{p}_2})) \Leftrightarrow (\mathtt{term}(r|_{I(\bar{p}_1)}) = \mathtt{term}(r|_{I(\bar{p}_2)}))$.*

Let $\{\bar{p}_1, \ldots, \bar{p}_n\}$ be $H_i \cup \check{H}_i$ more explicitly written. The run $r[r|_{I(\bar{p}_1)}]_{\bar{p}_1} \ldots [r|_{I(\bar{p}_n)}]_{\bar{p}_n}$ is called a global pumping *on r with indexes i, j, and injection I.* ♢

By Condition C_2, $r[r|_{I(\bar{p}_1)}]_{\bar{p}_1} \ldots [r|_{I(\bar{p}_n)}]_{\bar{p}_n}$ is clearly a run of $ta(\mathcal{A})$, but it is still necessary to prove that it

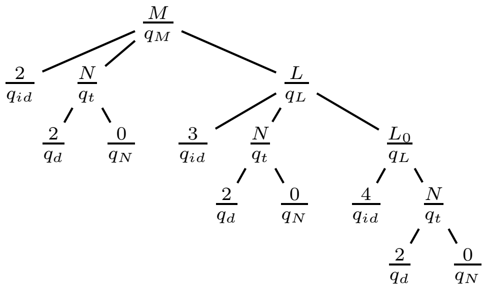

Figure 3. Pump-injection of Example III.6.

is a run of $\mathcal{A}$. By abuse of notation, when we write $r[r|_{I(\bar{p}_1)}]_{\bar{p}_1} \ldots [r|_{I(\bar{p}_n)}]_{\bar{p}_n}$, we sometimes consider that I and $\{\bar{p}_1, \ldots, \bar{p}_n\}$ are still explicit, and say that it is a global pumping with some indexes $0 \leq j < i \leq h(r)$.

Example III.6 *Following our running example, we define a pump-injection $I : (H_4 \cup \check{H}_4) \to (H_3 \cup \check{H}_3)$ as follows: $I(1) = 3.1, I(2) = 2, I(3) = 3.3$. We note that I is a correct pump-injection: $I(H_4) \subseteq H_3$ and $I(\check{H}_4) \subseteq \check{H}_3$ hold, thus (C_1) holds. For (C_2), we have $r(1) = r(I(1)) = q_{id}$, $r(2) = r(I(2)) = q_t$, and $r(3) = r(I(3)) = q_L$. Regarding (C_3), for each different $\bar{p}_1, \bar{p}_2$ in $H_4 \cup \check{H}_4$, $\mathtt{term}(r|_{\bar{p}_1}) \neq \mathtt{term}(r|_{\bar{p}_2})$ and $\mathtt{term}(r|_{I(\bar{p}_1)}) \neq \mathtt{term}(r|_{I(\bar{p}_2)})$ hold.*

After applying the pump-injection I, we obtain the term and run r' of Figure 3. ♢

Our goal is to prove that any global pumping $r[r|_{I(\bar{p}_1)}]_{\bar{p}_1} \ldots [r|_{I(\bar{p}_n)}]_{\bar{p}_n}$ is a run, and in particular, that all global equality and disequality constraints are satisfied. To this end we first state the following intermediate statement, which determines the height of the terms pending at some positions after the pumping action. It can be easily proved by induction on the height of the involved term.

Lemma III.7 *Let $\mathcal{A}$ be a PCTAGC. Let r be a run of $\mathcal{A}$. Let r' be the global pumping $r[r|_{I(\bar{p}_1)}]_{\bar{p}_1} \ldots [r|_{I(\bar{p}_n)}]_{\bar{p}_n}$ on r with indexes $0 \leq j < i \leq h(r)$ and injection I. Let $k \geq 0$ be a natural number and let p be a position of r such that $h(r|_p)$ is $i + k$.*
Then, p is also a position of r' and $h(r'|_p)$ is $j + k$.

Corollary III.8 *Let $\mathcal{A}$ be a PCTAGC. Let r be a run of $\mathcal{A}$. Let r' be a global pumping of r. Then, $h(r') < h(r)$.*

The following lemma states that equality and disequality relations are preserved, not only for terms pending at the positions of the domain of I, but also for terms pending at prefixes of positions of such domain. Again, it is rather easy to prove by induction on the height of the involved terms.

Lemma III.9 *Let $\mathcal{A}$ be a PCTAGC. Let r be a run of $\mathcal{A}$. Let r' be the global pumping $r[r|_{I(\bar{p}_1)}]_{\bar{p}_1} \ldots [r|_{I(\bar{p}_n)}]_{\bar{p}_n}$ with indexes $0 \leq j < i \leq h(r)$ and injection I. Let p_1, p_2 be positions of r satisfying $h(r|_{p_1}), h(r|_{p_2}) \geq i$.*

Then, p_1, p_2 are also positions of r' and $(\mathtt{term}(r|_{p_1}) = \mathtt{term}(r|_{p_2})) \Leftrightarrow (\mathtt{term}(r'|_{p_1}) = \mathtt{term}(r'|_{p_2}))$ holds.

As a consequence of the previous lemmas, we prove that the result of a global pumping is a run.

Lemma III.10 *Let $\mathcal{A}$ be a PCTAGC. Let r be a run of $\mathcal{A}$. Let r' be the global pumping $r[r|_{I(\bar{p}_1)}]_{\bar{p}_1} \ldots [r|_{I(\bar{p}_n)}]_{\bar{p}_n}$ with indexes $0 \leq j < i \leq h(r)$ and injection I.*

Then, r' is a run of $\mathcal{A}$.

