The Emptiness Problem for Tree Automata with Global Constraints

Luis Barguñó Carles Creus Guillem Godoy
Universitat Politècnica de Catalunya
Barcelona

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 well-known class of tree automata with global equality and disequality constraints [14] (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.

1 Introduction

Tree automata techniques are widely used in several domains like automated deduction (see *e.g.* [10]), static analysis of programs [6] or protocols [28, 12], and XML processing [23]. 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 [21]. A decidable subclass is obtained by restricting the tests to sibling subtrees [4] (see [10] for a survey).

Another approach was proposed more recently in [13, 14] 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 a 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 [13, 14], 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 2) 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 3 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 par-

ticular well quasi-ordering.

In Section 4.1, 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 subtree reaching a given state q in a computation. We show that emptiness is decidable for this extension when 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 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 [4] (Section 4.2), and to unranked ordered labeled trees (Section 4.3). This demonstrates the robustness of the method.

As an application of our results, in Section 5 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 [3].

Related Work. The languages of TAGC and tree automata with local equality and disequality constraints are incomparable (see *e.g.* [17]). We show in Section 4.2 that the local tests between sibling subtrees of [4] can be added to TAGC while preserving the decidability emptiness. The tree automata of [4] have been generalized from ranked trees to unranked ordered trees [29, 20]. In unranked trees, the number of brothers (under a position) is unbounded, and UTASC transitions use MSO formulae (on words) with 2 free variables in order to select the sibling positions to be tested for equality and disequality. The decidable generalization of TAGC to unranked ordered trees proposed in Section 4.3 and the automata of [29, 20] 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's representation of trees [7, 1]. This model is incomparable to TAGC whose constraints are conjunctions of equalities [17]. The decidable extension of TA with one tree shaped memory [9] can simulate TAGC with equality constraints only, providing that at most one state per run can be used to test equalities, see [13].

As explained in Section 2.2, the TAGC strictly generalize the TAGEDs of [13, 14]. The latter have been introduced as a tool to decide a fragment of the spatial logic TQL [13]. 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 5, 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 [25, 11] (in this case we could call them *local* by analogy with equality tests) or they can be *global* [19]. We compare in Section 4.1 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.

2 Preliminaries

2.1 Terms, Positions, Tree Automata

We use the standard notations for terms and positions, see [2]. 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.* [10]) 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 $\mathtt{term}(r)$ we refer to t, and by $\mathtt{symbol}(r)$ 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' holding $p \not\leq p'$.

A well quasi-ordering [15] $\leq$ 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 $i < j$.

2.2 Tree Automata with Global Constraints

In this subsection, we define a class of tree automata with global constraints which generalizes the class of TAGEDs [14].

Definition 2.1 *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 TAGC 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 $\mathcal{A}$.* $\Diamond$

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.

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

Example 2.2 *Let $\Sigma = \{a : 0, f : 2\}$. The set $\{f(t,t) \mid t \in \mathcal{T}(\Sigma)\}$ is not a regular tree language (this can be shown using a classical* pumping *argument).*

However, it is recognized by the following TAGC $\mathcal{A} = \big\langle\{q_0, q_1, q_{\mathsf{f}}\}, \Sigma, \{q_{\mathsf{f}}\}, q_1 \approx q_1, \{a \to q_0|q_1, f(q_0, q_0) \to q_0|q_1, f(q_1, q_1) \to q_{\mathsf{f}}\}\big\rangle$, where $a \to q|q_r$ is an abbreviation for $a \to q$ and $a \to q_r$. An example of successful run of $\mathcal{A}$ on $t = f(f(a,a), f(a,a))$ is $q_{\mathsf{f}}\big(q_1(q_0,q_0), q_1(q_0,q_0)\big)$. $\Diamond$

Moreover, the TAGEDs of [14] are also a particular case of TAGC, since they can be redefined in our setting as TAGC whose constraints are conjunctions of atoms $q \approx q'$ and $q \not\approx q'$, with additional restrictions. 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, like in the following example which presents a TAGC whose language cannot be recognized by a TAGED.

Example 2.3 *Let $\Sigma = \{0:0, s:1, f:2\}$ and let L be the set of terms of $\mathcal{T}(\Sigma)$ of the form $f(s^{n_1}(0), \ldots, f(s^{n_k}(0), 0))$, such that $k \geq 0$ and the integers n_i, for $i \leq k$, are pairwise distinct. It is recognized by following the TAGC $\big\langle\{q_0, q, q_{\mathsf{f}}\}, \Sigma, \{q_{\mathsf{f}}\}, q \not\approx q, \{0 \to q_0|q|q_{\mathsf{f}}, s(q_0) \to q_0|q, f(q, q_{\mathsf{f}}) \to q_{\mathsf{f}}\}\big\rangle$. However, L cannot be recognized by a PTAGC without a reflexive constraint of the form $q \not\approx q$.*

Assume on the contrary that there is a PTAGC $\mathcal{A}$ without such a constraint, i.e. a TAGED, that recognizes this language. There exists an accepting run r of $\mathcal{A}$ on the term $t = f(s(0), f(s^2(0), \ldots f(s^{|Q|+1}(0))))$. We have therefore $r \models C_{\mathcal{A}}$ (the global constraint of $\mathcal{A}$, which is positive by hypothesis) and let C_1 be the conjunction of all the atomic constraints occurring in $C_{\mathcal{A}}$ and satisfied by r.

There are two different positions $p_i = 2^{i-1}.1$ and $p_j = 2^{j-1}.1$, $1 \leq i < j \leq |Q| + 1$ such that $r(p_i) = r(p_j)$. Let us show that $r' = r[r|_{p_i}]_{p_j}$ is an accepting run of $\mathcal{A}$ on $t' = t[t|_{p_i}]_{p_j}$. Since $r(p_i) = r(p_j)$ and r is a run of $\mathcal{A}$ on t, by replacing $t|_{p_j}$ with $t|_{p_i}$ and $r|_{p_j}$ with $r|_{p_i}$, r' is a run of $ta(\mathcal{A})$ on t. Hence we only have to ensure that the constraint $C_{\mathcal{A}}$ is fulfilled by r'.

For all $p, p' \in Pos(t')$ such that $2^{j-1}.1$ is neither a prefix of p nor of p', and such that p and p' are no prefix of

$2^{j-1}.1$, *if* $r'(p) \approx r'(p')$ *is in* C_1 *(resp.* $r'(p) \not\approx r'(p')$ *is in* C_1*), we know that* $r \models r(p) \approx r(p')$ *(resp.* $r \models r(p) \not\approx r(p')$*), so the constraints are respected in* t*, hence also in* t' *since the positions* p *and* p' *are referring to common subterms of* t *and* t'*.*

If a position $p = 2^{j-1}.1.v$ *is involved in some constraint, let* $p' = 2^{i-1}.1.v$*. By construction, we have* $r'|_p = r'|_{p'}$ *and* $t'|_p = t'|_{p'}$*. Due to this last equality, any constraint involving* p *is satisfied iff the same constraint with* p' *instead of* p *also holds. And thanks to the equality* $r'|_p = r'|_{p'}$*, this other constraint holds and is satisfied by* r'*.*

Finally, we have to consider constraints that involve a strict prefix p *of* $2^{j-1}.1$*. It is clear that every subterm of* t *at a position* 2^ℓ*, for* $0 \leq \ell \leq |Q|$*, is unique, so every subterm at such a position can only satisfy a disequality constraint or an equality with itself (in that case* $r(2^\ell)$ *is unique in* r*). In the latter case,* $r'(p)$ *is also unique in* r' *and the equality is obviously satisfied. In the other case, it is easy to see that it also holds in* t' *that all subterms at positions* 2^ℓ*, and hence the subterm at position* p*, are unique, and satisfy all the disequalities. So* t' *is recognized by* $\mathcal{A}$ *but is not in the language, a contradiction. It follows that TAGC are strictly more expressive than TAGED.* ◊

This example will be referred several times in the following section, in order to illustrate the definitions used in the decision procedure of the emptiness problem for TAGC.

Example 2.4 *The following running example represents a Menu where, for each dish, we have an identifier (*q_{id}*) and the time needed to cook that dish (*q_t*). Our menu will always have a special dish and a list of extra dishes. We have other states representing digits (*q_d*), numbers (*q_N*) and lists of dishes (*q_L*). Finally, the state* q_M *represents a Menu.*

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* $\Delta = \{i \to q_d | q_N | q_{id} | q_t \mid 0 \leq i \leq 9\} \cup \{N(q_d, q_N) \to q_N | q_{id} | q_t, L_0(q_{id}, q_t) \to q_L, L(q_{id}, q_t, q_L) \to q_L, M(q_{id}, q_t, q_L) \to q_M\}$.
The constraint C *will ensure that all the dish identifiers involved on our menu are different (i.e.* q_{id} *is a key) and that the time needed to prepare each dish will always be the same:* $C = q_{id} \not\approx q_{id} \wedge q_t \approx q_t$.
An example of a term in $\mathcal{L}(\mathcal{A})$ *with an associated successful run are depicted together in Figure 1.* ◊

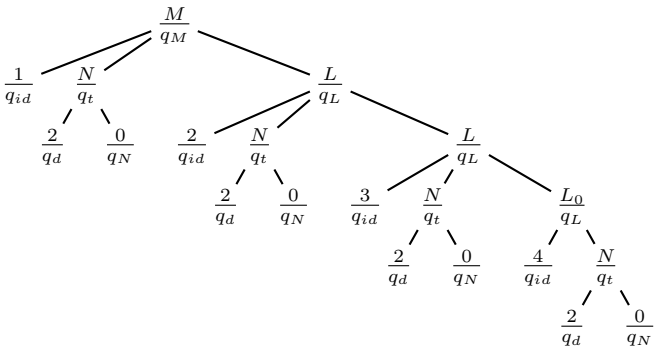

Figure 1. Term and successful run (Ex. 2.4).

Decision Problems. The *membership* is the problem to decide, given a term $t \in \mathcal{T}(\Sigma)$ and a TAGC $\mathcal{A}$ over Σ whether $t \in \mathcal{L}(\mathcal{A})$.

Proposition 2.5 *Membership is NP-complete for TAGC.*

Proof. Given a TAGC $\mathcal{A} = \langle Q, \Sigma, F, C, \Delta \rangle$ and a term $t \in \mathcal{T}(\Sigma)$, a non-deterministic algorithm consist in guessing a function r from $Pos(t)$ into Q, and checking that r is a successful run of $\mathcal{A}$ on t. The checking can be performed in polynomial time.

For NP-hardness, [14, 17] present PTIME reductions of satisfiability of Boolean expressions into membership for TAGC whose constraints are conjunctions of equalities of the form $q \approx q$. □

We recall that for plain TA, membership is in PTIME.

The *universality* is the problem to decide, given a TAGC $\mathcal{A}$ over Σ whether $\mathcal{L}(\mathcal{A}) = \mathcal{T}(\Sigma)$. It is known to be undecidable already for the small subclass of TAGC.

Proposition 2.6 *[14, 17] Universality is undecidable for positive TAGC containing only* $\approx$ *constraints.*

The following consequence is a new result for TAGEDs.

Proposition 2.7 *It is undecidable whether the language of a given positive TAGC containing only* $\approx$ *atomic constraints is regular.*

Proof. We show that universality is reducible to regularity. Let us define the quotient of a term language L by a term s *wrt* a function symbol f: $L/s := \{t \mid f(s,t) \in L\}$. This operation preserves regular languages: for all s and f, if L is regular then L/s is regular.

Let L and L' be two TAGC languages over Σ such that L' is not regular (such a language exists) and let $L_1 := f\big(L, \mathcal{T}(\Sigma)\big) \cup f\big(\mathcal{T}(\Sigma), L'\big)$ where f is a binary symbol, possibly not in Σ ($f(L, \mathcal{T}(\Sigma))$ denotes $\{f(s,t) \mid s \in L, t \in \mathcal{T}(\Sigma)\}$). It is obvious that L_1 is a TAGC language.

If $L = \mathcal{T}(\Sigma)$, then $L_1 = f\big(\mathcal{T}(\Sigma), \mathcal{T}(\Sigma)\big)$ and it is regular. Assume that $L \neq \mathcal{T}(\Sigma)$ and let $s \in \mathcal{T}(\Sigma) \setminus L$. By construction, $L_1/s = L'$ which is not regular. Hence L_1 is not regular. Therefore $L = \mathcal{T}(\Sigma)$ iff the TAGC language L_1 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 3.

