The Emptiness Problem for Tree Automata with Global Constraints

Luis Barguñó  Carles Creus  Guillem Godoy
Universitat Politècnica de Catalunya
Barcelona

Florent Jacquemard  Camille Vacher
INRIA Saclay,
LSV-CNRS/ENS Cachan
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 subterms, 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-
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 subtrees 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 most of 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)equation 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 $\Sigma$ is a finite set of function symbols with arity. We sometimes denote $\Sigma$ 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 $\Sigma$ of arity $m$ as $\Sigma_m$. The set of (ranked) terms over the signature $\Sigma$ is defined recursively as $T(\Sigma) := \{ f \mid f : 0 \in \Sigma \} \cup \{ f(t_1, \ldots, t_m) \mid f : m \in \Sigma, t_1, \ldots, t_m \in T(\Sigma) \}$. Positions in terms are denoted by sequences of natural numbers. With $\Lambda$ 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 $Pos(t) := \{ \lambda \} \cup \{ i.p \mid i \in \{ 1, \ldots, m \} \land p \in Pos(t_i) \}$. A term $t \in T(\Sigma)$ can be seen as a function from its set of positions $Pos(t)$ into $\Sigma$. 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 $p' - p = p''$. 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 $t[p]_p$, is defined recursively as $t[p]_p = t$ and $f(t_1, \ldots, t_m)[i.p]_p = t[i.p]_p$. The replacement in $t$ of the subterm at position $p$ by $s$, denoted $t[p]_p = s$, is defined recursively as $t[s]_\lambda = s$ and $f(t_1, \ldots, t_{i-1}, t_i, t_{i+1}, \ldots, t_m)[i]_p = f(t_1, \ldots, t_{i-1}, t_i[s], t_{i+1}, \ldots, t_m)$. The fact $t \equiv t[s]_p$ may also be used to emphasis that $t[p]_p = s$. The height of a term $t$, denoted $h(t)$, is the maximal length of a position of $Pos(t)$. In particular, the length of $\Lambda$ is 0.

A tree automaton (TA, see e.g. [10]) is a tuple $A = (Q, \Sigma, F, \Delta)$ where $Q$ is a finite set of states, $\Sigma$ is a signature, $F \subseteq Q$ is the subset of final states and $\Delta$ is a set of transitions rules of the form $f(q_1, \ldots, q_m) \rightarrow q$ 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 \( r = (t, M) \) where \( t \) is a term in \( T(\Sigma) \) and \( M : Pos(t) \to Q_{\mathcal{A}} \) is a mapping satisfying, for all \( p \in Pos(t) \), that the rule \( t(p)(M(p,1), \ldots, M(p,m)) = M(p) \) is in \( \Delta_{\mathcal{A}} \), where \( m \) is the arity of the symbol \( t(p) \) in \( \Sigma \). 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 \( \text{term}(r) \) we refer to \( t \), and by \( \text{symbo}(r) \) 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 \( r = (t, M) \), 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 \( r' = (t', M') \), 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 \( M[M'|_p(p,p') = M'(p') \) for each \( p' \) holding \( p \neq p' \).

A well quasi-ordering \[ \leq \] 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 \( \Sigma \) 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 \( q \not= q' \), where \( q, q' \in Q \). A TAGC \( \mathcal{A} \) is called positive if \( C_{\mathcal{A}} \) is a conjunction of atomic constraints. A TAGC \( \mathcal{A} \) is called positive conjunctive if \( C_{\mathcal{A}} \) is a conjunction of conjunctive 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 \( t \in \text{term}(r) \).

For atomic constraints, \( r \models q \approx q' \) (respectively \( r \models q \not= q' \)) 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 \( t \in T(\Sigma) \) 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 \( \mathcal{A} \).

It is important to note that the semantics of \( \neg (q \approx q') \) and \( q \not= q' \) differ, as well as the semantics of \( \neg (q \not= q') \) 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 \( \{f(t,t) \mid t \in T(\Sigma)\} \) is not a regular tree language (this can be shown using a classical pumping argument).

However, it is recognized by the following TAGC \( \mathcal{A} = \{\langle q_0, q_1, q_1 \rangle, \Sigma, \{q_1 \}, q_1 \approx q_1, \langle a \to q_0, q_1, f(q_0, q_0) \to q_0, q_1, f(q_1, q_1) \to q_1 \rangle \} \), where \( a \to q_0, q_1 \) is an abbreviation for \( a \to q \) and \( a \to q \). An example of successful run of \( \mathcal{A} \) on \( t = f(f(a,a), f(a,a)) \) is \( q_1(q_1, q_0, q_0, q_1, q_0, q_0) \). 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 \( q \not= q' \), with additional restrictions. In particular \( q \) and \( q' \) are required to be distinct in \( q \not= q' \) for TAGEDs. Reflexive disequality constraints such as \( q \not= q \) 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 \( T(\Sigma) \) 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 pair wise distinct. It is recognized by following the TAGC \( \{\langle q_0, q, q_1 \rangle, \Sigma, \{q_1 \}, q \not= q, \langle 0 \to q_0, q_0, s(q_0) \to q_0, q, f(q_0, q_1) \to q_1 \rangle \} \). However, \( L \) cannot be recognized by a PTAGC without a reflexive constraint of the form \( q \not= q \).