Proof: By Condition (C_2) of the definition of pump-injection, in order to see that r' is a run, it suffices to see that all global constraints are satisfied. Thus, let us consider two different positions p_1, p_2 of $Pos(r')$ involved in an atom of the constraint of $\mathcal{A}$, *i.e.* either $r'(p_1) \approx r'(p_2)$ or $r'(p_1) \not\approx r'(p_2)$ occurs in the constraint of $\mathcal{A}$. According to Lemma III.4, we can distinguish the following cases:

- Suppose that a position in $H_i \cup \check{H}_i$, say $\bar{p}_1$, is a prefix of both p_1, p_2. Then, $r'|_{p_1} = r|_{I(\bar{p}_1).(p_1 - \bar{p}_1)}$ and $r'|_{p_2} = r|_{I(\bar{p}_1).(p_2 - \bar{p}_1)}$ hold. Hence, $r'|_{p_1}$ and $r'|_{p_2}$ are also subruns of r occurring at different positions. Thus, since r is a run, they satisfy the atom involving $r'(p_1)$ and $r'(p_2)$.
- Suppose that two different positions in $H_i \cup \check{H}_i$, say $\bar{p}_1$ and $\bar{p}_2$, are prefixes of p_1 and p_2, respectively. Then, $r'|_{p_1} = r|_{I(\bar{p}_1).(p_1 - \bar{p}_1)}$ and $r'|_{p_2} = r|_{I(\bar{p}_2).(p_2 - \bar{p}_2)}$ hold. By the injectivity of I, $I(\bar{p}_1) \neq I(\bar{p}_2)$ holds. Moreover, by Lemma III.4, $I(\bar{p}_1) \parallel I(\bar{p}_2)$ holds. Hence, as before, $r'|_{p_1}$ and $r'|_{p_2}$ are subruns of r occurring at different (in fact, parallel) positions. Thus, they satisfy the atom involving $r'(p_1)$ and $r'(p_2)$.
- Suppose that one of p_1, p_2, say p_1, is a proper prefix of a position in $H_i \cup \check{H}_i$, and that p_2 satisfies that some position in $H_i \cup \check{H}_i$ is a prefix of p_2. It follows that $h(r'|_{p_2})$ is smaller than or equal to j, and $r'|_{p_2}$ is also a subrun of r. Moreover, p_1 is also a position of r, $r'(p_1) = r(p_1)$ holds, and $h(r|_{p_1}) = i + k$ holds for some $k > 0$. Hence, $\mathtt{term}(r|_{p_1}) \neq \mathtt{term}(r'|_{p_2})$ holds. Since r is a run and $r'|_{p_2}$ is a subrun of r, the atom involving $r(p_1)$ and $r'(p_2)$ is necessarily of the form $r(p_1) \not\approx r'(p_2)$. Thus, the atom involving $r'(p_1)$ and $r'(p_2)$ is necessarily of the form $r'(p_1) \not\approx r'(p_2)$. By Lemma III.7, $h(r'|_{p_1})$ is $j + k$. Therefore, $\mathtt{term}(r'|_{p_1}) \neq \mathtt{term}(r'|_{p_2})$ holds, and hence, such an atom is satisfied for such positions in r'.
- Suppose that both p_1, p_2 are proper prefixes of positions in $H_i \cup \check{H}_i$. Then, p_1, p_2 are positions of r satisfying $h(r|_{p_1}), h(r|_{p_2}) \geq i$. Moreover, $r(p_1) = r'(p_1)$ and $r(p_2) = r'(p_2)$ hold. Since r is a run, the atom involving $r(p_1)$ and $r(p_2)$ is satisfied in the run r for positions p_1 and p_2. By Lemma III.9, $(\mathtt{term}(r|_{p_1}) = \mathtt{term}(r|_{p_2})) \Leftrightarrow (\mathtt{term}(r'|_{p_1}) = \mathtt{term}(r'|_{p_2}))$ holds. Thus, the atom involving $r'(p_1)$ and $r'(p_2)$ is satisfied in the run r' for positions p_1 and p_2. ■

B. A well quasi-ordering

In this subsection we define a well quasi-ordering. It assures the existence of a computational bound for certain sequences of elements of the corresponding well quasi-ordered set. It will be connected with global pumpings in the next subsection.

Definition III.11 *Let $\leq$ denote the usual quasi-ordering on natural numbers. Let n be a natural number.*

We define the extension of $\leq$ to n-tuples of natural numbers as $\langle x_1, \ldots, x_n \rangle \leq \langle y_1, \ldots, y_n \rangle$ if $x_i \leq y_i$ for each i in $\{1, \ldots, n\}$. We define $\mathtt{sum}(\langle x_1, \ldots, x_n \rangle) := x_1 + \cdots + x_n$.

We define the extension of $\leq$ to multisets of n-tuples of natural numbers as $[e_1, \ldots, e_\alpha] \leq [e'_1, \ldots, e'_\beta]$ if there is an injection $I : \{1, \ldots, \alpha\} \to \{1, \ldots, \beta\}$ satisfying $e_i \leq e'_{I(i)}$ for each i in $\{1, \ldots, \alpha\}$. We define $\mathtt{sum}([e_1, \ldots, e_\alpha]) := \mathtt{sum}(e_1) + \cdots + \mathtt{sum}(e_\alpha)$.