Closure Properties. Let us conclude this first section with the closure properties of the TAGC languages.

Proposition 2.8 *The class of TAGC languages is closed under union and intersection but not under complementation.*

Proof. We use a classical disjoint union for union and Cartesian product of state sets for intersection, with a careful redefinition of constraints on this product.

More precisely, let $\mathcal{A}_1 = \langle Q_1, \Sigma_1, F_1, C_1, \Delta_1 \rangle$ and $\mathcal{A}_2 = \langle Q_2, \Sigma_2, F_2, C_2, \Delta_2 \rangle$ be two TAGCs. We can assume *wlog* that Q_1 and Q_2 are disjoint.

The TAGC $\mathcal{A}_\cup = \langle Q_1 \uplus Q_2, \Sigma_1 \cup \Sigma_2, F_1 \uplus F_2, C_1 \vee C_2, \Delta_1 \uplus \Delta_2 \rangle$ recognizes $\mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2)$.

We define a TAGC $\mathcal{A}_\cap = \langle Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, F_1 \times Q_2 \cup Q_1 \times F_2, C_\cap, \Delta_\cap \rangle$ recognizing $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$. The constraint $C_\cap$ is obtained from $C_1 \wedge C_2$ by replacing every atom $q_1 \approx q_1'$ with $q_1, q_1' \in Q_1$ (resp. $q_2 \approx q_2'$ with $q_2, q_2' \in Q_2$) by $\bigwedge_{q_2, q_2' \in Q_2} \langle q_1, q_2 \rangle \approx \langle q_1', q_2' \rangle$ (resp. $\bigwedge_{q_1, q_1' \in Q_1} \langle q_1, q_2 \rangle \approx \langle q_1', q_2' \rangle$), and similarly for the atoms $q_1 \not\approx q_1'$, $q_2 \not\approx q_2'$. The set of transitions is $\Delta_\cap = \big\{ f\big(\langle q_{1,1}, q_{2,1} \rangle, \ldots, \langle q_{1,n}, q_{2,n} \rangle\big) \to \langle q_1, q_2 \rangle \mid f(q_{i,1}, \ldots, q_{i,n}) \to q_i \in \Delta_i \text{ for } i = 1, 2 \big\}$.
The closure under complementation of TAGC would contradict Proposition 2.6 and Theorem 3.21 below. □

3 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 just PCTAGC.

Lemma 3.1 *Given a TAGC $\mathcal{A}$, some PCTAGC $\mathcal{A}_1, \ldots, \mathcal{A}_n$ can be computed satisfying $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cup \ldots \cup \mathcal{L}(\mathcal{A}_n)$.*

In order to prove this lemma, we shall conveniently use some extensions of TAGC studied in Section 4.1. The reader is therefore referred to this section for a complete proof.

The decidability of emptiness for PCTAGC is proved in three steps. In Subsection 3.1, we present a new notion of pumping which allows to transform a run into a smaller run under certain conditions. In Subsection 3.2, we define a well quasi-ordering $\leq$ on a certain set S. In Subsection 3.3, 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 well and the form of the sequence implies 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.

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 3.3).

3.1 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.

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 3.2 *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 3.3 *According to Definition 3.2, for our running example (Example 2.4), we have the H_i and $\check{H}_i$ presented in Figure 2.* ◊

The following lemma is rather straightforward from the previous definition.

Lemma 3.4 *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.*

Proof. For the first fact, note that any proper prefix p of a position $\bar{p}$ in $H_i \cup \check{H}_i$ satisfies $h(r|_p) > i$. Thus, such a p is not in $H_i \cup \check{H}_i$. For the second fact, consider any p in $Pos(r)$. If $h(r|_p) \leq i$ holds, then the smallest position $\bar{p}$ satisfying $\bar{p} < p$ and $h(r|_{\bar{p}}) \leq i$ is in $H_i \cup \check{H}_i$, and we are done. Otherwise, if $h(r|_p) > i$ holds, then the smallest position $\bar{p}$ of the form $p.1.\ldots.1$ and satisfying $h(r|_{\bar{p}}) \leq i$ is in $H_i \cup \check{H}_i$, and we are done. □

Definition 3.5 *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 $I : (H_i \cup \check{H}_i) \to (H_j \cup \check{H}_j)$ *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 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 3.6 *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.

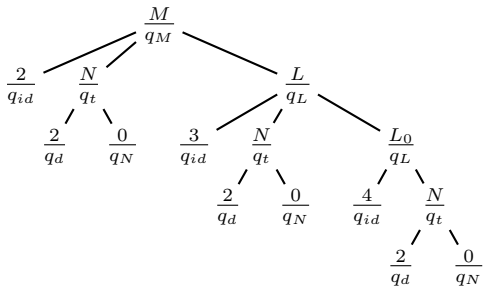

Figure 3. Pump-injection of Example 3.6.

Lemma 3.7 *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 $j + k$.

Proof. Position p is obviously a position of r' since no position in $H_i \cup \check{H}_i$ is a proper prefix of p. We prove the second part of the statement by induction on k. First, assume $k = 0$. Then, $h(r|_p)$ is i. Thus, p is in H_i, say p is $\bar{p}_1$. Therefore, $r'|_p$ is $r|_{I(\bar{p}_1)}$. By Condition (C_1) of the definition of pump-injection, $I(\bar{p}_1) \in H_j$ holds. Hence, $h(r'|_p) = h(r|_{I(\bar{p}_1)}) = j$.

Now, assume $k > 0$. Let m be the arity of $\mathtt{symbol}(r|_p)$. Thus, $p.1, \ldots, p.m$ are all the child positions of p in r. Since $h(r|_p)$ is $i + k$, all $h(r|_{p.1}), \ldots, h(r|_{p.m})$ are smaller than or equal to $i + k - 1$, and at least one of them is equal to $i + k - 1$.

Consider any α in $\{1, \ldots, m\}$. If $h(r|_{p.\alpha})$ is $i + k'$ for some $0 \leq k' \leq k - 1$, then, by induction hypothesis, $h(r'|_{p.\alpha})$ is $j + k'$. Otherwise, if $h(r|_{p.\alpha})$ is strictly smaller than i, then $p.\alpha$ is one of the positions in $\check{H}_i$, say $\bar{p}_1$. Moreover, $r'|_{\bar{p}_1}$ is $r|_{I(\bar{p}_1)}$, and by Condition (C_1) of the definition of I, $I(\bar{p}_1)$ belongs to $\check{H}_j$. Therefore, $h(r|_{I(\bar{p}_1)}) < j$ holds, and hence, $h(r'|_{p.\alpha}) = h(r'|_{\bar{p}_1}) = h(r|_{I(\bar{p}_1)}) < j \leq j + k - 1$ holds.

From the above cases we conclude that, if $h(r|_{p.\alpha})$ is $i + k - 1$, then $h(r'|_{p.\alpha})$ is $j + k - 1$, and if $h(r|_{p.\alpha})$ is smaller than $i+k-1$, then $h(r'|_{p.\alpha})$ is smaller than $j+k-1$. It follows that all $h(r'|_{p.1}), \ldots, h(r'|_{p.m})$ are smaller than or equal to $j + k - 1$, and at least one of them is equal to $j + k - 1$. As a consequence, $h(r'|_p)$ is $j + k$. □

Corollary 3.8 *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.

Lemma 3.9 *Let $\mathcal{A}$ be a PCTAGC. Let r be a run of $\mathcal{A}$. Let r' be the global pumping $r[r|_{I(\bar{p}_1)}]_{\bar{p}_1} \dots [r|_{I(\bar{p}_n)}]_{\bar{p}_n}$ 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.

Proof. The first statement follows by Lemma 3.7. We prove the second part of the statement by induction on $h(r|_{p_1}) + h(r|_{p_2})$. We distinguish the following cases:
i. Assume that $h(r|_{p_1}) \neq h(r|_{p_2})$. Then, $(\mathtt{term}(r|_{p_1}) \neq \mathtt{term}(r|_{p_2}))$ hold and, moreover, $h(r|_{p_1}) = i + k_1$ and $h(r|_{p_2}) = i + k_2$ hold for some different natural numbers k_1 and k_2. By Lemma 3.7, $h(r'|_{p_1}) = j + k_1$ and $h(r'|_{p_2}) = j + k_2$. Thus, $(\mathtt{term}(r'|_{p_1}) \neq \mathtt{term}(r'|_{p_2}))$ hold, and we are done.

ii. Assume that $h(r|_{p_1}) = h(r|_{p_2}) = i + k$ for some k. We start by assuming the case $k = 0$. Then, $h(r|_{p_1})$ is i. Thus, p_1, p_2 are in H_i, say p_1 is $\bar{p}_1$ and p_2 is $\bar{p}_2$. Therefore, $r'|_{p_1}$ is $r|_{I(\bar{p}_1)}$ and $r'|_{p_2}$ is $r|_{I(\bar{p}_2)}$. By Condition (C_3) of the definition of pump-injection, $(\mathtt{term}(r|_{\bar{p}_1}) = \mathtt{term}(r|_{\bar{p}_2})) \Leftrightarrow (\mathtt{term}(r|_{I(\bar{p}_1)}) = \mathtt{term}(r|_{I(\bar{p}_2)}))$ holds. Thus, $(\mathtt{term}(r|_{p_1}) = \mathtt{term}(r|_{p_2})) \Leftrightarrow (\mathtt{term}(r'|_{p_1}) = \mathtt{term}(r'|_{p_2}))$ holds and we are done.

Now, we assume $k > 0$. Note that, in this case, $\mathtt{symbol}(r'|_{p_1}) = \mathtt{symbol}(r|_{p_1})$ and $\mathtt{symbol}(r'|_{p_2}) = \mathtt{symbol}(r|_{p_2})$ hold. In the case where $\mathtt{symbol}(r|_{p_1})$ differs from $\mathtt{symbol}(r|_{p_2})$, it is clear that $(\mathtt{term}(r|_{p_1}) \neq \mathtt{term}(r|_{p_2}))$ and $(\mathtt{term}(r'|_{p_1}) \neq \mathtt{term}(r'|_{p_2}))$ hold, and we are done. Hence, we consider the remaining case where $\mathtt{symbol}(r|_{p_1}) = \mathtt{symbol}(r|_{p_2})$ holds. Let m be the arity of $\mathtt{symbol}(r|_{p_1})$. Thus, $p_1.1, \dots, p_1.m$ and $p_2.1, \dots, p_2.m$ are all the child positions of p_1 and p_2 in r, respectively. In order to prove $(\mathtt{term}(r|_{p_1}) = \mathtt{term}(r|_{p_2})) \Leftrightarrow (\mathtt{term}(r'|_{p_1}) = \mathtt{term}(r'|_{p_2}))$ it suffices to prove $(\mathtt{term}(r|_{p_1.\alpha}) = \mathtt{term}(r|_{p_2.\alpha})) \Leftrightarrow (\mathtt{term}(r'|_{p_1.\alpha}) = \mathtt{term}(r'|_{p_2.\alpha}))$ for any α in $\{1, \dots, m\}$. Thus, we consider any of such α's and distinguish the following cases:

i.a. If $h(r|_{p_1.\alpha}), h(r|_{p_2.\alpha}) \geq i$ holds, then the result follows by induction hypothesis.

i.b. If $h(r|_{p_1.\alpha}) \geq i$ and $h(r|_{p_2.\alpha}) < i$, then $\mathtt{term}(r|_{p_1.\alpha}) \neq \mathtt{term}(r|_{p_2.\alpha})$ holds, and moreover, $h(r|_{p_1.\alpha}) = i + k_1$ for some $k_1 \geq 0$, and $p_2.\alpha$ belongs to $\check{H}_i$. By Lemma 3.7, $h(r'|_{p_1.\alpha}) = j + k_1$ holds. By Condition (C_1) of the definition of pump-injection, $h(r'|_{p_2.\alpha}) < j$ holds. Thus, $\mathtt{term}(r'|_{p_1.\alpha}) \neq \mathtt{term}(r'|_{p_2.\alpha})$ holds, and we are done.

i.c. The case where $h(r|_{p_1.\alpha}) < i$ and $h(r|_{p_2.\alpha}) \geq i$ hold is analogous to the previous one.

i.d. If $h(r|_{p_1.\alpha}), h(r|_{p_2.\alpha}) < i$, then $p_1.\alpha, p_2.\alpha$ belong to $\check{H}_i$, say $p_1.\alpha$ is $\bar{p}_1$ and $p_2.\alpha$ is $\bar{p}_2$. Therefore, $r'|_{p_1.\alpha}$ is $r|_{I(\bar{p}_1)}$ and $r'|_{p_2.\alpha}$ is $r|_{I(\bar{p}_2)}$. By Condition (C_3) of the definition of pump-injection, $(\mathtt{term}(r|_{\bar{p}_1}) = \mathtt{term}(r|_{\bar{p}_2})) \Leftrightarrow (\mathtt{term}(r|_{I(\bar{p}_1)}) = \mathtt{term}(r|_{I(\bar{p}_2)}))$ holds. Thus, $(\mathtt{term}(r|_{p_1.\alpha}) = \mathtt{term}(r|_{p_2.\alpha})) \Leftrightarrow (\mathtt{term}(r'|_{p_1.\alpha}) = \mathtt{term}(r'|_{p_2.\alpha}))$ holds and we are done. □ As a consequence of previous lemmas, we prove that the result of a global pumping is a run.