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 \( r' = r|_{p_i} \) is an accepting run of \( \mathcal{A} \) on \( t' = t[p_i] \). Since \( r(p_i) = r(p_j) \) and \( r \) is a run of \( \mathcal{A} \) on \( t \) by replacing \( t|_{p_i} \) with \( t|_{p_1} \) and \( r|_{p_i} \) with \( r|_{p_1} \), \( r' \) is a run of \( \mathcal{A} \) 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
If \( r'(p) \cong r'(p') \) is in \( C_1 \) (resp. \( r'(p) \not\cong r'(p') \) is in \( C_1 \)), we know that \( r \models r(p) \cong r(p') \) (resp. \( r \models r(p) \not\cong r(p') \)), so the constraints are respected in \( t \), hence also in \( t' \) since the positions \( p \) and \( p' \) are referring to common sub-terms of \( t \) and \( t' \).

If a position \( p = 2^{-1} \cdot 1 \cdot v \) is involved in some constraint, let \( p' = 2^{-1} \cdot 1 \cdot v \). By construction, we have \( r'|_{p} = r'|_{p'} \) and \( t'|_{p} = t'|_{p'} \). Due to this last equality, any constraint involving \( p \) is satisfied if 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 \( 2^{-1} \cdot 1 \). It is clear that every subterm of \( t \) at a position \( 2^k \), for \( 0 \leq k \leq |Q| \), is unique, so every subterm at such a position can only satisfy a disequality constraint or an equality with itself (in that case \( r(2^k) \) 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 \( 2^k \), and hence the subterm at position \( p \), are unique, and satisfy all the disequalities. So \( t' \) is recognized by \( A \) 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 \( A = (Q, \Sigma, F, C, \Delta) \) is defined as follows: \( Q = \{0, \ldots, 9 : 0, N, L_0 : 2, L, M : 3\}, \Sigma = \{q_d, q_N, q_{id}, q_t, q_L, q_M\}, F = \{q_M\}, \) and \( \Delta = \{(i \rightarrow q_d|q_N|q_{id}|q_t \mid 0 \leq i \leq 9) \cup \{N(q_d, q_N) \rightarrow q_N|q_{id}|q_t, L_0(q_{id}, q_t) \rightarrow q_L, L(q_{id}, q_t, q_L) \rightarrow q_L, M(q_{id}, q_t, q_L) \rightarrow q_M\}\}. \) 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: \( C = q_{id} \neq q_{id} \wedge q_{t} \cong q_{t} \).

An example of a term in \( L(A) \) with an associated successful run are depicted together in Figure 1.

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

Proposition 2.5 Membership is NP-complete for TAGC.

Proof. Given a TAGC \( A = (Q, \Sigma, F, C, \Delta) \) and a term \( t \in T(\Sigma) \), a non-deterministic algorithm consist in guessing a function \( r \) from \( Pos(t) \) into \( Q \), and checking that \( r \) is a successful run of \( A \) 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 \( q \cong q' \).

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

The universality is the problem to decide, given a TAGC \( A \) over \( \Sigma \) whether \( L(A) = T(\Sigma) \). 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 \( \cong \) atomic 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 \( \cong \) 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 \( s \) wrr 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 \( \Sigma \) such that \( L' \) is not regular (such a language exists) and let \( L_1 := f(L, T(\Sigma)) \cup f(T(\Sigma), L') \) where \( f \) is a binary symbol, possibly not in \( \Sigma \). \( f(L, T(\Sigma)) \) denotes \( \{f(s,t) \mid s \in L, t \in T(\Sigma)\} \). It is obvious that \( L_1 \) is a TAGC language.

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

The emptiness is the problem to decide, given a TAGC \( A \), whether \( L(A) = \emptyset \)? 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 \( A_1 = (Q_1, \Sigma_1, F_1, C_1, \Delta_1) \) and \( A_2 = (Q_2, \Sigma_2, F_2, C_2, \Delta_2) \) be two TAGCs. We can assume wlog that \( Q_1 \) and \( Q_2 \) are disjoint.

The TAGC \( A_{\cup} = (Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, F_1 \cup F_2, C_1 \cup C_2, \Delta_1 \cup \Delta_2) \) recognizes \( \mathcal{L}(A_1) \cup \mathcal{L}(A_2) \).