We define the extension of $\leq$ to pairs of multisets of n-tuples of natural numbers as $\langle P_1, \check{P}_1 \rangle \leq \langle P_2, \check{P}_2 \rangle$ if $P_1 \leq P_2$ and $\check{P}_1 \leq \check{P}_2$. $\Diamond$

As a direct consequence of Higman's Lemma [21] we have the following:

Lemma III.12 *Given n, $\leq$ is a well quasi-ordering for pairs of multisets of n-tuples of natural numbers.*

In any infinite sequence $e_1, e_2, \ldots$ of elements from a well quasi-ordered set there always exist two indexes $i < j$ satisfying $e_i \leq e_j$. In general, this fact does not imply the existence of a bound for the length of sequences without such indexes. For example, the relation $\leq$ between natural numbers is a well quasi-ordering, but there may exist arbitrarily long sequences $x_1, \ldots, x_k$ of natural numbers such that (‡) $x_i > x_j$ for all $1 \leq i < j \leq k$. In order to bound the length of sequences satisfying (‡), it is sufficient to force that the first element and each next element of the sequence are chosen among a finite number of possibilities. Indeed in this this case, by König's lemma, the prefix trees describing all such (finite) sequences is finite. As a particular case of this fact we have the following result (proved in [10]).

Lemma III.13 *There exists a computable function $B : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that, given two natural numbers a, n, $B(a, n)$ is a bound for the length ℓ of the maximum-length sequence $\langle T_1, \check{T}_1 \rangle, \ldots, \langle T_\ell, \check{T}_\ell \rangle$ of pairs of multisets of n-tuples of natural numbers such that the following conditions hold:*

1) *The tuple $\langle 0, \ldots, 0 \rangle$ does not occur in any T_i, $\check{T}_i$ for i in $\{1, \ldots, \ell\}$.*
2) $\mathtt{sum}(T_1) = 1$ *and* $\mathtt{sum}(\check{T}_1) = 0$.
3) *For each i in $\{1, \ldots, \ell - 1\}$, $\mathtt{sum}(T_{i+1}) + \mathtt{sum}(\check{T}_{i+1}) \leq a \cdot \mathtt{sum}(T_i) + \mathtt{sum}(\check{T}_i)$.*
4) *There are no i, j satisfying $1 \leq i < j \leq \ell$ and $\langle T_i, \check{T}_i \rangle \leq \langle T_j, \check{T}_j \rangle$*

i	r_{H_i}	$r_{\check{H}_i}$
5	$\big[\langle 0,0,0,0,0,1 \rangle\big]$	$[\ ]$
4	$\big[\langle 0,0,0,0,1,0 \rangle\big]$	$\big[\langle 0,0,1,0,0,0 \rangle, \langle 0,0,0,1,0,0 \rangle\big]$
3	$\big[\langle 0,0,0,0,1,0 \rangle\big]$	$\big[\langle 0,0,1,0,0,0 \rangle, \langle 0,0,0,2,0,0 \rangle,$ $\langle 0,0,1,0,0,0 \rangle\big]$
2	$\big[\langle 0,0,0,0,1,0 \rangle\big]$	$\big[\langle 0,0,1,0,0,0 \rangle, \langle 0,0,0,3,0,0 \rangle,$ $\langle 0,0,1,0,0,0 \rangle, \langle 0,0,1,0,0,0 \rangle\big]$
1	$\big[\langle 0,0,0,4,0,0 \rangle\big]$	$\big[\langle 0,0,1,0,0,0 \rangle, \langle 0,0,1,0,0,0 \rangle,$ $\langle 0,0,1,0,0,0 \rangle, \langle 0,0,1,0,0,0 \rangle\big]$
0	$\big[\langle 0,0,1,0,0,0 \rangle, \langle 4,0,1,0,0,0 \rangle,$ $\langle 0,4,0,0,0,0 \rangle, \langle 0,0,1,0,0,0 \rangle,$ $\langle 0,0,1,0,0,0 \rangle\ \big]$	$[\ ]$

Figure 4. Multisets r_{H_i}, $r_{\check{H}_i}$ (Example III.16).

In order to bound the height of a term accepted by a given PCTAGC $\mathcal{A}$ (and of minimum height), Lemma III.13 will be used by making a to be the maximum arity of the signature of $\mathcal{A}$, and making n to be the number of states of $\mathcal{A}$.

C. Mapping a run to a sequence of the well quasi-ordered set

We will associate, to each number i in $\{0, \ldots, h(r)\}$, a pair of multisets of tuples of natural numbers, which can be compared with other pairs according to the definition of $\leq$ in the previous subsection. To this end, we first associate tuples to terms and multisets of tuples to sets of positions.

Definition III.14 *Let $\mathcal{A}$ be a PCTAGC. Let $q_1, \ldots, q_n$ be the states of $\mathcal{A}$. Let r be a run of $\mathcal{A}$. Let P be a set of positions of r. Let t be a term. We define $r_{t,P}$ as the following tuple of natural numbers: $\big\langle \big|\{p \in P \mid \mathtt{term}(r|_p) = t \wedge r(p) = q_1\}\big|, \ldots, \big|\{p \in P \mid \mathtt{term}(r|_p) = t \wedge r(p) = q_n\}\big| \big\rangle$* $\Diamond$

Definition III.15 *Let $\mathcal{A}$ be a PCTAGC. Let r be a run of $\mathcal{A}$. Let P be a set of positions of r. Let $\{t_1, \ldots, t_k\}$ be the set of terms $\{t \mid \exists p \in P : \mathtt{term}(r|_p) = t\}$. We define r_P as the multiset $[r_{t_1,P}, \ldots, r_{t_k,P}]$.* $\Diamond$

Example III.16 *Following our running example, for the representation of the tuples of natural numbers we order the states as $\langle q_d, q_N, q_{id}, q_t, q_L, q_M \rangle$. The multisets r_{H_i} and $r_{\check{H}_i}$ are presented in Figure 4.* $\Diamond$

The following lemma connects the existence of a pump-injection with the quasi-ordering relation.