Lemma 3.10 *Let $\mathcal{A}$ be a PCTAGC. Let r be a run of $\mathcal{A}$. Let r' be the global pumping $r[r|_{I(\bar{p}_1)}]_{\bar{p}_1} \dots [r|_{I(\bar{p}_n)}]_{\bar{p}_n}$ 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 3.4, 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 3.4, $I(\bar{p}_1) \parallel I(\bar{p}_2)$ 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 3.7, $h(r'|_{p_1})$ 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 3.9, $(\mathtt{term}(r|_{p_1}) = \mathtt{term}(r|_{p_2})) \Leftrightarrow$

$(\mathtt{term}(r'|_{p_1}) = \mathtt{term}(r'|_{p_2}))$ 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. □

3.2 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 3.11 *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 $\langle P_{11}, P_{12}\rangle \leq \langle P_{21}, P_{22}\rangle$ if $P_{11} \leq P_{21}$ and $P_{12} \leq P_{22}$. ◊

As a direct consequence of Higman's Lemma [15] we have the following:

Lemma 3.12 *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 large sequences $x_1, \ldots, x_k$ of natural numbers satisfying $x_i > x_j$ for each $1 \leq i < j \leq k$. In order to bound the length of such sequences, it is sufficient to force that the first element and each next element of the sequence are chosen among a finite number of possibilities. As a particular case of this fact we have the following result:

Lemma 3.13 *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 $\langle T_{11}, T_{12}\rangle, \langle T_{21}, T_{22}\rangle, \langle T_{31}, T_{32}\rangle, \ldots \langle T_{L1}, T_{L2}\rangle$ of pairs of multisets of n-tuples of natural numbers satisfying the following conditions:*

- *The tuple $\langle 0, \ldots, 0\rangle$ does not occur in any T_{i1}, T_{i2}, for i in $\{1, \ldots, L\}$.*
- *$\mathtt{sum}(T_{11}) = 1$ and $T_{12} = \emptyset$ hold.*
- *For each i in $\{1, \ldots, L-1\}$, $\mathtt{sum}(T_{(i+1)1}) + \mathtt{sum}(T_{(i+1)2}) \leq a \cdot \mathtt{sum}(T_{i1}) + \mathtt{sum}(T_{i2})$ holds.*
- *There are no i, j satisfying $1 \leq i < j \leq L$ and $\langle T_{i1}, T_{i2}\rangle \leq \langle T_{j1}, T_{j2}\rangle$*

In order to bound the height of an accepted term with minimum height by a PCTAGC $\mathcal{A}$, Lemma 3.13 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 $\mathcal{A}$.

3.3 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 terms to tuples and sets of positions to multisets of tuples.

Definition 3.14 *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: $\langle |\{p \in P \mid \mathtt{term}(r|_p) = t \wedge r(p) = q_1\}|, \ldots, |\{p \in P \mid \mathtt{term}(r|_p) = t \wedge r(p) = q_n\}|\rangle$* ◊

Definition 3.15 *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 $[r_{t_1,P}, \ldots, r_{t_k,P}]$.* ◊

Example 3.16 *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.* ◊

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

Lemma 3.17 *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. $\Rightarrow$. Assume there exists a pump-injection $I : (H_i \cup \check{H}_i) \to (H_j \cup \check{H}_j)$. We just prove $r_{H_i} \leq r_{H_j}$, since $r_{\check{H}_i} \leq r_{\check{H}_j}$ can be proved analogously. By Condition (C_1) of the definition of pump-injection, $I(H_i) \subseteq H_j$. We write $\{\mathtt{term}(r|_p) \mid p \in H_i\}$ and $\{\mathtt{term}(r|_p) \mid p \in H_j\}$ more explicitly as $\{t_{i,1}, \ldots, t_{i,\alpha}\}$ and $\{t_{j,1}, \ldots, t_{j,\beta}\}$, respectively. Hence, it remains to prove that $[r_{t_{i,1},H_i}, \ldots, r_{t_{i,\alpha},H_i}] \leq [r_{t_{j,1},H_j}, \ldots, r_{t_{j,\beta},H_j}]$. To this end we define the function $I' : \{1, \ldots, \alpha\} \to \{1, \ldots, \beta\}$ as follows. For each

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

Figure 4. Multisets r_{H_i}, $r_{\check{H}_i}$ (Example 3.16).

γ in $\{1,\ldots,\alpha\}$, we choose a position p in H_i satisfying $\mathtt{term}(r|_p) = t_{i,\gamma}$, determine the index δ of the term $t_{j,\delta}$ satisfying $t_{j,\delta} = \mathtt{term}(r|_{I(p)})$, and define $I'(\gamma) := \delta$. This function I' is injective due to Condition (C_3) of the definition of pump-injection. In order to conclude, it suffices to prove $r_{t_{i,\gamma},H_i} \leq r_{t_{j,I'(\gamma)},H_j}$ for each γ in $\{1,\ldots,\alpha\}$. We just prove it for $\gamma = 1$. For proving $r_{t_{i,1},H_i} \leq r_{t_{j,I'(1)},H_j}$ it suffices to prove the following statement for each state q of $\mathcal{A}$: $\big|\{p \in H_i \mid \mathtt{term}(r|_p) = t_{i,1} \wedge \mathtt{state}(r|_p) = q\}\big| \leq \big|\{p \in H_j \mid \mathtt{term}(r|_p) = t_{j,I'(1)} \wedge \mathtt{state}(r|_p) = q\}\big|$.

To this end, since I is injective, it suffices to prove that $I(\{p \in H_i \mid \mathtt{term}(r|_p) = t_{i,1} \wedge \mathtt{state}(r|_p) = q\})$ is included in $\{p \in H_j \mid \mathtt{term}(r|_p) = t_{j,I'(1)} \wedge \mathtt{state}(r|_p) = q\}$ for each state q of $\mathcal{A}$. Thus, consider any $\bar{p}$ of $\{p \in H_i \mid \mathtt{term}(r|_p) = t_{i,1} \wedge \mathtt{state}(r|_p) = q\}$. Let p' be the chosen position for defining $I'(1)$. In particular, $\mathtt{term}(r|_{p'}) = t_{i,1}$ and $\mathtt{term}(r|_{I(p')}) = t_{j,I'(1)}$ hold. Note that $\mathtt{term}(r|_{\bar{p}}) = \mathtt{term}(r|_{p'}) = t_{i,1}$ holds. Thus, by Condition (C_3) of the definition of pump-injection, $\mathtt{term}(r|_{I(\bar{p})}) = \mathtt{term}(r|_{I(p')})$ holds. Therefore, $\mathtt{term}(r|_{I(\bar{p})}) = t_{j,I'(1)}$ holds. In order to show the inclusion $I(\bar{p}) \in \{p \mid \mathtt{term}(r|_p) = t_{j,I'(1)} \wedge \mathtt{state}(r|_p) = q\}$ it rests to see $\mathtt{state}(r|_{I(\bar{p})}) = q$. Note that, since $\bar{p}$ belongs to $\{p \mid \mathtt{term}(r|_p) = t_{i,1} \wedge \mathtt{state}(r|_p) = q\}$, $\mathtt{state}(r|_{\bar{p}}) = q$ holds. By Condition (C_2) of the definition of pump-injection, $\mathtt{state}(r|_{I(\bar{p})}) = \mathtt{state}(r|_{\bar{p}}) = q$ holds, and we are done.

$\Leftarrow$. Assume $\langle r_{H_i}, r_{\check{H}_i}\rangle \leq \langle r_{H_j}, r_{\check{H}_j}\rangle$ 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 $\mathtt{term}(r|_{\bar{p}'_1}) \neq \mathtt{term}(r|_{\bar{p}'_2})$ and $\mathtt{term}(r|_{I(\bar{p}'_1)}) \neq \mathtt{term}(r|_{I(\bar{p}'_2)})$ hold. Hence, this simple case is enough to prove the whole statement.

We write $\{\mathtt{term}(r|_p) \mid p \in H_i\}$ and $\{\mathtt{term}(r|_p) \mid p \in H_j\}$ 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 $\mathcal{A}$: $\big|\{p \in H_i \mid \mathtt{term}(r|_p) = t_{i,\delta} \wedge r(p) = q\}\big| \leq \big|\{p \mid \mathtt{term}(r|_p) = t_{j,I'(\delta)} \wedge r(p) = q\}\big|$.

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

Example 3.18 *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 3.17. 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 3.13.

Lemma 3.19 *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 $\{0,\ldots,h(r)\}$, $|H_i| = \mathtt{sum}(r_{H_i})$ and $|\check{H}_i| = \mathtt{sum}(r_{\check{H}_i})$.

Proof. Item (1) is trivial by definition of H_i and $\check{H}_i$ for $i = h(r)$. For Item (2), it suffices to observe that the positions in $H_{i-1} \cup \check{H}_{i-1}$ are all the positions of $\check{H}_i$ plus all the child positions in H_i, and that each position has at most a childs. For Item (3) we just prove $|H_i| = \mathtt{sum}(r_{H_i})$, since $|\check{H}_i| = \mathtt{sum}(r_{\check{H}_i})$ can be proved analogously. We write $\{\mathtt{term}(r|_p) \mid p \in H_i\}$ more explicitly as $\{t_1,\ldots,t_\alpha\}$.

Note that H_i is the disjoint union $\{p \in H_i \mid \mathtt{term}(r|_p) = t_1\} \cup \ldots \cup \{p \in H_i \mid \mathtt{term}(r|_p) = t_\alpha\}$. Thus, $|H_i|$ equals $|\{p \in H_i \mid \mathtt{term}(r|_p) = t_1\}| + \ldots + |\{p \in$

$H_i \mid \mathtt{term}(r|_p) = t_\alpha\}|$. We conclude by observing that $|\{p \in H_i \mid \mathtt{term}(r|_p) = t_1\}| = \mathtt{sum}(r_{t_1,H_i}), \ldots |\{p \in H_i \mid \mathtt{term}(r|_p) = t_\alpha\}| = \mathtt{sum}(r_{t_\alpha,H_i})$ holds. □

Lemma 3.20 *Let $B : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be the computable function of Lemma 3.13. 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 3.19, $|H_{h(r)}| = 1$ 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 $\{0, \ldots, h(r)\}$, $|H_i| = \mathtt{sum}(r_{H_i})$ and $|\check{H}_i| = \mathtt{sum}(r_{\check{H}_i})$. Thus, $\mathtt{sum}(r_{H_{h(r)}}) = 1$, $\mathtt{sum}(r_{\check{H}_{h(r)}}) = 0$, and for each i in $\{1, \ldots, h(r)\}$, $\mathtt{sum}(r_{H_{i-1}}) + \mathtt{sum}(r_{\check{H}_{i-1}}) \leq a \cdot \mathtt{sum}(r_{H_i}) + \mathtt{sum}(r_{\check{H}_i})$. Hence, since $h(r) \geq B(a, n)$ holds, by Lemma 3.13 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 3.17, 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 3.21 *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 3.20, there exists a global pumping r' on r. By Corollary 3.8, $h(r') < h(r)$ holds. Moreover, by the definition of global pumping, $r'(\Lambda) = r(\Lambda)$ holds. Finally, by Lemma 3.10, 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 3.1 and Theorem 3.21, we can conclude to the decidability of emptiness for TAGC.