We define a TAGC \( A_\cap = (Q_1 \times Q_2, \Sigma_1 \times \Sigma_2, F_1 \times F_2, C_\cap, \Delta_\cap) \) recognizing \( \mathcal{L}(A_1) \cap \mathcal{L}(A_2) \). The constraint \( C_\cap \) is obtained from \( C_1 \land C_2 \) 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 \( \land_{q_1, q_1' \in Q_1} (q_1, q_1') \approx (q_1, q_1') \) (resp. \( \land_{q_2, q_2' \in Q_2} (q_2, q_2') \approx (q_2, q_2') \)), and similarly for the atoms \( q_1 \not\approx q_1', q_2 \not\approx q_2' \).

The constraint \( C_\cap \) is a run is that the resulting states of \( \Delta_\cap = \{ f((q_1,1), q_1,1), \ldots, (q_1,n), q_2,n) \} \rightarrow (q_1, q_2) \mid f(q_1,1,\ldots,q_1,n) \rightarrow q_1 \in \Delta_1 \text{ for } i = 1,2 \} \). The closure under complementation of TAGC would contradict Proposition 2.6 and Theorem 3.2 below. \( \Box \)

### 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 \( A \), some PCTAGC \( A_1, \ldots, A_n \) can be computed satisfying \( \mathcal{L}(A) = \bigcup \mathcal{L}(A_1) \).

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 \( e_0,\ldots,e_0 \) 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.

**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 set of positions involved in such a kind of pumping.

**Definition 3.2** Let \( A \) be a PCTAGC. Let \( r \) be a run of \( A \).

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 \( \bar{H}_i \) as \( \{ p,j \in Pos(r) \mid h(r|_p,j) < i \land h(r|_p,j) > i \} \).

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

The following lemma is rather straightforward from the previous definition.

<table>
<thead>
<tr>
<th>( i )</th>
<th>( H_i )</th>
<th>( \bar{H}_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>{A}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>4</td>
<td>{3}</td>
<td>{1,2}</td>
</tr>
<tr>
<td>3</td>
<td>{3,3}</td>
<td>{1,2,3,1,3,2}</td>
</tr>
<tr>
<td>2</td>
<td>{3,3,3}</td>
<td>{1,2,3,1,3,2,3,3,1}</td>
</tr>
<tr>
<td>1</td>
<td>{2,3,3,3,3,3,3,3}</td>
<td>{1,3,3,3,3,3,3,3}</td>
</tr>
<tr>
<td>0</td>
<td>{1,2,3,3,3,3,3,3}</td>
<td>\emptyset</td>
</tr>
</tbody>
</table>
Lemma 3.4  Let $A$ be a PCTAGC. Let $r$ be a run of $A$. Let $i$ be an integer between 0 and $h(r)$, then any two different positions in $H_t \cup H_i$ are parallel, and for any arbitrary position $p$ in $Pos(r)$ there is a position $\bar{p}$ in $H_t \cup H_i$ 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 $H_t \cup H_i$ satisfies $h(r|_{\bar{p}}) > i$. Thus, such a $p$ is not in $H_t \cup H_i$. For the second fact, consider any $p$ in $Pos(r)$. If $h(r|_{\bar{p}}) \leq i$ holds, then the smallest position $\bar{p}$ satisfying $\bar{p} < p$ and $h(r|_{\bar{p}}) \leq i$ is in $H_t \cup H_i$, and we are done. Otherwise, if $h(r|_{\bar{p}}) > i$ holds, then the smallest position $\bar{p}$ of the form $p.1\ldots1$ and satisfying $h(r|_{\bar{p}}) \leq i$ is in $H_t \cup H_i$, and we are done.

Definition 3.5  Let $A$ be a PCTAGC. Let $r$ be a run of $A$. Let $i, j$ be integers satisfying $0 \leq j < i \leq h(r)$. A pump-injection $I : (H_t \cup H_i) \to (H_j \cup H_i)$ is an injection function such that the following conditions hold:

(C1) $I(H_t) \subseteq H_j$ and $I(H_i) \subseteq H_j$.

(C2) For each $\bar{p}$ in $H_t \cup H_i$, $r(\bar{p}) = r(I(\bar{p}))$.

(C3) For each $\bar{p}_1, \bar{p}_2$ in $H_t \cup H_i$, (term$(r|_{\bar{p}_1}) = term(r|_{\bar{p}_2}) \iff (term(r|I(\bar{p}_1)) = term(r|I(\bar{p}_2))$).

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

By Condition $C_2$, $r[r|I(\bar{p}_1)]_{\bar{p}_1} \ldots r[r|I(\bar{p}_n)]_{\bar{p}_n}$ is clearly a run of $ta(A)$, but it is still necessary to prove that it is a run of $A$. By abuse of notation, when we write $r[r|I(\bar{p}_1)]_{\bar{p}_1} \ldots r[r|I(\bar{p}_n)]_{\bar{p}_n}$, 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 $I : (H_4 \cup H_3) \to (H_4 \cup H_3)$ 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 $I(H_3) \subseteq H_3$ hold, thus $(C_1)$ holds. For $(C_2)$, we have $r(1) = r(I(1)) = q_{1.0}$, $r(2) = r(I(2)) = q_{0}$, and $r(3) = r(I(3)) = q_{1.0}$. Regarding $(C_3)$, for each different $\bar{p}_1, \bar{p}_2$ in $H_4 \cup H_3$, term$(r|_{\bar{p}_1}) \neq term(r|_{\bar{p}_2})$ and term$(r|I(\bar{p}_1)) \neq term(r|I(\bar{p}_2))$ 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 $r[r|I(\bar{p}_1)]_{\bar{p}_1} \ldots r[r|I(\bar{p}_n)]_{\bar{p}_n}$ 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.