Lemma III.17 *Let $\mathcal{A}$ be a PCTAGC. Let r be a run of $\mathcal{A}$. Let i, j be integers satisfying $0 \leq j < i \leq h(r)$.*

Then, there exists a pump-injection $I : (H_i \cup \check{H}_i) \to (H_j \cup \check{H}_j)$ if and only if $\langle r_{H_i}, r_{\check{H}_i} \rangle \leq \langle r_{H_j}, r_{\check{H}_j} \rangle$.

Proof: We just prove the right-to-left direction: the other direction is technical but not conceptually difficult, and it is not necessary for the rest of the paper. Hence assume that $\langle r_{H_i}, r_{\check{H}_i}\rangle \leq \langle r_{H_j}, r_{\check{H}_j}\rangle$ holds. We have to construct a pump-injection $I : (H_i \cup \check{H}_i) \to (H_j \cup \check{H}_j)$. We just define $I : H_i \to H_j$ and prove Conditions (C_2) and (C_3) for $\bar{p}, \bar{p}_1, \bar{p}_2$ in H_i. This is because $I : \check{H}_i \to \check{H}_j$ can be defined analogously, and Conditions (C_2) and (C_3) for the corresponding positions can be checked analogously. Moreover, for positions $\bar{p}'_1 \in H_i$ and $\bar{p}'_2 \in \check{H}_i$, Condition (C_3) holds whenever Condition (C_1) holds since in this case $\texttt{term}(r|_{\bar{p}'_1}) \neq \texttt{term}(r|_{\bar{p}'_2})$ and $\texttt{term}(r|_{I(\bar{p}'_1)}) \neq \texttt{term}(r|_{I(\bar{p}'_2)})$ hold. Hence, this simple case is enough to prove the whole statement.

We write $\{\texttt{term}(r|_p) \mid p \in H_i\}$ and $\{\texttt{term}(r|_p) \mid p \in H_j\}$ more explicitly as $\{t_{i,1}, \ldots, t_{i,\alpha}\}$ and $\{t_{j,1}, \ldots, t_{j,\beta}\}$, respectively. Since $\langle r_{H_i}, r_{\check{H}_i}\rangle \leq \langle r_{H_j}, r_{\check{H}_j}\rangle$ holds, $r_{H_i} \leq r_{H_j}$ also holds. Thus, there exists an injective function $I' : \{1, \ldots, \alpha\} \to \{1, \ldots, \beta\}$ satisfying the following statement for each δ in $\{1, \ldots, \alpha\}$ and each state q of $\mathcal{A}$:
$\big|\{p \in H_i \mid \texttt{term}(r|_p) = t_{i,\delta} \wedge r(p) = q\}\big| \leq \big|\{p \in H_j \mid \texttt{term}(r|_p) = t_{j,I'(\delta)} \wedge r(p) = q\}\big| \quad (\dagger)$.

In order to define $I : H_i \to H_j$, we define I for each of such sets $\{p \in H_i \mid \texttt{term}(r|_p) = t_{i,\delta} \wedge r(p) = q\}$ as any injective function $I : \{p \in H_i \mid \texttt{term}(r|_p) = t_{i,\delta} \wedge r(p) = q\} \to \{p \in H_j \mid \texttt{term}(r|_p) = t_{j,I'(\delta)} \wedge r(p) = q\}$, which is possible by the above inequality (†). The global I is then injective thanks to the injectivity of I'. Conditions (C_2) and (C_3) trivially follow from this definition. ■

Example III.18 *Following our running example, we first prove $\langle r_{H_4}, r_{\check{H}_4}\rangle \leq \langle r_{H_3}, r_{\check{H}_3}\rangle$. To this end just note that $[\langle 0,0,0,0,1,0\rangle] \leq [\langle 0,0,0,0,1,0\rangle]$, $[\langle 0,0,1,0,0,0\rangle] \leq [\langle 0,0,1,0,0,0\rangle]$, and $[\langle 0,0,0,1,0,0\rangle] \leq [\langle 0,0,0,2,0,0\rangle]$ hold. We can define $I : (H_4 \cup \check{H}_4) \to (H_3 \cup \check{H}_3)$ from this relation according to Lemma III.17. Doing the adequate guess we obtain the following definition: $I(1) = 3.1$, $I(2) = 2$, $I(3) = 3.3$ which is the pump-injection considered above for our running example.* ♢

The following lemma follows directly from the definition of the sets H_i and $\check{H}_i$, and allows to connect such definitions with Lemma III.13.

Lemma III.19 *Let $\mathcal{A}$ be a PCTAGC. Let a be the maximum arity of the symbols in the signature of $\mathcal{A}$. Let r be a run of $\mathcal{A}$. Then, the following conditions hold:*

(1) $|H_{h(r)}| = 1$ *and* $|\check{H}_{h(r)}| = 0$.
(2) *For each i in $\{1, \ldots, h(r)\}$,* $|H_{i-1}| + |\check{H}_{i-1}| \leq a \cdot |H_i| + |\check{H}_i|$.
(3) *For each i in $\{0, \ldots, h(r)\}$,* $|H_i| = \texttt{sum}(r_{H_i})$ *and* $|\check{H}_i| = \texttt{sum}(r_{\check{H}_i})$.