Corollary 3.22 *Emptiness is decidable for TAGC.*

4 Extensions

In this section, we extend the emptiness result by considering the addition of new constraints to TAGC. For convenience, we denote below by TAGC$[\tau_1, \ldots, \tau_k]$ the class of TAGC containing atomic constraints of type $\tau_1, \ldots, \tau_k$. For instance, with this notation the class TAGC defined in sections 2.2 is TAGC$[\approx, \not\approx]$.

4.1 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.

Example 4.1 *Let us add a new argument to the dishes of to the menu of Example 2.4 which represents the price coded on two digits by a term $N(d_1, d_0)$. We add a new state q_p for the type of prices, and other states q_{cheap}, $q_{moderate}$, $q_{expensive}$, q_{chic} describing price level ranges, and transitions $0|1 \to q_{cheap}$, $2|3 \to q_{moderate}$, $4|5|6 \to q_{expensive}$, $7|8|9 \to q_{chic}$ and $N(q_{cheap}, q_d) \to q_p$,... The price is a new argument of L_0, L and M, hence we replace the transitions with these symbols in input by $L_0(q_{id}, q_t, q_p) \to q_L$, $L(q_{id}, q_t, q_p, q_L) \to q_L$, $M(q_{id}, q_t, q_p, q_L) \to q_M$. We can use a linear equality $|q_{cheap}| + |q_{moderate}| - |q_{expensive}| - |q_{chic}| \geq 0$ to characterize the moderate menus, and $|q_{expensive}| + |q_{chic}| \geq 6$ to characterize the menus with too many expensive dishes. A linear equality $\|q_p\| \leq 1$ expresses that all the dishes have the same price.*

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 [5]) and it has a decidable emptiness test. Indeed, the set of successful runs of a given TA with state set Q describes a context free language (over Q^*), and the Parikh projection (the set of tuples over $\mathbb{N}^{|Q|}$ whose components are the $[\![\, |q| \,]\!]_r$ for every run r) of such a language is a semi-linear set. The idea for deciding emptiness for a TAGC$[|.|_\mathbb{Z}]$ $\mathcal{A}$ is to compute this semi linear set and test the emptiness of its intersection with the set of solutions in $\mathbb{N}^{|Q|}$ of $C_\mathcal{A}$, (a Boolean combinations of linear inequalities of type $|.|_\mathbb{Z}$) which is also semi-linear. This can be done in NPTIME, see [5].

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

Theorem 4.2 *Emptiness is undecidable for PTAGC$[\approx, |.|_\mathbb{Z}]$.*

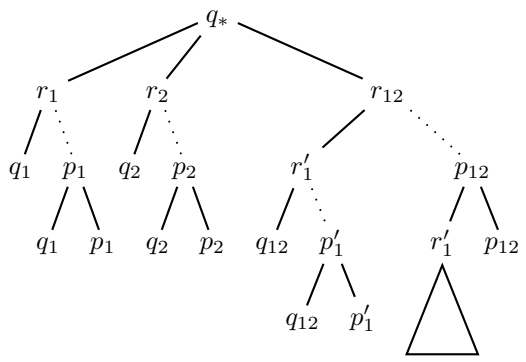

Figure 5. A run on $*(t_{x_1}, t_{x_2}, t_{x_1.x_2})$

Proof. We reduce Hilbert's tenth problem. Let $a_1\, x_1^{i_{1,1}} \dots x_m^{i_{1,m}} + \dots + a_n\, x_1^{i_{n,1}} \dots x_m^{i_{n,m}} = 0 (\phi)$ be a Diophantine equation with m variables $x_1, \dots, x_m$, $i_{j,k} \geq 0$ for all $j \leq n$, $k \leq m$ and $a_1, \dots, a_n \in \mathbb{Z}$. We will construct a TAGC$[\approx, |.|_{\mathbb{Z}}]$ $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff there exists a valuation of $x_1, \dots, x_m$ in $\mathbb{N}$ satisfying the above equation.

The most important step of the construction is that we can encode (non negative) integer multiplication with TAGC$[\approx, |.|_{\mathbb{Z}}]$. Let us consider for instance the product $x_1.x_2$ and three TAGC states q_1, q_2 and q_{12}, whose number of occurrences will represent the respective valuations of x_1, x_2 and their product (with our notations, $|q_1| = x_1$, $|q_2| = x_2$, $|q_{12}| = x_1.x_2$). We use a signature with symbols a and b of arity 0, f of arity 2, and $*$ of arity 3. For $x \geq 1$, let t_x be the right comb $f(a, f(a, \dots, f(a, b)))$ with x occurrences of a and one occurrence of b. The product $x_1.x_2$ is represented by $t_{x_1,x_2} := f(t_{x_1}, \dots, f(t_{x_1}, b))$ with x_2 occurrences of t_{x_1}. Let us now construct a TAGC$[\approx, |.|_{\mathbb{Z}}]$ $\mathcal{A}_*$ recognizing all $*(t_{x_1}, t_{x_2}, t_{x_1,x_2})$. During the computation of $\mathcal{A}_*$, in the first argument t_{x_1}, every a will go to the state q_1, with the transition rule $a \to q_1$, and t_{x_1} will go to a state r_1. For the intermediate steps, we use a state p_1 and transitions $b \to p_1$, $f(q_1, p_1) \to p_1$, and $f(q_1, p_1) \to r_1$. We proceed in the same manner for t_{x_2} with the states q_2, r_2, p_2. Finally, for the recognition of the third argument t_{x_1,x_2} by $\mathcal{A}_*$, every a will go to the state q_{12}, every subterm t_{x_1} with go to a state r'_1 and the term t_{x_1,x_2} will go to r_{12}. For this purpose we have the transitions $a \to q_{12}$, $b \to p'_1$, $f(q_{12}, p'_1) \to p'_1$, $f(q_{12}, p'_1) \to r'_1$, and $b \to p_{12}$, $f(r'_1, p_{12}) \to p_{12}$, $f(r'_1, p_{12}) \to r_{12}$. Finally, we add a rule $*(r_1, r_2, r_{12}) \to q_*$ to $\mathcal{A}_*$. Let r be a run of $\mathcal{A}_*$ on a term $t = *(t_{x_1}, t_{x_2}, t_3)$ similar to the one presented in Figure 5. By construction, the number of occurrences of q_1 in r is x_1, *i.e.* $[\![\, |q_1| \,]\!]_r = x_1$. Similarly, $[\![\, |q_2| \,]\!]_r = x_2$. Let us consider the constraint $C_* = |r'_1| = |q_2| \wedge r_1 \approx r'_1$ (for simplicity, we use linear equalities representing the conjunction of two linear inequalities). By imposing the constraint C_* to the run r of $\mathcal{A}_*$, we ensure that t_3, the third argument of t, contains x_2 nested copy of t_{x_1}. It follows that the number $[\![\, |q_{12}| \,]\!]_r$ of occurrences of q_{12} in the run r is equal to $x_1.x_2$.

The general construction of the TAGC$[\approx, |.|_{\mathbb{Z}}]$ $\mathcal{A} = \langle Q, \Sigma, F, C, \Delta \rangle$ associated to the above Diophantine equation (ϕ) follows the same principle, except that we may have several level of nesting in a term like t_3 above. For encoding the product of more than two integers, we have in $\mathcal{A}$ some constraints $\approx$ and $|.|_{\mathbb{Z}}$ similar to the above ones, and an additional constraint $\sum a_j.|q_j| = 0$ where each q_j is uniquely associated to a variable x_j, as above.

Let $\mathcal{W}$ be the set containing all the variables of (ϕ), $\{x_1, \dots, x_m\}$ and all the prefixes of the monomials of (ϕ) $x_j^{i_{j,1}} \dots x_m^{j_{1,m}}$ for all $j \leq n$ (we consider every monomial as a word over the alphabet $\{x_1, \dots, x_m\}$). Let us denote the prefix ordering over strings by $\preceq$, and for every words v, v', w such that $w = v.v'$ (*i.e.* $v \preceq w$) we recall that $w - v = v'$ and let $w \div v$ be the first letter of v' when this latter word is not empty. We assume below that $\mathcal{W}$ is ordered arbitrarily in a sequence without duplicates $\overrightarrow{w} = (w_1, \dots, w_N)$.

The signature Σ contains the symbols a, b and f with the respective arity 0, 0 and 2 as above, and two symbols c of arity 0 and g of arity 2. Let $Q = \{q_w, s_w, r_w^v, p_w^v \mid v, w \in \mathcal{W}, v \preceq w\} \cup \{s_0\}$ and $F = \{s_{w_N}\}$. The transitions of Δ are the following: $a \to q_w$, $b \to p_w^v$, $f(q_w, p_w^v) \to p_w^v$, $f(q_w, p_w^v) \to r_w^v$ for all $v, w \in \mathcal{W}$ with $v \preceq w$, and $f(r_w^v, p_w^{v.x_i}) \to p_w^{v.x_i}$, $f(r_w^v, p_w^{v.x_i}) \to r_w^{v.x_i}$, for all $v, w \in \mathcal{W}$ and x_i variable of (ϕ) ($i \leq m$), with $v.x_i \preceq w$. Moreover, we also have: $c \to s_0$, $g(r_{w_1}^{w_1}, s_0) \to s_{w_1}$ and $g(r_{w'}^{w'}, s_w) \to s_{w'}$ for all $w, w' \in \mathcal{W}$ such that $w = w_i$ and $w' = w_{i+1}$ in the sequence $\overrightarrow{w}$, for some $i < N$.

We can observe that with these transitions, if r is an accepting run of $ta(\mathcal{A})$ on some term t, then for all $w \in \mathcal{W}$, the state s_w occurs exactly once in r at some position $p_w \in 2^*$, and the leaves of $r|p_w.1$ are all labelled by the state q_w. Moreover, the state q_w occurs only in this subterm of r.

Finally, the constraint C of $\mathcal{A}$ is defined as the conjunction of the following atomic constraints:

i. $|r_w^v| = |q_{w \div v}|$ for all $v, w \in \mathcal{W}$ with $v \prec w$,

ii. $r_w^v \approx r_{w'}^v$ for all $v, w, w' \in \mathcal{W}$ with $v \preceq w, w'$,

iii. $\displaystyle\sum_{j=1}^{n} a_j.|q_{w_j}| = 0.$

In *iii.*, the a_j's are the coefficients of (ϕ) and $w_j = x_1^{i_{j,1}} \dots x_m^{i_{j,m}}$ (we assume that the n first elements of the sequence $\overrightarrow{w}$ correspond to these monomials).

With the above observation, the construction of the transition rules of $\mathcal{A}$, and the above constraints *i.* and *ii.* we have

that for all run r of $\mathcal{A}$ on a term t, all variables x_i of (ϕ) and all $w.x_i \in \mathcal{W}$, $[\![\, |q_{w.x_i}| \,]\!]_r = [\![\, |q_w| \,]\!]_r \cdot [\![\, |q_{x_i}| \,]\!]_r$. It follows, using the constraint iii, that $\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff (ϕ) has a solution in $\mathbb{N}$. □

We present now a restriction on linear equalities which permits to have a decidable emptiness test when combined with the $\approx$ and $\not\approx$ 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. Note that it is equivalent to consider inequalities in both directions whose coefficients are all non-negative, like $\sum a_q.|q| \geq a$, with $a_q, a \in \mathbb{N}$, to refer to $\sum -a_q.|q| \leq -a$, and also linear equalities $\sum a_q.|q| = a$, with $a_q, a \in \mathbb{N}$, to refer to a conjunction of two natural linear disequalities. The types of the natural linear inequalities are denoted by $|.|_{\mathbb{N}}$ and $\|.\|_{\mathbb{N}}$. Below, we shall abbreviate these two types by $\mathbb{N}$.