Lemma 3.7  Let $A$ be a PCTAGC. Let $r$ be a run of $A$. Let $r'$ be the global pumping $r[r|I(\bar{p}_1)]_{\bar{p}_1} \ldots r[r|I(\bar{p}_n)]_{\bar{p}_n}$ 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 $h(r|_{\bar{p}})$ 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 $H_t \cup H_i$ is a proper prefix of $p$. We prove the second part of the statement by induction on $k$. First, assume $k = 0$. Then, $h(r|_{\bar{p}})$ is $i$. Thus, $p$ is in $H_t$, 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 symbol$(r|_{\bar{p}})$. Thus, $p.1 \ldots p.m$ are all the child positions of $p$ in $r$. Since $h(r|_{\bar{p}})$ 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 $\alpha$ 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 $H_i$, say $\bar{p}_1$. Moreover, $r'|_{\bar{p}_1}$ is $r|I(\bar{p}_1)$, and by Condition $(C_1)$ of the definition of injection $I$, $I(\bar{p}_1)$ belongs to $H_j$. 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 $A$ be a PCTAGC. Let $r$ be a run of $A$. 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 $A$ be a PCTAGC. Let $r$ be a run of $A$. Let $r'$ be the global pumping $r[r^*_I(p_1)][p_1] \ldots [r^*_I(p_n)][p_n]$, 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 $(\text{term}(r[p_1]) = \text{term}(r[p_2])) \iff (\text{term}(r'[p_1]) = \text{term}(r'[p_2]))$ 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, $(\text{term}(r[p_1]) \neq \text{term}(r[p_2]))$ holds and, moreover, $h(r[p_1]) = i + k$ and $h(r[p_2]) = i + k'$. By Lemma 3.7, $h(r'[p_1]) = j + k$ and $h(r'[p_2]) = j + k'$. Thus, $(\text{term}(r'[p_1]) \neq \text{term}(r'[p_2]))$ holds 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, $h(r[p_1]) = i$. Thus, $p_1, p_2$ are in $H_I$, say $p_1$ is $p_1$ and $p_2$ is $p_2$. Therefore, $r'[p_1]$ is $r_I(p_1)$ and $r'[p_2]$ is $r_I(p_2)$. By Condition (C2) of the definition of pump-injection, $(\text{term}(r[p_1]) = \text{term}(r[p_2])) \iff (\text{term}(r'[p_1]) = \text{term}(r'[p_2]))$ holds. Thus, $(\text{term}(r'[p_1]) = \text{term}(r'[p_2])) \iff (\text{term}(r'[p_1]) = \text{term}(r'[p_2]))$ holds and we are done.

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

- **ia.** If $h(r[p_1]) \neq h(r[p_2]) \geq i$ holds, then the result follows by induction hypothesis.

- **ib.** If $h(r[p_1]) \geq i$ and $h(r[p_2]) < i$, then $\text{term}(r[p_1]) \neq \text{term}(r[p_2])$ holds, and moreover, $h(r[p_1]) = i + k_1$ for some $k_1 \geq 0$, and $p_2, \alpha$ belongs to $H$. By Lemma 3.7, $h(r'[p_1]) = j + k_1$. By Condition (C3) of the definition of pump-injection, $h(r'[p_1]) < j$ holds. Thus, $(\text{term}(r'[p_1]) \neq \text{term}(r'[p_2])$ holds, and we are done.

- **ic.** The case where $h(r[p_1]) < i$ and $h(r[p_2]) \geq i$ hold is analogous to the previous one.

- **id.** If $h(r[p_1]), h(r[p_2]) < i$, then $p_1, \alpha, p_2, \alpha$ belong to $H_I$, say $p_1, \alpha, p_2, \alpha$ belong to $\bar{H}_I$. Therefore, $r'[p_1]$ is $r_I(p_1)$ and $r'[p_2]$ is $r_I(p_2)$. By Condition (C3) of the definition of pump-injection, $(\text{term}(r[p_1]) = \text{term}(r[p_2])) \iff (\text{term}(r'[p_1]) = \text{term}(r'[p_2]))$ holds. Thus, $(\text{term}(r[p_1]) = \text{term}(r[p_2])) \iff (\text{term}(r'[p_1]) = \text{term}(r'[p_2]))$ 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 $A$ be a PCTAGC. Let $r$ be a run of $A$. Let $r'$ be the global pumping $r[r^*_I(p_1)][p_1] \ldots [r^*_I(p_n)][p_n]$, with indexes $0 \leq j < i \leq h(r)$ and injection $I$.