Lemma III.20 *Let $B : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be the computable function of Lemma III.13. Let $\mathcal{A}$ be a PCTAGC. Let a be the maximum arity of the symbols in the signature of $\mathcal{A}$. Let n be the number of states of $\mathcal{A}$. Let r be a run of $\mathcal{A}$ satisfying $h(r) \geq B(a, n)$. Then, there is a global pumping on r.*

Proof: Consider the sequence $\langle r_{H_{h(r)}}, r_{\check{H}_{h(r)}}\rangle, \ldots, \langle r_{H_0}, r_{\check{H}_0}\rangle$. Note that the n-tuple $\langle 0, \ldots, 0\rangle$ does not appear in the multisets of the pairs of this sequence. By Lemma III.19, $|H_{h(r)}| = 1$ and $|\check{H}_{h(r)}| = 0$ hold, and for each i in $\{1, \ldots, h(r)\}$, $|H_{i-1}| + |\check{H}_{i-1}| \leq a \cdot |H_i| + |\check{H}_i|$ holds. Moreover, for each i in $\{0, \ldots, h(r)\}$, $|H_i| = \texttt{sum}(r_{H_i})$ and $|\check{H}_i| = \texttt{sum}(r_{\check{H}_i})$. Thus, $\texttt{sum}(r_{H_{h(r)}}) = 1$, $\texttt{sum}(r_{\check{H}_{h(r)}}) = 0$, and for each i in $\{1, \ldots, h(r)\}$, $\texttt{sum}(r_{H_{i-1}}) + \texttt{sum}(r_{\check{H}_{i-1}}) \leq a \cdot \texttt{sum}(r_{H_i}) + \texttt{sum}(r_{\check{H}_i})$. Hence, since $h(r) \geq B(a, n)$ holds, by Lemma III.13 there exist i, j satisfying $h(r) \geq i > j \geq 0$ and $\langle r_{H_i}, r_{\check{H}_i}\rangle \leq \langle r_{H_j}, r_{\check{H}_j}\rangle$. By Lemma III.17, There exists a pump-injection $I : (H_i \cup \check{H}_i) \to (H_j \cup \check{H}_j)$. Therefore, there exists a global pumping on r. ■

Theorem III.21 *Emptiness is decidable for PCTAGC.*

Proof: Let a be the maximum arity of the symbols in the signature of $\mathcal{A}$. Let n be the number of states of $\mathcal{A}$. Let r be an accepting run of $\mathcal{A}$ with minimum height.

Suppose that $h(r) \geq B(a, n)$ holds. Then, by Lemma III.20, there exists a global pumping r' on r. By Corollary III.8, $h(r') < h(r)$ holds. Moreover, by the definition of global pumping, $r'(\Lambda) = r(\Lambda)$ holds. Finally, by Lemma III.10, r' is a run of $\mathcal{A}$. Thus, r' contradicts the minimality of r. We conclude that $h(r) < B(a, n)$ holds. The decidability of emptiness of $\mathcal{A}$ follows, since the existence of successful runs implies that one of them can be found among a computable and finite set of possibilities. ■

Using Lemma III.1 and Theorem III.21, we can conclude the decidability of emptiness for TAGC.

Corollary III.22 *Emptiness is decidable for TAGC.*

IV. Extensions

In this section, we extend the emptiness result by considering the addition of new constraints to TAGC. We shall extend the notation TAGC$[\tau_1, \ldots, \tau_k]$ introduced in Section to these new type of constraints, in addition to $\approx$ and $\not\approx$.

A. Arithmetic Constraints

We study first the addition of counting constraints to TAGC. Let Q be a set of states. A *linear inequality* over Q is an expression of the form $\sum_{q \in Q} a_q.|q| \geq a$ or $\sum_{q \in Q} a_q.\|q\| \geq a$ where every a_q and a belong to $\mathbb{Z}$.
Let r be a run on a term t of a TA or TAGC $\mathcal{A}$ over Σ and with state set Q, and let $q \in Q$. The interpretations of $|q|$ and $\|q\|$ *wrt* r (and t) are defined respectively by the following cardinalities $[\![\, |q| \,]\!]_r = |r^{-1}(q)|$ and $[\![\, \|q\| \,]\!]_r = \big|\{s \in \mathcal{T}(\Sigma) \mid \exists p \in Pos(t), r(p) = q, s = t|_p\}\big|$.
This permits to define the satisfiability of linear equalities *wrt* t and r and the notion of successful runs for extensions of TAGC with atomic constraints which can have the form of the above linear inequalities.

Let us denote by $|.|_{\mathbb{Z}}$ and $\|.\|_{\mathbb{Z}}$ the types of the above linear inequalities, seen as atomic constraints of TAGC. The class TAGC$[\,|.|_{\mathbb{Z}}]$ has been studied under different names (*e.g.* Parikh automata in [19], linear constraint tree automata in [22]) and it has a decidable emptiness test.

Combining constraints of type $\approx$ and counting constraints of type $|.|_{\mathbb{Z}}$ however leads to undecidability.

Theorem IV.1 *Emptiness is undecidable for PTAGC*$[\approx, |.|_{\mathbb{Z}}]$.

Proof: (Sketch) The proof is based on a reduction from the Hilbert's tenth problem. The main point is the fact that we can encode (non negative) integer multiplication by combining $\approx$ and $|.|_{\mathbb{Z}}$. We can indeed build a PTAGC$[\approx, |.|_{\mathbb{Z}}]$ $\mathcal{A}_*$ recognizing terms of the form $*(t_1, t_2, t_{12})$ such that in a run of $\mathcal{A}_*$, all the leaves of t_1, t_2 and t_{12} respectively are labeled by the state q_1, q_2 and q_{12}. Moreover, t_{12} is of the form $f(t_1, \ldots, f(t_1, b))$, and in a run of $\mathcal{A}_*$, the position of the first t_1 (first argument of $*$) is labelled by a state r_1 and the positions of all the t_1's in t_{12} are labeled by r'_1. The constraint $r_1 \approx r'_1 \wedge |r'_1| = |q_2|$ ensures that t_{12} contains $|q_2|$ occurrences of t_1, hence that $|q_{12}| = |q_1| * |q_2|$. ■

We present now a restriction on linear equalities which enables a decidable emptiness test when combined with $\approx$ and $\not\approx$ as global constraints. A *natural linear inequality* over Q is a linear inequality as above whose coefficients a_q and a all have the same sign. The types of the natural linear inequalities are denoted by $|.|_{\mathbb{N}}$ and $\|.\|_{\mathbb{N}}$. Below, we shall abbreviate these two types by $\mathbb{N}$.