The main difference between the linear inequalities of type $|.|_{\mathbb{Z}}$ and $|.|_{\mathbb{N}}$ (and respectively $\|.\|_{\mathbb{Z}}$ and $\|.\|_{\mathbb{N}}$) is that the former permits to compare the respective number of occurrence of two states, like *e.g.* in $|q| \leq |q'|$, whereas the latter only permit to compare the number of occurrences of one state (or a sum of the number occurrences of several states with coefficients) to a constant like *e.g.* in $|q| \leq 4$ or $|q| + 2.|q'| \leq 9$. This difference permits to obtain the decidability of the extension of TAGC with arithmetic constraints. The proof that emptiness is decidable for TAGC$[\approx, \not\approx, \mathbb{N}]$ works in two steps. In the first step (Proposition 4.3) we restrict ourselves to positive TAGC (let us recall that PTAGC denotes the class of positive TAGC$[\approx, \not\approx]$) and show that we can get rid of $|.|_{\mathbb{N}}$ and $\|.\|_{\mathbb{N}}$ without while keeping the same class of recognized languages. Then, in Proposition 4.6, we show how to get rid of negative constraints in TAGC$[\approx, \not\approx, \mathbb{N}]$.

Proposition 4.3 *For all PTAGC$[\approx, \not\approx, \mathbb{N}]$ $\mathcal{A}$, one can effectively construct a PTAGC $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$.*

The first step of this construction consists in rewriting conjunctions of constraints with $|.|_{\mathbb{N}}$ and $\|.\|_{\mathbb{N}}$ into disjunctions of conjunctions of simpler constraints.

Lemma 4.4 *Every conjunction φ of natural linear inequalities over Q can be effectively rewritten into an equivalent disjunction $\bigvee_{1 \leq i \leq n} \bigwedge_{q \in Q} p_{|.|}(q) \wedge p_{\|.\|}(q)$ for some $n \in \mathbb{N}$, where $p_{|.|}(q)$ (resp. $p_{\|.\|}(q)$) is either $|q| = a_q$ or $|q| \geq a_q$ for some $a_q \in \mathbb{N}$, or $\top$ (resp. $\|q\| = b_q$ or $\|q\| \geq b_q$ for some $b_q \in \mathbb{N}$, or $\top$).*

Proof. First, we separate the conjunction of inequalities in two parts: we note $\varphi_{|.|}$ the conjunction of inequalities on occurrences, and $\varphi\|.\|$ the conjunction of the ones on cardinality. So $\varphi = \varphi_{|.|} \wedge \varphi_{\|.\|}$. Then, we further separate the two parts into $\varphi_{|.|} = \chi_{|.|} \wedge \psi_{|.|}$ and $\varphi_{\|.\|} = \chi_{\|.\|} \wedge \psi_{\|.\|}$ where $\chi_{|.|}$ contains all the inequalities of the form $|q| = a$ or $|q| \geq a$ where q does not occur in another inequality of $\varphi_{|.|}$, and $\psi_{|.|}$ contains all the other inequalities. Note that each state occurring in $\chi_{|.|}$ occurs only in this subformula and does not occur in $\psi_{|.|}$ Respectively, $\chi_{\|.\|}$ is the conjunction of all the inequalities of $\varphi_{\|.\|}$ of the form $\|q\| = a$ or $\|q\| \geq a$ where q does not occur in another inequality, and $\psi_{\|.\|}$ contains all the other inequalities.

Let $\psi_{|.|} = \bigwedge_{1 \leq i \leq k} \sum_{q \in Q} a_q^i.|q| \leq a^i \bigwedge_{1 \leq j \leq l} \sum_{q \in Q} b_q^j.|q| \geq b^j$ for some k and l, with $a_q^i, a^i, b_q^j, b^j \in \mathbb{N}$ for all $i \leq k$, $j \leq l$. Let q be a state for which there exists $i \leq k$ such that $a_q^i \neq 0$ or there exists $j \leq l$ such that $b_q^j \neq 0$. Then, we do one of the following transformations;

1. If there exists some i such that $a_q^i \neq 0$, then, we define $s_q = \inf_{1 \leq i \leq k, a_q^i \neq 0} (\lfloor \frac{a^i}{a_q^i} \rfloor)$. If $|q| > s_q$, then there exists some inequality wich can not be satisfied. So in order to erase the occurrences of $|q|$ in the inequalities, we can make a case analysis on the values of $|q|$ between 0 and s_q We replace $\varphi_{|.|}$ by a disjunction $\bigvee_{0 \leq s \leq s_q} \wedge \varphi_{|.|}^s$, where $\varphi_{|.|}^s = \chi_{|.|}^s \wedge \psi_{|.|}^s$. We define $\chi_{|.|}^s = \chi_{|.|} \wedge |q| = s$ and $\psi_{|.|}^s$ the following way. First we replace every inequality $\sum_{q' \in Q} a_{q'}^i.|q'| \leq a^i$ by $\sum_{q' \in Q \setminus \{q\}} a_{q'}^i.|q'| \leq a^i - s.a_q^i$ and every inequality $\sum_{q' \in Q} b_{q'}^j.|q'| \geq b^i$ by $\sum_{q' \in Q \setminus \{q\}} b_{q'}^j.|q'| \geq \sup(0, b^j - s.b_q^j)$. Then, for every inequality with no occurrence of a state on the left-hand, side, which can be seen as an inequality between integers with a 0 on the left-hand side:

- if the inequality is true, then we erase it,
- if the inequality is false, then we delete the whole conjunction, which cannot be satisfied with the value given to $|q|$.

2. Otherwise, there exists some j such that $b_q^i \neq 0$ and we define $s_q = \sup_{1 \leq j \leq l, b_q^j \neq 0} (\lceil \frac{b^j}{b_q^j} \rceil)$. Note that if $|q| \geq s_q$ all the inequalities involving q are satisfied. So to erase the occurrences of $|q|$ we only have to do a case analysis on the value of $|q|$ between 0 and $s_q - 1$ and to erase all the inequalities with an occurrence of $|q|$ if $|q| \geq s_q$ We replace $\varphi_{|.|}$ by a disjunction $\bigvee_{0 \leq s \leq s_q} \varphi_{|.|}^s$ where $\varphi_{|.|}^s = \chi_{|.|}^s \wedge \psi_{|.|}^s$. We define $\chi_{|.|}^s = \chi_{|.|} \wedge q = s$ if $s < s_q$, $\chi_{|.|}^s = \chi_{|.|} \wedge q \geq s$ is $s = s_q$ and we define $\psi_{|.|}^s$ as above (except that there is no $a_q^i \neq 0$) if $s < s_q$ and $\psi_{|.|}^{s_q}$ as $\psi_{|.|}$ where every inequality $\sum_{q \in Q} b_q^j \geq b^j$ where $b_q^j \neq 0$ is deleted.

Note that, at this point of the procedure, q does not occur anymore in any $\psi_{|.|}^s$ and occurs only once in each $\chi_{|.|}^s$. While there is still a non-empty conjunction $\psi_{|.|}^s$, we apply the procedure to $\varphi_{|.|}^s$. At the end, we have an expression

equivalent to $\varphi_{|.|}$ that can be written $\bigvee_{1\leq i\leq n}\chi^i_{|.|}$ for some n where each $\chi^i_{|.|}$ is a conjunction of (in)equalities of the form $|q|=a$ or $|q|\geq a$.
We apply the same procedure to $\varphi_{\|.\|}$, which becomes a disjunction $\bigvee_{1\leq j\leq m}\chi^j_{\|.\|}$ for some n where each $\chi^j_{\|.\|}$ is a conjunction of (in)equalities of the form $\|q\|=a$ or $\|q\|\leq a$.

Hence, we have that ϕ is equivalent to $\bigvee_{1\leq i\leq n}\chi^i_{|.|}\bigwedge\bigvee_{1\leq j\leq m}\chi^j_{\|.\|}$. Since each q occurs at most once in each $\chi^i_{|.|}$ and once in each $\chi^j_{\|.\|}$, we can obtain what we are looking for by rewriting this expression in disjunctive normal form and adding a $\top$ symbol for each q than does not occur in $\chi^i_{|.|}$, and one for each that does not occur in $\chi^i_{\|.\|}$. □

Then, we only need to show that a TAGC with a conjunction of simple natural in(equalities) can be effectively transformed into a TAGC$[\approx,\not\approx]$.

Lemma 4.5 *Given a PTAGC$[\approx,\not\approx,\mathbb{N}]$ $\mathcal{A}$, whose arithmetic constraints are like in Lemma 4.4, one can effectively compute a PTAGC$[\approx,\not\approx]$ $\mathcal{A}'$ such that $L(\mathcal{A})=L(\mathcal{A}')$.*

Proof. The general idea is to have multiple copies of the states involved in natural (in)equality constraints, and to create $\approx$ and $\not\approx$ relations between them, to ensure that the wanted cardinality of the original state is reached by any acceptable run. Then we count the first occurrence of each state, to ensure that any acceptable run also have the correct number of occurrences of each original state.
Let $\mathcal{A}=\langle Q,\Sigma,F,C,\Delta\rangle$. First, we compute for each $q\in Q$ the number $c(q)$ of copies of this state that we need, depending on the constraints on q.

- if $p_{\|q\|}=\top$ then $c(q)=1$,
- if $p_{\|q\|}=(\|q\|=a_q)$ then $c(q)=a_q$, each state will recognize only one of the a_q different subterms,
- if $p_{\|q\|}=(\|q\|\geq a_q)$ then $c(q)=a_q+1$, the extra state will recognize the possible extra terms.

We construct now a positive TAGC$[\approx,\not\approx]$ $\mathcal{A}'=\langle Q',\Sigma,F',C',\Delta'\rangle$ recognizing $\mathcal{L}(\mathcal{A})$. Let us define $M=\max(\sup_{q,p_{\|.\|}(q)=(\|q\|=b_q)}(b_q),\ \sup_{q,p_{\|.\|}(q)=(\|q\|\geq b_q)}(b_q))$. Then we define a new set of copies of states of Q,

$$\overline{Q}=\bigcup_{q\in Q,c(q)>0}\{q^1,\ldots,q^{c(q)}\},$$

and we define Q' as $\overline{Q}\times\big(\overline{Q}\to\{0,\ldots,M+1\}\big)$. So a state of Q' is a pair $\langle q^i,\delta\rangle$ where $q\in Q$, $1\leq i\leq c(q)$, $q^i\in\overline{Q}$ and δ is a mapping $\delta:\overline{Q}\to\{0,\ldots,M+1\}$. We define the result of the addition m_1+m_2 of two elements $m_1,m_2\in\{0,\ldots,M+1\}$ in the following way:

- if $m_1=M+1$ or $m_2=M+1$, then $m_1+m_2=M+1$
- if $m_1\in\{0,\ldots,M\}$ and $m_2\in\{0,\ldots,M\}$, then m_1+m_2 is the result of the addition in $\mathbb{N}$ if it is lower or equals to M, otherwise $m_1+m_2=M+1$.

Then, we define the constraints, rules, and final states of $\mathcal{A}'$.
First, we have in C' all the constraints $\langle q^i_1,\delta_1\rangle\approx\langle q^i_2,\delta_2\rangle$ such that $(q_1\approx q_2)\in C$ and all the constraints $\langle q^i_1,\delta_1\rangle\not\approx\langle q^i_2,\delta_2\rangle$ such that $(q_1\not\approx q_2)\in C$. Then, for every state q occurring in a constraint $\|q\|=b_q$ or $\|q\|\geq b_q$ of C, we add to C' all the constraints $q^i\approx q^i$ for all $1\leq i\leq b_q$ and $q^i\not\approx q^j$ for all $1\leq i<j\leq b_q$.
Δ' is the set containing all the transition rules $f(\langle q^{i_1}_1,\delta_1\rangle,\ldots,\langle q^{i_n}_n,\delta_n\rangle)\to\langle q,\delta\rangle$ such that q_1,…, q_n, $q\in Q$, $f(q_1,\ldots,q_n)\to q\in\Delta$, for all $j\leq n$, $1\leq i_j\leq c(q_j)$, for all $q'\in Q'\setminus\{q\}$, $\delta(q')=\sum_{1\leq j\leq n}\delta_j(q')$, and $\delta(q)=1+\sum_{1\leq j\leq n}\delta_j(q)$.
F' is the set containing the states $\langle q^i_{\mathsf{f}},\delta\rangle$, such that $q_{\mathsf{f}}\in F$ and δ respect the followings conditions for all $q\in Q$ such that $c(q)>0$

- if $p_{|.|}(q)=(|q|=a_q)$ then $\sum_{1\leq i\leq c(q)}\delta(q^i)=a_q$,
- if $p_{|.|}(q)=(|q|\geq a_q)$ then $\sum_{1\leq i\leq c(q)}\delta(q^i)\geq a_q$ or $\sum_{1\leq i\leq c(q)}\delta(q^i)=M+1$,
- if $p_{\|q\|}=(\|q\|=b_q)$ or $p_{\|q\|}=(\|q\|\geq b_q)$, then $\forall 1\leq i\leq b_q,\delta(q^i)\geq 1$ or $\delta(q^i)=M+1$.