Then, $r'$ is a run of $A$.

**Proof.** By Condition (C2) 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 $P$ involved in an atom of the constraint of $A$, i.e., either $r'[p_1] = r'[p_2]$ or $r'[p_1] \neq r'[p_2]$ occurs in the constraint of $A$. According to Lemma 3.4, we can distinguish the following cases:

- **Suppose that a position in $H_I \in H_I$, say $p_1$, is a prefix of both $p_1, p_2$. Then, $r'[p_1] = r_I(p_1), p_1 - p_1$, and $r'[p_2] = r_I(p_2), p_2 - p_1$, hold. Hence, $r'[p_1]$ and $r'[p_2]$ are subsums 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 $H_I \in H_I$, say $p_1$ and $p_2$, are prefixes of $p_1$ and $p_2$, respectively. Then, $r'[p_1] = r_I(p_1), p_1 - p_1$, and $r'[p_2] = r_I(p_2), p_2 - p_2$, hold. By the injectivity of $I$, $I(p_1) \neq I(p_2)$ holds. Moreover, by Lemma 3.4, $I(p_1) \parallel I(p_2)$ holds. Hence, as before, $r'[p_1]$ and $r'[p_2]$ are subsums 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 $H_I \in H_I$, and that $p_2$ satisfies that some position in $H_I \in H_I$ 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, $(\text{term}(r[p_1]) \neq (\text{term}(r[p_2])$ 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 $r'(p_1) \neq r'(p_2)$. Thus, the atom involving $r'(p_1)$ and $r'(p_2)$ is necessarily of the form $r'(p_1) \neq r'(p_2)$. By Lemma 3.7, $h(r'[p_1]) = j + k$. Therefore, $(\text{term}(r'[p_1]) \neq (\text{term}(r'[p_2])$ 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 $H_I \in H_I$. 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 $r(p_2) = r(p_2)$ holds. 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, $(\text{term}(r[p_1]) = (\text{term}(r[p_2])) \iff \text{term}(r'[p_1]) = (\text{term}(r'[p_2])$ holds and we are done.
following conditions:

\( \text{term}(r'_{p1}) = \text{term}(r'_{p2}) \) holds. Thus, the atom involving \( r'(p_1) \) and \( r'(p_2) \) is satisfied in the run \( r' \) for positions \( p_1 \) and \( p_2 \).

\( \square \)

### 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 \( (x_1, \ldots, x_n) \leq (y_1, \ldots, y_n) \) if \( x_i \leq y_i \) for each \( i \) in \( \{1, \ldots, n\} \). We define \( \text{sum}(x_1, \ldots, x_n) := x_1 + \cdots + x_n \).

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

We define the extension of \( \leq \) to \( n \)-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 \( i \leq j \) 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 \( x_i \geq x_j \) 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 \( B : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \) 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 \( T_{ii}, T_{ij} \), for \( i \) in \( \{1, \ldots, L\} \).
- \( \sum(T_{i1}) = 1 \) and \( T_{12} = \emptyset \) hold.
- For each \( i \) in \( \{1, \ldots, L - 1\} \), \( \sum(T_{i(i+1)} + \sum(T_{i(i+1)_2}) \leq a \cdot \sum(T_{i1}) + \sum(T_{i2}) \) 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 \( A \), Lemma 3.13 will be used by making \( a \) to be the maximum arity of the signature of \( A \), and making \( n \) to be the number of states of \( A \).

### 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 \( A \) be a PCTAGC. Let \( q_1, \ldots, q_b \) be the states of \( A \). Let \( r \) be a run of \( A \). Let \( P \) be a set of positions of \( r \). Let \( t \) be a term. We define \( r_P, t_P \) as the following tuple of natural numbers: \( \langle \{ p \in P \mid \text{term}(r_p) = t \wedge t_r(p) = q_1 \} \rangle, \ldots, \{ p \in P \mid \text{term}(r_p) = t \wedge t_r(p) = q_n \} \rangle \).

**Definition 3.15** Let \( A \) be a PCTAGC. Let \( r \) be a run of \( A \). Let \( P \) be a set of positions of \( r \). Let \( \{1, \ldots, t_k\} \) be the set of terms of \( t \). We define \( r_P \) as the multiset \( [r_{t_1}, r_{t_2}, \ldots, r_{t_k}] \).

**Example 3.16** Following our running example, for the representation of the tuples of natural numbers we order the states as \( q_4, q_{2N}, q_{4d}, q_4, q_D, q_M \). The multisets \( r_H \) and \( r_{H_t} \) are presented in Figure 4.

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

**Lemma 3.17** Let \( A \) be a PCTAGC. Let \( r \) be a run of \( A \). Let \( i, j \) be integers satisfying \( 0 \leq j < i \leq h(r) \).

Then, there exists a pump-injection \( I : (H_i \cup H_j) \rightarrow (H_j \cup H_j) \) if and only if \( r_{H_i}, r_{H_j} \leq r_{H_j}, r_{H_j} \).