We show in [10] that the class PTAGC$[\approx, \not\approx, \mathbb{N}]$ has the same expressiveness as TAGC$[\approx, \not\approx]$. The proofs works in two steps, using the intermediate class (equally expressive) PTAGC$[\approx, \not\approx, \mathbb{N}]$. We add new states and replace: in the first step, the negative constraints by counting constraints and positive constraints (of type $\approx$ and $\not\approx$), and in the second step, the counting constraints by positive constraints.

Both Lemma III.1 and the next theorem follow immediately from this result.

Theorem IV.2 *Emptiness is decidable for TAGC*$[\approx, \not\approx, \mathbb{N}]$.

B. Equality Tests Between Brothers

The constraints of TAGC are checked once for all on a whole run. There exists another kind of equality and disequality constraints for extending TA which are tested locally at every transition step. One example of TA with such local constraints defined in [8] are tree automata with constraints between brothers (**TACB**).

A TACB is a tuple $\mathcal{A} = \langle Q, \Sigma, F, \Delta \rangle$ where Q, F, Σ are defined as with TA and the transitions rules of Δ have the form: $f(q_1, \ldots, q_n) \xrightarrow{c} q$, where c is a conjunction of atoms of the form $i = j$ or $i \neq j$ with $1 \leq i, j \leq n$. A *run* of the TACB $\mathcal{A}$ on a term $t \in \mathcal{T}(\Sigma)$ is a function r from $Pos(t)$ into Q such that, for all $p \in Pos(t)$, there exists a rule $t(p)\big(r(p.1), \ldots, r(p.m)\big) \xrightarrow{c} r(p) \in \Delta$ satisfying $t(p) \in \Sigma_m$, and moreover, for all $i = j$ in the conjunction c, $t|_{p.i} = t|_{p.j}$ holds, and for all $i \neq j$ in the conjunction c, $t|_{p.i} \neq t|_{p.j}$ holds.

The notions of successful runs and languages can be extended straightforwardly from TA to TACB. Global constraints can also be added to TACB in the natural way. The automata of the resulting classes TACB$[\approx, \not\approx, ...]$ will therefore perform global and local test during their computations. The emptiness decision algorithm of Section III still works for this extension of TAGC with local brother constraints. This is because a global pumping $r[r|_{I(\bar{p}_1)}]_{\bar{p}_1} \ldots [r|_{I(\bar{p}_n)}]_{\bar{p}_n}$ on a run r can be proved to satisfy the constraints between brothers in a completely analogous way as in the proof of Lemma III.10 and Theorem IV.2.

Theorem IV.3 *Emptiness is decidable for TACB*$[\approx, \not\approx, \mathbb{N}]$.

C. Unranked Ordered Trees

Our tree automata models and results can be generalized from ranked to unranked ordered terms. In this setting, Σ is called an *unranked signature*, meaning that there is no arity fixed for its symbols, *i.e.* that in a term $a(t_1, \ldots, t_n)$, the number n of children is arbitrary and does not depend on a. Let us denote by $\mathcal{U}(\Sigma)$ the set of unranked ordered terms over Σ. The notions of positions, subterms *etc* are defined for unranked terms of $\mathcal{U}(\Sigma)$ as for ranked terms of $\mathcal{T}(\Sigma)$.

We extend the definition of automata for unranked ordered terms, called hedge automata [23], with global constraints. A *hedge automaton with global constraints* (**HAGC**) is a tuple $\mathcal{A} = \langle Q, \Sigma, F, C, \Delta \rangle$ where Q, F and C are as in the definition of TAGC in Section II-B and the transitions of Δ have the form $a(L) \to q$ where $a \in \Sigma$, $q \in Q$ and L is a regular (word) language over Q^*, assumed given by a NFA with input alphabet Q. The notion of run is extended to HAGC in the natural way.

The emptiness decision results of Corollary III.22 can be extended from TAGC to HAGC using a standard transformation from unranked to ranked binary terms, like the *extension* encoding described in [2], Chapter 8.

Theorem IV.4 *Emptiness is decidable for HAGC*$[\approx, \not\approx, \mathbb{N}]$.

V. Monadic Second Order Logic

A ranked term $t \in \mathcal{T}(\Sigma)$ over Σ can be seen as a model for logical formulae, with an interpretation domain which is the set of positions $Pos(t)$. We consider monadic second order formulae interpreted on such models, with quantifications over first order variables (interpreted as positions), denoted $x, y \ldots$ and over unary predicates (*i.e.* set variables interpreted as sets of positions), denoted $X, Y \ldots$

The formulae are built with the following predicates:

1) equality $x = y$, and membership $X(x)$ – the position x belongs to the set X,

2) $a(x)$, for $a \in \Sigma$ – the position x is labeled by a in t,
3) $S_i(x, y)$, for all i smaller than or equal to the maximal arity of symbols of Σ, which is true on every pair of positions $(p, p.i)$, (we call $+1$ the type of such predicates),
4) term equality $X \approx Y$, resp. disequality $X \not\approx Y$, which is true when for all x in X and all y in Y, $t|_x = t|_y$, resp. $t|_x \neq t|_y$, (predicate types $\approx$ and $\not\approx$),
5) linear inequalities $\sum a_i.|X_i| \geq a$ or $\sum a_i.\|X_i\| \geq a$, where every a_i and a belong to $\mathbb{Z}$; $|X_i|$ is interpreted as the cardinality of X_i and $\|X_i\|$ as the cardinality of $\{t|_p \mid p \in X_i\}$. (predicate types $|.|_\mathbb{Z}$ and $\|.\|_\mathbb{Z}$).