We now have to prove the correctness of this construction by showing that for all $t\in\mathcal{T}(\Sigma)$ there exists an accepting run r of $\mathcal{A}$ on t such that $r\models C$ iff there exists an accepting run r' of $\mathcal{A}'$ on t.
$\Rightarrow$: let r be an accepting run of $\mathcal{A}$ on t. First, we compute a term $\overline{r}$ on $\overline{Q}$ such that $Pos(\overline{r})=Pos(r)=Pos(t)$ and that for all $p\in Pos(t)$, $\overline{r}(p)$ is a copy $q^i\in\overline{Q}$ of the state $r(p)=q$. For every $q\in Q$, if $p_{\|.\|}(q)=\top$, then there is only one copy q^1 of q in $\overline{Q}$. So, for all p with $r(p)=q$ we define $\overline{r}(p)=q^1$. If $p_{\|.\|}(q)=(\|p\|=b_q)$ (resp. $p_{\|.\|}(q)=(\|p\|\geq b_q)$) there exists b_q (resp b_q+1) copies of q. Since $r\models\varphi$, there exists exactly (resp. at least) b_q different terms $t_1,\ldots,t_{b_q}$ such that there exists b_q positions $p_1,\ldots,p_{b_q}$ verifying $r(p_1)=\ldots=r(p_{b_q})=q$ and $t|_{p_1}=t_1,\ldots,t|_{p_{b_q}}=t_{b_q}$. For every position $p\in Pos(t)$ such that $r(p)=q$ and $t|_p=t_i$, we define $\overline{r}(p)=q^i$. And, if $p_{\|.\|}(q)=(\|p\|\geq b_q)$, for every $p\in Pos(t)$ such that $r(p)=q$ and $t|_p$ is different from all the $t_1,\ldots,t_{b_q}$, we define $r(p)=q^{b_q+1}$.

Then, we define r' in the following way: for all position $p\in Pos(t)$, $r'(p)=\langle\overline{r}(p),\delta_p\rangle$ where for all $\overline{q}\in\overline{Q}$, $\delta_p(\overline{q})=|\overline{q}|$ *wrt* to $\overline{r}|_p$ and $t|_p$ if $|\overline{q}|\leq M$ and $\delta_p(\overline{q})=M+1$ otherwise. It is easy to see that the rules of Δ' allow us to define such a run. We know have to check that the constraint C' is satisfied by r' and t.

This construction satisfies the atomic constraints $\langle q_1^i, \delta_1\rangle \approx \langle q_2^j, \delta_2\rangle$ (resp. $\langle q_1^i, \delta_1\rangle \not\approx \langle q_2^j, \delta_2\rangle$, that were added to C' because a constraint $q_1 \approx q_2$ (resp. $q_1 \not\approx q_2$) was already in C. Whatever copies q_1^i and q_1^j we have chosen in $\overline{r}$ to replace occurrences of q_1 and q_2 at positions p_1 and p_2 in r, we know that those constraints will be respected because C already implied that $t|_{p_1} \approx t_{p_2}$ (resp. $t|_{p_1} \not\approx t|_{p_2}$).

For every state $q \in Q$ such that $p_{\|.\|}(q) = (\|q\| = b_q)$, or $p_{\|.\|}(q) = (\|q\| \geq b_q)$ there are constraints of two types: $\langle q^i, \delta_1\rangle \not\approx \langle q^j, \delta_2\rangle$ for $1 \leq i < j \leq b_q$, for all δ_1, δ_2 and $\langle q^i, \delta_1\rangle \approx \langle q^i, \delta_2\rangle$ for $1 \leq i \leq b_q$ and for all δ_1, δ_2. By construction of r_B, $1 \leq i \leq b_q$ all the positions p such that $r_B(p) = \langle q^i, \delta\rangle$, for all δ, have the same subterm $t|_p = t_i$. Moreover, for every $1 \leq i < j \leq b_q$ we have that $t_i \neq t_j$. So both types of constraints are respected.

Know we have to make sure that r_B is an accepting run, that is $r_B(\varepsilon) \in F_B$. Since r is an accepting run of $\mathcal{A}$ we know that $r_B(\varepsilon) = \langle q_f^i, \delta\rangle$ where $q_f \in F$ and $\delta(q') = |q'|$ *wrt* r' if $|q'| \leq M$ and $|q'| =$ ' $>'$ otherwise. Since $q_f \in F$ we just have to make sure that δ satisfies the conditions. By construction of r_B, every occurrence of a state $q \in Q$ in r corresponds to an occurrence of one of the copies $q^i \in Q'$ in r'. So, for all $q \in Q$ we have that $|q|$ *wrt* r is equals to $\sum_{1\leq i\leq c(q)} |q^i|$ *wrt* r' and we have that $\sum_{1\leq i\leq c(q)} \delta(q^i)$ is either $|q|$ *wrt* r if $|q| \leq M$ or ' $>'$ otherwise. Hence, if $p_{|.|}(q) = (|q| = a_q)$ or $p_{|.|}(q) = (|q| \geq a_q)$, the associated conditions on δ are respected since φ (and in particular $p_{|.|}(q)$) is satisfied *wrt* r. By construction, for every $q \in Q$ such that $p_{\|.\|}(q) = (\|q\| = b_q)$ or $p_{\|.\|}(q) = (\|q\| \geq b_q)$ we have instantiated each copy q^i for all $1 \leq i \leq b_q$ at least once in r'. So the associated conditions are also respected: r_B is an accepting of B on t.

$\Leftarrow$: let r' be an accepting run of $\mathcal{A}'$ on t. We define a run r of $\mathcal{A}$ on t in the following way: $\forall p \in Pos(t)$, $r(p)$ is the original state $q \in Q$ corresponding to the copy q^i such that $r'(p) = \langle q^i, \delta\rangle$. By construction of $\mathcal{A}'$ we have that r is a valid run *wrt* Δ and that $r(\varepsilon) \in F$. For every positions $p_1, p_2 \in Pos(t)$ with $r(p_1) = q_1$, $r(p_2) = q_2$, such that there exists a constraint $q_1 \approx q_2$ (resp. $q_1 \not\approx q_2$), there was, by construction of $\mathcal{A}'$, a constraint $\langle q_1^i, \delta_1\rangle \approx \langle q_2^j, \delta_2\rangle$ (resp. $\langle q_1^i, \delta_1\rangle \not\approx \langle q_2^j, \delta_2\rangle$) for all copies q_1^i of q_1 and q_2^j of q_2 and for all δ_1, δ_2. In particular, there is in $\mathcal{A}'$ the constraint $r'(p_1) \approx r'(p_2)$ (resp. $r'(p_1) \not\approx r'(p_2)$), so $t|_{p_1} = t|_{p_2}$ (resp. $t|_{p_1} \neq t|_{p_2}$). Hence, all the constraints of $\mathcal{A}$ are satisfied by *wrt* r and t.

We now have to ensure that C is satisfied *wrt* r and t. As above, by construction of r, it is easy to see that for all $q \in Q$ we have that $\sum_{1\leq i\leq c(q)} \delta(q^i)$ is either $|q|$ *wrt* r if $|q| \leq M$ or $M+1$ otherwise. Hence, all expressions $p_{|.|}(q)$ of C are satisfied. And for every expression $p_{\|.\|}(q) = (\|q\| = b_q)$, we have exactly b_q copies of q in $\overline{Q}$. The constraints $\langle q^i, \delta_1\rangle \approx \langle q^i, \delta_2\rangle$ and $\langle q^i, \delta_1, \rangle \not\approx \langle q^j, \delta_2\rangle$ for all δ_1, δ_2, $i \neq j$ ensure that each copy q^i is used for only one subterm t_i and that, for all $1 \leq i < j \leq b_q$ $t_i \neq t_j$. Since all the copies q^i are guaranteed to occur at least once in r', by the arithmetic constraints, we know that we have exactly b_q different subterms at the positions labelled by q. And if $p_{\|.\|}(q) = (\|q\| \geq b_q)$, the extra copy q^{b_q+1} recognizes all the extra subterms that are recognized in q if $\|q\| > b_q$ is satisfied by r and t. Hence, C is fully satisfied by r and t, and r is a successful run of $\mathcal{A}$ on t. □

Proposition 4.6 *For all TAGC*$[\approx, \not\approx, \mathbb{N}]$ $\mathcal{A}$*, one can effectively construct a PTAGC*$[\approx, \not\approx, \mathbb{N}]$ $\mathcal{A}'$ *s.t.* $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$.

Proof. Let $\mathcal{A} = \langle Q, \Sigma, F, C, \Delta\rangle$. We can assume *wlog* that $C = C_+ \wedge C_- \wedge C_{\mathbb{N}}$ where C_+ is a conjunction of atomic constraints of the form $q \approx q'$ or $q \not\approx q'$, C_- is a conjunction of negations of atomic constraints of the form $\neg(q \approx q')$ or $\neg(q \not\approx q')$, and $C_{\mathbb{N}}$ is a conjunction of natural linear inequalities. Otherwise, we can put C is disjunctive normal form $C = \bigvee_{i=1}^n C_i$, where every C_i has the above form, apply the transformation below to each $\mathcal{A}_i = \langle Q, F, C_i, \Delta\rangle$, obtaining $\mathcal{A}'_i$ and let $\mathcal{A}'$ be the disjoint union of the $\mathcal{A}'_i$'s.

We construct below a positive TAGC$[\approx, \not\approx, \mathbb{N}]$ $\mathcal{A}' = \langle Q', \Sigma, F', C', \Delta'\rangle$ recognizing $\mathcal{L}(\mathcal{A})$. The idea for the construction of $\mathcal{A}'$ is to replace the negative constraints of C_- by positive constraints and arithmetic constraints, using copies of states of Q. For instance, assume that C_- contains only $\neg(q_1 \approx q_2)$. By definition, for every successful run r of $\mathcal{A}$ on a term $t \in \mathcal{T}(\Sigma)$, there exist two positions $p_1, p_2 \in Pos(t)$ such that $r(p_1) = q_1$, $r(p_2) = q_2$ and $t|_{p_1} \neq t|_{p_2}$. For expressing this property without the negative constraint, we add two copies q'_1, q'_2 of respectively q_1 and q_2 (*i.e.* we let $Q' = Q \cup \{q'_1, q'_2\}$) and define Δ' by adding to Δ all the transitions obtained from Δ by replacing at least one occurrence of q_1 by q'_1 or one occurrence of q_2 by q'_2. We define similarly C'_+ and $C'_{\mathbb{N}}$ from C'_+ and $C_{\mathbb{N}}$. Also let $F' = F \cup \{q'_i \mid q_i \in F, i = 1, 2\}$. Finally, we let $C' = C'_+ \wedge q'_1 \not\approx q'_2 \wedge C'_{\mathbb{N}} \wedge |q'_1| = 1 \wedge |q'_2| = 1$.

In general, when C_- contains several negative constraints, we have to take care of the multiple occurrences of states in the different negative constraints of C_-. For instance, let $C_- = \neg(q_1 \approx q_2) \wedge \neg(q_1 \approx q_3)$. For a successful run r of $\mathcal{A}$ on t, there exist positions $p_1^1, p_2, p_1^2, p_3 \in Pos(t)$ such that $r(p_1^1) = r(p_1^2) = q_1$, $r(p_2) = q_2$, $r(p_3) = q_3$ and and $t|_{p_1^1} \neq t|_{p_2}$ and $t|_{p_1^2} \neq t|_{p_3}$. In order to replace C_- by positive constraints as above, we must decide how many copies of q_1, q_2 and q_3 we will need. If $p_1^1 \neq p_1^2$, then we can choose two copies q_1^1 and q_1^2 of q_1, that will reach respectively p_1^1 and p_1^2 in a computation of $\mathcal{A}'$ on t. With constraints $q_1^1 \not\approx q_1^2$, $|q_1^1| = 1$ and $|q_1^2| = 1$ as above, we are done. However, if $p_1^1 = p_1^2$, then we must have only one copy of q_1, because only one state can reach this position in a computation of $\mathcal{A}'$ on t. We shall enumerate below all the

cases of the number of copies needed for each state, using the number of finite models of a first-order formula.