**Proof.** \( \Rightarrow \). Assume there exists a pump-injection \( I : (H_i \cup H_j) \rightarrow (H_j \cup H_j) \). We just prove \( r_{H_i} \leq r_{H_j} \), since \( r_{H_j} \leq r_{H_j} \) can be proved analogously. By Condition (C1) of the definition of pump-injection, \( I(H_i) \subseteq H_j \). We write \( \{ \text{term}(r_p) \mid p \in H_i \} \) and \( \{ \text{term}(r_p) \mid p \in H_j \} \) more explicitly as \( \{ t_i, 1, \ldots, t_i, a \} \) and \( \{ t_j, 1, \ldots, t_j, b \} \), respectively. Hence, it remains to prove that \( \{ r_{t_i, 1}, H_i, \ldots, r_{t_i, a, H_i} \} \leq \{ r_{t_j, 1}, H_j, \ldots, r_{t_j, b, H_j} \} \). To this end we define the function \( I' : \{1, \ldots, \alpha\} \rightarrow \{1, \ldots, \beta\} \) as follows. For each
Moreover, for positions \( \bar{t}_{\delta} \) for the corresponding positions can be checked analogously. This function \( I' \) is injective due to Condition (C1) of the definition of pump-injection. In order to conclude, it suffices to determine the index \( \delta \) of the term \( t_{\delta} \) satisifying the condition \( \bar{t}_{\delta} = \text{term}(r_{\bar{t}_{\delta}}) \), and define \( I'(\gamma) := \delta \). Thus, \( I'(\gamma) \) is the disjoint union \( \mathcal{D} : \{(p \in H_1 \mid \text{term}(r_p) = t_{\gamma} = \text{state}(r_p) = q)\} \), and allows to connect such definitions and problems.

This case term \( r_{\bar{t}_{\delta}} \neq \text{term}(r_{\bar{t}_{\delta}}) \) and term \( r_{\bar{t}_{\delta}} \neq \text{term}(r_{\bar{t}_{\delta}}) \) hold. Hence, this simple case is enough to prove the whole statement.

We write \( \{ \text{term}(r_p) \mid p \in H_1 \} \) and \( \{ \text{term}(r_p) \mid p \in H_2 \} \) more explicitly as \( \{ t_{i_1}, \ldots, t_{i_\alpha} \} \) and \( \{ t_{i_j}, \ldots, t_{i_\beta} \} \), respectively. Since \( \langle r_{H_2}, r_{H_1} \rangle \leq \langle r_{H_2}, r_{H_1} \rangle \) holds, \( r_{H_2} \leq r_{H_1} \) also holds. Thus, there exists an injective function \( I : \{ 1, \ldots, \alpha \} \rightarrow \{ 1, \ldots, \beta \} \) satisfying the following statement for each \( \delta \) in \( \{ 1, \ldots, \alpha \} \) and each state q of \( A \):

\[
\{ \langle p \in H_1 \mid \text{term}(r_p) = t_{i_\delta} \land r(p) = q \rangle \} \leq \{ \langle p \in H_1 \mid \text{term}(r_p) = t_{i_\delta} \land r(p) = q \rangle \}
\]

In order to define \( I : H_2 \rightarrow H_1 \), we define \( I \) for each of such sets \( \{ p \mid \text{term}(r_p) = t_{i_\delta} \land r(p) = q \} \) as any injective function \( I : \{ p \mid \text{term}(r_p) = t_{i_\delta} \land r(p) = q \} \rightarrow \{ p \mid \text{term}(r_p) = t_{i_\delta} \land r(p) = q \} \), which is possible by the above disquality. The global \( I \) is then injective thanks to the injectivity of \( I' \). Conditions (C2) and (C3) trivially follow from such definition, and we are done.

\[\Box\]

The following lemma follows directly from the definition of the sets \( H_i \) and \( \bar{H}_i \), and allows to connect such definitions with Lemma 3.13.

**Lemma 3.19** Let \( A \) be a PCTAGC. Let \( r \) be the maximum arity of the symbols in the signature of \( A \). Let \( r \) be a run of \( A \). Then, the following conditions hold:

1. \( |H_{h(r)}| = 1 \) and \( |\bar{H}_{h(r)}| = 0.\)
2. For each \( i \in \{ 1, \ldots, h(r) \} \), \( |H_{i-1}| + |\bar{H}_{i-1}| \leq \langle A, H_i \rangle \).
3. For each \( i \in \{ 0, \ldots, h(r) \} \), \( |H_i| = \text{sum}(r_{H_i}) \) and \( |\bar{H}_i| = \text{sum}(r_{\bar{H}_i}).\)

**Proof.** Item (1) is trivial by definition of \( H_i \) and \( \bar{H}_i \) for \( i = h(r) \). For Item (2), it suffices to observe that the positions in \( H_{i-1} \cup \bar{H}_{i-1} \) are all the positions of \( H_{i-1} \) plus all the child positions in \( H_i \), and that each position has at most \( a \) children. For Item (3) we just prove \( |H_i| = \text{sum}(r_{H_i}) \), since \( |H_i| = \text{sum}(r_{\bar{H}_i}) \) can be proved analogously. We write \( \{ \text{term}(r_p) \mid p \in H_1 \} \) more explicitly as \( \{ t_{i_1}, \ldots, t_{i_\alpha} \} \).