We write MSO$[\tau_1, \ldots, \tau_k]$ for the set of monadic second order logic formulae with equality, membership, labeling predicates and other predicates of type $\tau_1, \ldots, \tau_k$, amongst the above types $+1$, $\approx$, $\not\approx$, and $|.|_\mathbb{Z}$, $\|.\|_\mathbb{Z}$. We also use the notations $|.|_\mathbb{N}$ and $\|.\|_\mathbb{N}$ for natural linear inequalities and the abbreviations $\mathbb{Z}$ and $\mathbb{N}$ as in Section IV-A. Let EMSO$[\overline{\tau}]$ be the fragment of MSO$[\overline{\tau}]$ of the formulae of the form $\exists X_1 \ldots \exists X_n\, \phi$ where all the set variables in ϕ belong to $\{X_1, \ldots, X_n\}$.

It is well known that MSO$[+1]$ has exactly the same expressiveness as TA [24] and therefore it is decidable. The extension MSO$[+1, \approx]$ is undecidable, see *e.g.* [9], as well as MSO$[+1, |.|_\mathbb{Z}]$ [19] (the latter extension is also undecidable for unranked ordered terms when counting constraints are applied to sibling positions [17]), but the fragment EMSO$[+1, |.|_\mathbb{Z}]$ is decidable [19].

In [1] the fragment EMSO with $\approx$ and a restricted form of $\not\approx$ is shown decidable, with a two way correspondence between these formulae and a decidable subclass of TAGEDs. This construction can be straightforwardly adapted to establish a two way correspondence between EMSO$[+1, \approx, \not\approx, \mathbb{N}]$ and TAGC$[\approx, \not\approx, \mathbb{N}]$. With Theorem IV.2, it gives the following result.

Theorem V.1 *EMSO$[+1, \approx, \not\approx, \mathbb{N}]$ is decidable on ranked terms.*

Unranked Ordered Terms.: In unranked ordered terms, the number of child of a position is unbounded. Therefore, for navigating in such terms with logical formulae, the successor predicates of category 3 above are not sufficient. In order to describe unranked ordered terms as models, we replace the above predicates S_i by: $S_\downarrow(x, y)$ which holds on every pair of positions of the form $(p, p.i)$ (*i.e.* y is a child of x), $S_\rightarrow(x, y)$ which holds on every pair of positions of the form $(p.i, p.i + 1)$ (*i.e.* y is the successor sibling of x). The type of these predicate is still called $+1$. Note that the above predicates $S_1, S_2, \ldots$ can be expressed using these two predicates only.

Using the results of Section IV-C, we can generalize Theorem V.1 to EMSO over unranked ordered terms.

Theorem V.2 *EMSO$[+1, \approx, \not\approx, \mathbb{N}]$ is decidable on unranked ordered terms.*

VI. Conclusion

We have answered (positively) the open problem of decidability of the emptiness problem for the TAGEDs [1], by proposing a decision algorithm for a class TAGC of tree automata with global constraints extending TAGEDs. Our method for emptiness decision, presented in Section III appeared to be robust enough to deal with several extensions like global counting constraints, local equality tests between sibling subterms and extension to unranked terms. It could perhaps be extended to equality modulo commutativity and other equivalence relations. Another interesting subject mentioned in the introduction is the combination of the HAGC of Section IV-C with the unranked tree automata with tests between siblings [12], [13].

A challenging question would be to investigate the precise complexity of the problem, avoiding the use of Higman's Lemma in the algorithm. For instance, in [1], it is shown, using a direct reduction into solving positive and negative set constraints [25], [26], that emptiness is decidable in NEXPTIME for PCTAGC$[\not\approx]$ such that in every atomic constraints $q \not\approx q'$, q and q' are distinct states. On the other hand, the best known lower bound for emptiness decision for TAGC is EXPTIME-hardness (this holds already for PCTAGC$[\approx]$ as shown in [1]).

Another branch of research related to TAGC concerns automata and logics for *data trees*, *i.e.* trees labeled over an infinite (countable) alphabet (see [27] for a survey). Indeed, data trees can be represented by terms over a finite alphabet, with an encoding of the data values into terms. This can be done in several ways, and with such encodings, the data equality relation becomes the equality between subterms. Therefore, this could be worth studying in order to relate our results on TAGC to decidability results on automata or logics on data trees like those in [28], [22].

Acknowledgements

We thank Luc Segoufin for valuable discussions and the anonymous referees for their numerous comments and suggestions.

References

[1] E. Filiot, J.-M. Talbot, and S. Tison, "Tree Automata with Global Constraints," in *12th International Conference in Developments in Language Theory (DLT 2008)*, ser. Lecture Notes in Computer Science, vol. 5257. Springer, 2008, pp. 314–326.

[2] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, C. Löding, D. Lugiez, S. Tison, and M. Tommasi, *Tree Automata Techniques and Applications*. http://tata.gforge.inria.fr, 2007.

[3] A. Bouajjani and T. Touili, “On computing reachability sets of process rewrite systems,” in *Proceedings of the 16th International Conference on Term Rewriting and Applications (RTA 2005)*, ser. Lecture Notes in Computer Science, vol. 3467. Springer, 2005, pp. 484–499.

[4] K. N. Verma and J. Goubault-Larrecq, “Alternating Two-Way AC-Tree Automata,” *Inf. Comput.*, vol. 205, no. 6, pp. 817–869, 2007.