Let $Q = \{q_1, \ldots, q_p\}$ and let $C_- = d_1 \wedge \ldots \wedge d_m \wedge e_1 \wedge \ldots \wedge e_n$ where for all $k \leq m$, $d_k = \neg(q_{u_k} \approx q_{u'_k})$ and for all $\ell \leq n$, $e_\ell = \neg(q_{v_\ell} \not\approx q_{v'_\ell})$. Let us associate the following closed first order formula to $\mathcal{A}$:

$$\begin{array}{rcl} \phi_{\mathcal{A}} & = & \exists x_{u_1}^{d_1}, y_{u'_1}^{d_1}, \ldots x_{u_m}^{d_m}, y_{u'_m}^{d_m}, x_{v_1}^{e_1}, y_{v'_1}^{e_1}, \ldots x_{v_m}^{e_n}, y_{v'_n}^{e_n}. \\ & & \bigwedge_{k=1}^{m} x_{u_k}^{d_k} \neq y_{u'_k}^{d_k} \wedge \bigwedge_{\ell=1}^{n} x_{v_\ell}^{e_\ell} \neq y_{v'_\ell}^{e_\ell} \end{array}$$

Let N be the number of variables in $\phi_{\mathcal{A}}$. Every variable of $\phi_{\mathcal{A}}$ is uniquely associated to an occurrence of a state of Q in C_-. For instance, $x_{u_k}^{d_k}$ (resp. $y_{u'_k}^{d_k}$) is associated to the occurrence of q_{u_k} is the left (resp. right) hand side of d_k. For a $\mathcal{A}$ with the above example of C_-, we have $\phi_{\mathcal{A}} = \exists x_1^{d_1}, y_2^{d_1}, x_1^{e_1}, y_3^{e_1}, x_1^{d_2}, y_4^{d_2} \,.\, x_1^{d_1} \neq y_2^{d_1} \wedge x_1^{e_1} = y_3^{e_1} \wedge x_1^{d_2}, y_4^{d_2}$.

For each $1 \leq i \leq p$, we define X_i as the set of variables associated to occurrences of q_i: $X_u = \{x_{u_k}^{d_k} \mid 1 \leq k \leq m, u_k = i\} \cup \{y_{u'_k}^{d_k} \mid 1 \leq k \leq m, u'_k = i\} \cup \{x_{v_\ell}^{e_\ell} \mid 1 \leq \ell \leq n, v_\ell = i\} \cup \{y_{v'_\ell}^{e_\ell} \mid 1 \leq \ell \leq n, v'_\ell = i\}$.

It is clear that $\phi_{\mathcal{A}}$ is satisfiable iff it admits a finite model with at most N elements. Let D be a set of N distinct elements. A solution of $\phi_{\mathcal{A}}$ is a mapping σ from $\{x_{u_1}^{d_1}, y_{u'_1}^{d_1}, \ldots\}$ into D which makes true the conjunction of $\phi_{\mathcal{A}}$. Let $sol(\phi_{\mathcal{A}})$ be the set of solutions of $\phi_{\mathcal{A}}$. To each $\sigma \in sol(\phi_{\mathcal{A}})$, we associate a PTAGC$[\approx, \not\approx, \mathbb{N}]$ $\mathcal{A}_\sigma = \langle Q_\sigma, F_\sigma, C_\sigma, \Delta_\sigma \rangle$, where Q_σ contains the states of Q and, for each $i \leq p$, $n_i = |\sigma(X_i)|$ copies of q_i: $q_i^1, \ldots, q_i^{n_i}$. The transition set Δ_σ contains all the transitions of Δ plus additional transitions obtained from Δ by replacing at least one occurrence of one of the q_i by one of its copies q_i^j. The final states set is defined by $F_\sigma = \{q_i^j \mid q_i^j \in Q_\sigma, q_i \in F\}$. Finally, we let

$$C_\sigma = C_+^\sigma \wedge C_{\mathbb{N}}^\sigma \wedge \bigwedge_{\substack{i \leq p \\ j \leq n_i}} \left(|q_i^j| = 1 \wedge \bigwedge_{\substack{j' \leq n_i \\ j' \neq j}} q_i^j \not\approx q_i^{j'} \right)$$

where C_+^σ contains all the positive constraints of C_+ where some (possibly zero or more) instances of states of Q are replaced by copies, and $C_{\mathbb{N}}^\sigma$ is obtained from $C_{\mathbb{N}}$ by replacement of every $|q_i|$ by $|q_i| + |q_i^1| + \ldots + |q_i^{n_i}|$ and of every $\|q_i\|$ by $\|q_i\| + \|q_i^1\| + \ldots + \|q_i^{n_i}\|$. Finally, let us rename the respective states of the $\mathcal{A}_\sigma$ such that there states sets are pairwise disjoint, and let $\mathcal{A}'$ be the disjoint union of all the $\mathcal{A}_\sigma$. It is clear that $\mathcal{A}'$ is positive. Let us show that $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$.

Let $t \in \mathcal{L}(\mathcal{A}')$. By construction, there exists a solution $\sigma \in sol(\phi_{\mathcal{A}})$ such that $t \in \mathcal{L}(\mathcal{A}_\sigma)$. Let r' be a successful run of $\mathcal{A}_\sigma$ on t. By definition of C_σ, for each $i \leq p$, there exists $n_i = |\sigma(X_i)|$ copies of q_i: $q_i^1, \ldots, q_i^{n_i}$, and each copy q_i^j occurs exactly once in r' at a position called p_i^j (*i.e.* $r^{-1}(q_i^j) = \{p_i^j\}$). Moreover, still by definition of C_σ, the subterms $t|_{p_i^j}$ are pairwise disjoint. The mapping θ which associate to each variable in X_i the corresponding position p_i^j is isomorphic to σ. Let r be obtained from r' by replacement of every subrun $r'|_{p_i^j}$ by a run of $ta(\mathcal{A})$ headed by q_i. It is clear by construction of Δ_σ that r is a run of $ta(\mathcal{A})$. Moreover, following the property of θ above, $t, r \models C_-$ and $t, r \models C_+ \wedge C_{\mathbb{N}}$. It follows that r is a successful run of $\mathcal{A}$ on t.

Conversely, let $t \in \mathcal{L}(\mathcal{A})$, and let r be a successful run of $\mathcal{A}$ on t. By definition, for every $d_k = \neg(q_{u_k} \approx q_{u'_k})$ in C_-, there exists two positions $p_{u_k}^{d_k}$ and $s_{u_k}^{d_k}$ in $Pos(t)$ such that $t|_{p_{u_k}^{d_k}} \neq t|_{s_{u'_k}^{d_k}}$, and for every $e_\ell = \neg(q_{v_\ell} \not\approx q_{v'_\ell})$ in C_- there exists two positions $p_{v_\ell}^{e_\ell}$ and $s_{v'_\ell}^{e_\ell}$ in $Pos(t)$ such that $t|_{p_{v_\ell}^{e_\ell}} = t|_{s_{v'_\ell}^{e_\ell}}$. The set of subterms of t at these positions define therefore a solution σ of $\phi_{\mathcal{A}}$. We can construct from r a run successful run r' of $\mathcal{A}_\sigma$ on t. The principle of the construction is to replace every subrun of r at the position *e.g.* $p_{u_k}^{d_k}$ by a run of $\mathcal{A}_\sigma$ headed by a copy of q_{u_k}. Therefore, $t \in \mathcal{L}(\mathcal{A}_\sigma) \subseteq \mathcal{L}(\mathcal{A}')$. □

Lemma 3.1 in Section 3 is a consequence of Propositions 4.3 and 4.6. Indeed, given a TAGC$[\approx, \not\approx]$ $\mathcal{A}$, Proposition 4.6 permits to construct a PTAGC$[\approx, \not\approx, \mathbb{N}]$ $\mathcal{A}'$ with $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$ and then Proposition 4.3 permits to construct a PTAGC $\mathcal{A}''$ with $\mathcal{L}(\mathcal{A}'') = \mathcal{L}(\mathcal{A})$. Let $\mathcal{A}'' = \langle Q'', \Sigma, F'', C'', \Delta'' \rangle$, and let us put the constraint C'' of $\mathcal{A}''$ in disjunctive normal form $C_1 \vee \ldots \vee C_n$ where every C_i, $i \leq n$ is a conjunction of atomic constraints $q \approx q'$, $q \not\approx q'$. The PCTAGC $\mathcal{A}_i = \langle Q'', \Sigma, F'', C_i, \Delta'' \rangle$, for $1 \leq i \leq n$, is such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cup \ldots \cup L(\mathcal{A}_n)$.

Another consequence is the decidability of emptiness for TAGC extended with natural arithmetic constraints.

Theorem 4.7 *Emptiness is decidable for TAGC*$[\approx, \not\approx, \mathbb{N}]$.

Proof. Given a TAGC$[\approx, \not\approx, \mathbb{N}]$ $\mathcal{A}$, we can construct as above, using Proposition 4.6 and Proposition 4.3, n PCTAGC $\mathcal{A}_1, \ldots, \mathcal{A}_n$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \cup \ldots \cup L(\mathcal{A}_n)$. Then we can apply Theorem 3.21 to each $\mathcal{A}_i$, $1 \leq i \leq n$, in order to decide the emptiness of $\mathcal{L}(\mathcal{A})$. □

4.2 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 [4] 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 like for 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.

Example 4.8 *Assume that the terms of Example 2.4 are now used to record the activity of a restaurant, and that we add a second argument of type q_t to L_0, L and M, so that the first argument q_t will characterize the theoretical time to cook, and the second q_t will characterize the real time that was needed to cook the dish. Let us replace the transitions with L_0, L and M in input by $L_0(q_{id}, q_t, q_t) \xrightarrow{2=3} q_L$, $L_0(q_{id}, q_t, q_t) \xrightarrow{2\neq 3} q'_L$, $L(q_{id}, q_t, q_t, q_L) \xrightarrow{2=3} q_L$, $L(q_{id}, q_t, q_t, q_L) \xrightarrow{2\neq 3} q'_L$, $M(q_{id}, q_t, q_t, q_L) \xrightarrow{2=3} q_M$, $M(q_{id}, q_t, q_t, q_L) \xrightarrow{2\neq 3} q'_M$, where q'_L is a new state meaning that there was an anomaly. We also also add a transition $L(q_{id}, q_t, q_t, q'_L) \to q'_L$ to propagate q'_L and $M(q_{id}, q_t, q_t, q'_L) \to q'_M$.*

The language of the TAGC obtained is the set of records well cooked, i.e. such that for all dishes, the real time to cook is equal to the theoretical time. $\Diamond$

The emptiness decision algorithm of Section 3 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 3.10 and Theorem 4.7.

Theorem 4.9 *Emptiness is decidable for TACB$[\approx, \not\approx, \mathbb{N}]$.*

4.3 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 child is arbitrary and does not depend on a. Let us denote $\mathcal{U}(\Sigma)$ the set of unranked ordered terms over Σ. The notions of positions, subterms *etc* are defined for unranked terms of $\mathcal{U}(\Sigma)$ like for ranked terms of $\mathcal{T}(\Sigma)$.

We extend the definition of automata for unranked ordered terms, called hedge automata [22], 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 2.2 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.

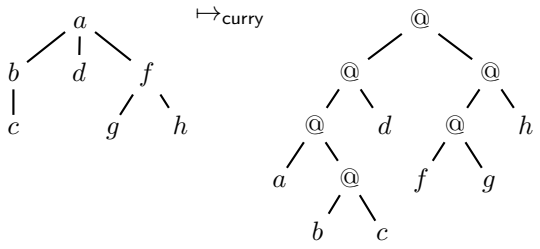

Figure 6. Currying of an unranked term

A *run* of $\mathcal{A}$ on an unranked ordered $t \in \mathcal{U}(\Sigma)$ is a function r from $Pos(t)$ into Q such that for all position $p \in Pos(t)$ with n child, there exists a transition $t(p)(L) \to r(p)$ in Δ such that $r(p.1) \ldots r(p.n) \in L$. The run r is called *successful* if moreover $r(\Lambda) \in F$ and $t, r \models C$, where satisfiability is defined like in Section 2.2. As above, we use the notation HAGC$[\tau_1, \ldots, \tau_n]$ where the τ_i's can be $\approx$, $\not\approx$, $|.|_{\mathbb{N}}$, $\|.\|_{\mathbb{N}}$...