Note that \( H_i \) is the disjoint union \( \{ p \in H_i \mid \text{term}(r_p) = t_{i_1} \} \cup \ldots \cup \{ p \in H_i \mid \text{term}(r_p) = t_{i_\alpha} \} \). Thus, \( |H_i| \) equals \( \{ p \in H_i \mid \text{term}(r_p) = t_{i_1} \} + \ldots + \{ p \in H_i \mid \text{term}(r_p) = t_{i_\alpha} \} \).
Lemma 3.20 Let $B : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be the computable function of Lemma 3.13. Let $A$ be a PCTAGC. Let $b$ be the maximum arity of the symbols in the signature of $A$. Let $n$ be the number of states of $A$. Let $r$ be a run of $A$ satisfying $h(r) \geq B(a, n)$.

Then, there is a global pumping on $r$.

Proof. Consider the sequence $\langle r_{h(r)}, r_{\bar{h}(r)}, \ldots, r_{H_0}, r_{\bar{H}_0}, \rangle$. Note that the $n$-tuple $(0, \ldots, 0)$ does not appear in the multisets of the pairs of this sequence. By Lemma 3.19, $|H_{h(r)}| = 1$ and $|H_{\bar{h}(r)}| = 0$, and for each $i$ in $\{1, \ldots, h(r)\}$, $|H_{i-1} + \bar{H}_{i-1}| \leq a \cdot |H_i| + \bar{H}_i$ holds. Moreover, for each $i$ in $\{0, \ldots, h(r)\}$, $|H_i| = \sum(r_{H_i})$ and $|\bar{H}_i| = \sum(r_{\bar{H}_i})$. Thus, $\sum(r_{H_{h(r)}}) = 1$, $\sum(r_{\bar{H}_{h(r)}}) = 0$, and for each $i$ in $\{1, \ldots, h(r)\}$, $\sum(r_{H_{i-1}}) + \sum(r_{\bar{H}_{i-1}}) \leq a \cdot \sum(r_{H_i}) + \sum(r_{\bar{H}_i})$. 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 $r_{H_i} - r_{H_j} \leq r_{H_{i-1}} - r_{H_{j-1}}$. By Lemma 3.17, there exists a pump-injection $I : (H_i \cup \bar{H}_i) \to (H_j \cup \bar{H}_j)$. Therefore, there exists a global pumping on $r$.

\[ H_i \mid \text{term}(r_{H_i}) = t_{\alpha_i} \]. We conclude by observing that $|\{p \in H_i \mid \text{term}(r_p) = t_{\alpha_i}\}| = \sum(r_{t_{\alpha_i} H_i}) = \sum(r_{t_{\alpha_i} H_i})$ holds.

Theorem 3.21 Emptiness is decidable for PCTAGC.

Proof. Let $a$ be the maximum arity of the symbols in the signature of $A$. Let $n$ be the number of states of $A$. Let $r$ be an accepting run of $A$ 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 $A$. Thus, $r'$ contradicts the minimality of $r$. We conclude that $h(r) < B(a, n)$ holds.

The decidability of emptiness of $A$ 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[$\approx$, $\neq$].

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 $\sum_{q \in Q} a_q \cdot |q| \geq a$ or $\sum_{q \in Q} a_q \cdot |q| \geq a$ where every $a_q$ and $a$ belong to $\mathbb{Z}$.

Let $r$ be a run on a term $t$ of a TA or TAGC $A$ over $\Sigma$ 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 $\|q\| = \{s \in T(\Sigma) \mid \exists p \in Pos(t), r(p) = q, s = t|_p\}$.

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 $0|1 \rightarrow q_{cheap}$, $2|3 \rightarrow q_{moderate}$, $4|5|6 \rightarrow q_{expensive}$, $7|8|9 \rightarrow q_{chic}$ and $N(q_{cheap}, q_{cheap}) \rightarrow q_{p}$. The price is a new argument of $L_0$, $L$ and $M$, hence we replace the transitions with these symbols in input by $L_0(q_{id}, q_{t}, q_p) \rightarrow q_L$. Let $(q_{id}, q_{t}, q_p, q_L) \rightarrow q_L$. $M(q_{id}, q_{t}, q_p, q_L) \rightarrow q_M$. We can use a linear inequality $|q_{cheap}| + |q_{moderate}| - |q_{expensive}| - |q_{chic}| \geq 0$ 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 $|.|_Z$ and $\|.|_Z$ the types of the above linear inequalities, seen as atomic constraints of TAGC. The class TAGC[$|.|_Z$] 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 $\mathbb{N}^{\mathbb{N}}$ whose components are the $|q|_Z$ for every run $r$) of such a language is a semilinear set. The idea for deciding emptiness for a TAGC[$|.|_Z$] $A$ is to compute this semi linear set and test the emptiness of its intersection with the set of solutions in $\mathbb{N}^{\mathbb{N}}$ of $C_A$, (a Boolean combinations of linear inequalities of type $|.|_Z$) which is also semilinear. This can be done in NPSPACE, see [5].