[5] G. Feuillade, T. Genet, and V. Viet Triem Tong, “Reachability Analysis over Term Rewriting Systems,” *Journal of Automated Reasoning*, vol. 33 (3-4), pp. 341–383, 2004.

[6] T. Schwentick, “Automata for XML - A Survey,” *J. Comput. Syst. Sci.*, vol. 73, no. 3, pp. 289–315, 2007.

[7] J. Mongy, “Transformation de noyaux reconnaissables d’arbres. forêts rateg,” Ph.D., Laboratoire d’Informatique Fondamentale de Lille, Université des Sciences et Technologies de Lille, Villeneuve d’Ascq, France, 1981.

[8] B. Bogaert and S. Tison, “Equality and Disequality Constraints on Direct Subterms in Tree Automata,” in *9th Symp. on Theoretical Aspects of Computer Science (STACS 1992)*, ser. LNCS, vol. 577. Springer, 1992, pp. 161–171.

[9] E. Filiot, J.-M. Talbot, and S. Tison, “Satisfiability of a Spatial Logic with Tree Variables,” in *Proceedings of the 21st International Workshop on Computer Science Logic (CSL 2007)*, ser. Lecture Notes in Computer Science, vol. 4646. Springer, 2007, pp. 130–145.

[10] L. Barguñó, C. Creus, G. Godoy, F. Jacquemard, and C. Vacher, “The Emptiness Problem for Tree Automata with Global Constraints,” available at `www.lsi.upc.edu/˜ggodoy/publications.html`.

[11] F. Jacquemard, F. Klay, and C. Vacher, “Rigid tree automata,” in *Proceedings of the 3rd International Conference on Language and Automata Theory and Applications (LATA 2009)*, ser. Lecture Notes in Computer Science, vol. 5457. Springer, 2009, pp. 446–457.

[12] K. Wong and C. Löding, “Unranked Tree Automata with Sibling Equalities and Disequalities,” in *In Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP 2007)*, ser. LNCS, vol. 4596. Springer, 2007, pp. 875–887.

[13] C. Löding and K. Wong, “On Nondeterministic Unranked Tree Automata with Sibling Constraints,” in *In IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2009)*, ser. Leibniz International Proceedings in Informatics. Schloss Dagstuhl - Leibniz Center for Informatics, 2009.

[14] W. Charatonik, “Automata on DAG Representations of Finite Trees,” Max-Planck-Institut für Informatik, Saarbrücken, Germany, Tech. Rep. MPI-I-99-2-001, 1999.

[15] S. Anantharaman, P. Narendran, and M. Rusinowitch, “Closure Properties and Decision Problems of DAG Automata,” *Inf. Process. Letters*, vol. 94, no. 5, pp. 231–240, 2005.

[16] H. Comon and V. Cortier, “Tree Automata with one Memory, Set Constraints and Cryptographic Protocols,” *Theoretical Computer Science*, vol. 331, no. 1, pp. 143–214, Feb. 2005. [Online]. Available: http://www.lsv.ens-cachan.fr/Publis/PAPERS/PS/ComonCortierTCS1.ps

[17] H. Seidl, T. Schwentick, and A. Muscholl, “Numerical Document Queries,” in *Principle of Databases Systems (PODS 2003)*. ACM Press, 2003, pp. 155–166.

[18] S. Dal Zilio and D. Lugiez, “XML Schema, Tree Logic and Sheaves Automata,” *Journal Applicable Algebra in Engineering, Communication and Computing*, vol. 17, no. 5, pp. 337–377, 2006.

[19] F. Klaedtke and H. Ruess, “Parikh Automata and Monadic Second-Order Logics with Linear Cardinality Constraints”, Intitute of Computer Science at Freiburg University, Tech. Rep. 177, 2002.

[20] F. Baader and T. Nipkow, *Term Rewriting and All That*. New York: Cambridge University Press, 1998.

[21] J. H. Gallier, “What’s so Special about Kruskal’s Theorem and the Ordinal Γ_0? A Survey of Some Results in Proof Theory,” *Annals of Pure Applied Logic*, vol. 53, no. 3, pp. 199–260, 1991.

[22] M. Bojanczyk, A. Muscholl, T. Schwentick, and L. Segoufin, “Two-Variable Logic on Data Trees and Applications to XML Reasoning,” *JACM*, vol. 56, no. 3, 2009, a preliminary version was presented at PODS 2006.

[23] M. Murata, “Hedge Automata: a Formal Model for XML Schemata,” Fuji Xerox INformation Systems, Tech. Rep., 1999.

[24] J. W. Thatcher and J. B. Wright, “Generalized Finite Automata Theory with an Application to a Decision Problem of Second-Order Logic,” *Mathematical System Theory*, vol. 2, pp. 57–82, 1968.

[25] W. Charatonik and L. Pacholski, “Set Constraints with Projections are in NEXPTIME,” in *Proceedings of the 35^{th} Symp. Foundations of Computer Science*, 1994, pp. 642–653.

[26] R. Gilleron, S. Tison, and M. Tommasi, “Some New Decidability Results on Positive and Negative Set Constraints.” in *Proceedings, First International Conference on Constraints in Computational Logics*, ser. LNCS, vol. 845. Spinger, 1994, pp. 336–351.

[27] L. Segoufin, “Automata and Logics for Words and Trees over an Infinite Alphabet,” in *Computer Science Logic*, ser. LNCS, vol. 4207. Springer, 2006.

[28] M. Jurdzinski and R. Lazic, “Alternation-Free Modal mu-Calculus for Data Trees,” in *Proceedings 22nd IEEE Symposium on Logic in Computer Science (LICS)*. IEEE Computer Society, 2007, pp. 131–140.