The emptiness decision results of Corollary 3.22 can be extended from TAGC to HAGC using a standard transformation from unranked to ranked binary terms. Let us associate to the unranked signature Σ the (ranked) signature $\Sigma_@ := \{a : 0 \mid a \in \Sigma\} \cup \{@ : 2\}$ where @ is a new symbol. The following operator curry defines a bijection from $\mathcal{U}(\Sigma)$ into $\mathcal{T}(\Sigma_@)$: $\mathsf{curry}(a) := a$ and $\mathsf{curry}\big(a(t_1, \ldots, t_n)\big) := @\big(\mathsf{curry}\big(a(t_1, \ldots, t_{n-1})\big), \mathsf{curry}(t_n)\big)$. An example of application of this operator is presented in Figure 6.

Proposition 4.10 *For all HAGC$[\approx, \not\approx, \mathbb{N}]$ $\mathcal{A}$ over Σ, on can construct effectively in PTIME a TAGC$[\approx, \not\approx, \mathbb{N}]$ $\mathcal{A}'$ over $\Sigma_@$ such that $\mathcal{L}(\mathcal{A}') = \mathsf{curry}\big(\mathcal{L}(\mathcal{A})\big)$.*

Proof. Let $\mathcal{A} = \langle Q, \Sigma, F, C, \Delta \rangle$. We call $ha(\mathcal{A})$ the underlying hedge automaton of $\mathcal{A}$: $ha(\mathcal{A}) = \langle Q, \Sigma, F, true, \Delta \rangle$ *i.e.* $ha(\mathcal{A})$ computes the same way as $\mathcal{A}$ but without the global constraints (note however that the top state of successful runs must be a final state).

We can assume *wlog* that Δ contains at most one transition $a(L) \to q$ for each pair $\langle a, q \rangle \in \Sigma \times Q$. Otherwise, we can replace two transition $a(L) \to q$ and $a(L') \to q$ by the unique $a(L \cup L') \to q$, preserving the language recognized. Let $B_{a,q}$ be the NFA recognizing L in the (unique) transition $a(L) \to q$ of Δ associated to a and q. Note that such an automaton has Q as input alphabet. Let P be the union (assumed disjoint) for all transitions in Δ of the state sets of the automata $B_{a,q}$.

The transitions of the automaton $\mathcal{A}'$, when computing on $\mathsf{curry}(t)$ for some $t \in \mathcal{U}(\Sigma)$, will simulate both the transitions of $\mathcal{A}$ (vertical transitions) and the transitions of the NFAs $B_{a,q}$ (horizontal transitions).

Let $\mathcal{A}' = \langle Q \cup P, \Sigma, F, C, \Delta' \rangle$ where Δ' contains the transitions: $a \to q$ if B_a, q recognizes the empty word, $a \to init_{a,q}$ for all $q \in Q$, where $init_{a,q}$ is the initial state of $B_{a,q}$, $@(p,q) \to p'$ if there is a transition $p \xrightarrow{q} p'$ in some $B_{a',q'}$, and $@(p,q) \to q'$ if there is a transition $p \xrightarrow{q} p'$ in some $B_{a',q'}$, where p' is a final state.

We can show by induction on $t \in \mathcal{U}(\Sigma)$ that $t \in L\big(ha(\mathcal{A})\big)$ iff $\mathsf{curry}(t) \in L\big(ta(\mathcal{A}')\big)$. More precisely, for all $t \in \mathcal{U}(\Sigma)$, to every run r of $ha(\mathcal{A})$ on t, we can associate a run r' of $ta(\mathcal{A}')$ on $\mathsf{curry}(t)$ with $r'(\Lambda) = r(\Lambda) \in F$, and reciprocally. Moreover, by construction, there exists an injective mapping θ from $Pos(t)$ into $Pos(\mathsf{curry}(t))$, such that for all $p \in Pos(t)$, the states $r(p)$ and $r'(\theta(p))$ coincide and $\mathsf{curry}(t|_p) = \mathsf{curry}(t)|_{\theta(p)}$. It follows that r is successful for $\mathcal{A}$ iff r' is successful for $\mathcal{A}'$. □

Theorem 4.11 *Emptiness is decidable for HAGC*$[\approx, \not\approx, \mathbb{N}]$.

5 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 $X \ldots$ 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 that the maximal arity of symbols of Σ, which is true on every pair $(p, p.i)$, (we write $+1$ for 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. $t|_x \neq t|_y$,
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 $\{t|_p \mid p \in X_i\}$.

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 $\tau_1 \ldots$, amongst the above types $+1$, $\approx$, $\not\approx$, and $|.|_{\mathbb{Z}}$, $\|.\|_{\mathbb{Z}}$ (3-5). We also use the notations $|.|_{\mathbb{N}}$ and $\|.\|_{\mathbb{N}}$ for natural linear inequalities and the abbreviations $\mathbb{Z}$ and $\mathbb{N}$ like in Section 4.1. 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 $\{X_1, \ldots, X_n\}$.

Example 5.1 *The formula of EMSO*$[\approx, \not\approx]$, $\exists X_a\, (\forall x\, X_a(x) \leftrightarrow a(x)) \wedge X_a \not\approx X_a$ *expresses that all the subterms headed by* a *in a term* t *are pairwise different. It other words,* a *is used to mark monadic keys in* t *(see Example 2.4). This can also be expressed by* $\|X_a\| \leq 1$.

It is well known that MSO$[+1]$ has exactly the same expressiveness as TA [27] and therefore it is decidable. The extension MSO$[\approx, +1]$ is undecidable, see *e.g.* [13], as well as MSO$[\,|.|_{\mathbb{Z}}, +1]$ [19] (the latter extension is also undecidable for unranked ordered terms when counting constraints are applied between childrens of positions [25]) but the fragment EMSO$[\,|.|_{\mathbb{Z}}, +1]$ is decidable [19].

In [14] 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$[\approx, \not\approx, \mathbb{N}, +1]$ and TAGC$[\approx, \not\approx, \mathbb{N}]$. With Theorem 4.7, it gives the following result.

Theorem 5.2 *EMSO*$[\approx, \not\approx, |.|_{\mathbb{N}}, \|.\|_{\mathbb{N}}, +1]$ *is decidable on ranked terms.*

Proof. We show that for every formula ϕ in EMSO$[S, \approx, \not\approx, \mathbb{N}]$, we can construct a TAGC$[\approx, \not\approx, \mathbb{N}]$ recognizing exactly the set of models of ϕ. The decidability of the logic follows then from Theorem 4.7. □

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 are not sufficient. In order 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 2 predicates only.

Using the results of Section 4.3, we can generalize Theorem 5.2 to EMSO over unranked ordered terms.

Theorem 5.3 *EMSO*$[\approx, \not\approx, |.|_{\mathbb{N}}, \|.\|_{\mathbb{N}}, +1]$ *is decidable on unranked ordered terms.*

6 Conclusion

We have answered (positively) the open problem of decidability of the emptiness problem for the TAGEDs [14], 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 3 appeared to be robust enough to deal with several extensions like global counting constraints, local test 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 4.3 with the unranked tree automata with tests between sibling [29, 20].

A challenging question would be to investigate the precise complexity of the problem, avoiding the use of the Higman's Lemma in the algorithm. For instance, in [14], it is shown, using a direct reduction into solving positive and negative set constraints [8, 16, 26], that emptiness is decidable in NEXPTIME for TAGC without constraints of the form $q \approx q'$, and with constraints of the form $q \not\approx q'$ for distinct states q and q'. On the other hand, the best known lower bound for emptiness decision for TAGC is EXPTIME-hardness (already for positive TAGC[$\approx$] as shown in [14]).

Another branch of research related to TAGC concerns automata and logics for *data trees*, *i.e.* trees labeled over an infinite (countable) alphabet (see [24] 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 to study in order to relate our results on TAGC to decidability results on automata or logics on data trees like those in [18, 5].

References

[1] S. Anantharaman, P. Narendran, and M. Rusinowitch. Closure properties and decision problems of dag automata. *Inf. Process. Lett.*, 94(5):231–240, 2005.

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

[3] 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`.

[4] 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*, volume 577 of *LNCS*, pages 161–171. Springer, 1992.

[5] M. Bojanczyk, A. Muscholl, T. Schwentick, and S. Luc. Two-variable logic on data trees and applications to xml reasoning. *JACM*, 56(3), 2009. A preliminary version was presented at PODS 06.

[6] A. Bouajjani and T. Touili. On computing reachability sets of process rewrite systems. In J. Giesl, editor, *Term Rewriting and Applications, 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005, Proceedings*, volume 3467 of *Lecture Notes in Computer Science*, pages 484–499. Springer, 2005.

[7] W. Charatonik. Automata on dag representations of finite trees. Technical Report Technical Report MPI-I-99-2-001, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1999.

[8] W. Charatonik and L. Pacholski. Set constraints with projections are in NEXPTIME. In *Proceedings of the 35^{th} Symp. Foundations of Computer Science*, pages 642–653, 1994.

[9] H. Comon and V. Cortier. Tree automata with one memory, set constraints and cryptographic protocols. *Theoretical Computer Science*, 331(1):143–214, Feb. 2005.

[10] 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.

[11] S. Dal Zilio and D. Lugiez. Xml schema, tree logic and sheaves automata. *Journal Applicable Algebra in Engineering, Communication and Computing*, 17(5):337–377, 2006.

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

[13] 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)*, volume 4646 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2007.

[14] E. Filiot, J.-M. Talbot, and S. Tison. Tree automata with global constraints. In *12th International Conference in Developments in Language Theory (DLT 2008)*, volume 5257 of *Lecture Notes in Computer Science*, pages 314–326. Springer, 2008.

[15] J. H. Gallier. What's so special about kruskal's theorem and the ordinal gamma$_0$? a survey of some results in proof theory. *Annals of Pure Applied Logic*, 53(3):199–260, 1991.

[16] 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*, volume 845 of *LNCS*, pages 336–351. Spinger, 1994.

[17] F. Jacquemard, F. Klay, and C. Vacher. Rigid tree automata. In A. Horia Dediu, A. Mihai Ionescu, and C. Martín-Vide, editors, *Proceedings of the 3rd International Conference on Language and Automata Theory and Applications (LATA'09)*, volume 5457 of *Lecture Notes in Computer Science*, pages 446–457, Tarragona, Spain, Apr. 2009. Springer.

[18] M. Jurdzinski and R. Lazic. Alternation-free modal mu-calculus for data trees. In *Proceedings 22nd IEEE Symposium on Logic in Computer Science (LICS)*, pages 131–140. IEEE Computer Society, 2007.

[19] F. Klaedtke and H. Ruess. Parikh automata and monadic second-order logics with linear cardinality constraints. Technical Report 177, Intitute of Computer Science at Freiburg University, 2002.

[20] 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)*, Leibniz International Proceedings in Informatics. Schloss Dagstuhl - Leibniz Center for Informatics, 2009.

[21] J. Mongy. *Transformation de noyaux reconnaissables d'arbres. Forêts RATEG*. PhD thesis, Laboratoire d'Informatique Fondamentale de Lille, Université des Sciences et Technologies de Lille, Villeneuve d'Ascq, France, 1981.

[22] M. Murata. Hedge automata: a formal model for XML schemata. Technical report, Fuji Xerox INformation Systems, 1999.

[23] T. Schwentick. Automata for xml - a survey. *J. Comput. Syst. Sci.*, 73(3):289–315, 2007.

[24] L. Segoufin. Automata and logics for words and trees over an infinite alphabet. In *Computer Science Logic*, volume 4207 of *LNCS*. Springer, 2006.

[25] H. Seidl, T. Schwentick, and A. Muscholl. Numerical document queries. In *Principle of Databases Systems (PODS)*, pages 155–166. ACM Press, 2003.

[26] K. Stefansson. Systems of set constraints with negative constraints are nexptime-complete. In *Proceedings, Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 137–141. IEEE Computer Society Press, 1994.

[27] 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*, 2:57–82, 1968.

[28] K. N. Verma and J. Goubault-Larrecq. Alternating two-way ac-tree automata. *Inf. Comput.*, 205(6):817–869, 2007.

[29] 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)*, volume 4596 of *LNCS*, pages 875–887. Springer, 2007.