Combining constraints of type $\approx$ and counting constraints of type $|.|_Z$ however leads to undecidability.

Theorem 4.2 Emptiness is undecidable for PTAGC[$\approx$, $|.|_Z$].
occurrences of \( |q_{12}| \) of occurrences of \( q_{12} \) in the run \( r \) is equal to \( x_1 \cdot x_2 \).

The general construction of the TAGC\( |\approx|, |\cdot|, |\cdot| \) \( \mathcal{A} = (Q, \Sigma, F, C, \Delta) \) associated to the above Diophantine equation (\( \phi \)) 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 \( |\cdot|, |\cdot| \) 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 \( W \) be the set containing all the variables of (\( \phi \)), \( \{x_1, \ldots, x_m\} \) and all the prefixes of the monomials of (\( \phi \)) \( x_1^{j_1} \cdots x_m^{j_m} \) for all \( j \leq n \) (we consider every monomial as a word over the alphabet \( \{x_1, \ldots, x_m\} \)). Let us denote the prefix ordering over strings by \( \preceq \), and for every words \( v, v', w \) such that \( v = v', v' \) (i.e. \( v \not\preceq w \)) 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 \( W \) is ordered arbitrarily in a sequence without duplicates \( \mathcal{w} = (w_1, \ldots, w_N) \).

The signature \( \Sigma \) contains the symbols \( a, b, f \) with the respective arities 0, 0 and 2 as above, and two symbols \( c \) of arity 0 and \( g \) of arity 2. Let \( Q = \{q_{w}, s_w, p_w^v, p_w | v, w \in W, v \preceq w \} \cup \{s_0\} \) and \( F = \{s_w \} \). The transitions of \( \Delta \) are the following: \( a \to q_w, b \to p_w^v, f(q_w, p_w) \to p_w^v, f(q_w, p_w) \to p_w^v, f(q_w, p_w) \to r_w \) for all \( v, w \in W \) with \( v \preceq w \), and \( f(q_w, p_w) \to p_w^v, f(q_w, p_w) \to r_w \) for all \( v, w \in W \) and \( x_i \) variable of \( \phi \) \((i \leq m)\), with \( v, w \preceq \).

Moreover, we also have: \( c \to s_0, g(r_w^v, s_0) \to s_w \) and \( g(r_w^v, s_0) \to s_w \) for all \( w, w' \in W \) such that \( w = w_1 \) and \( w' = w_{i+1} \) in the sequence \( \mathcal{w} \), for some \( i < N \).

We can observe that with these transitions, if \( r \) is an accepting run of \( \text{ta}(\mathcal{A}) \) on some term \( t \), then for all \( w \in \mathcal{W} \), the state \( s_w \) occurs exactly once in \( r \) at some position \( p_w \) of \( 2^* \), and the leaves of \( r' p_w \cdot 1 \) 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:

\[
\begin{align*}
  i. \quad |r_w^v| &= |q_w|\div v \quad \text{for all} \ v, w \in \mathcal{W} \ \text{with} \ v \preceq w, \\
  ii. \quad r_w^v \approx r_w^{w'} \quad \text{for all} \ v, w, w' \in \mathcal{W} \ \text{with} \ v \preceq w, w', \\
  iii. \quad \sum_{j=1}^{n} a_j \cdot |q_{w_j}| &= 0.
\end{align*}
\]

In iii., the \( a_j \)'s are the coefficients of (\( \phi \)) and \( w_j = x_1^{j_1} \cdots x_m^{j_m} \) (we assume that the \( n \) first elements of the sequence \( \mathcal{w} \) 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 $A$ on a term $t$, all variables $x_i$ of ($\phi$) and all $w,x_i \in \mathcal{W}$, $[|qw,w,x_i|_r] = [|qw|_r] - [|x_i|_r]$. It follows, using the constraint iii, that $\mathcal{L}(\Delta) \neq \emptyset$ iff ($\phi$) 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 $\neq$ 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 $\sum a_q \cdot |q| \geq a$, with $a_q, a \in \mathbb{N}$, to refer to $\sum -a_q \cdot |q| \leq -a$, and also linear inequalities $\sum a_q \cdot |q| = a$, 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 $\parallel \parallel$ and $\parallel \parallel$.

Below, we shall abbreviate these two types by $\parallel$.

The main difference between the linear inequalities of type $\parallel \parallel$ and $\parallel \parallel$ (and respectively $\parallel \parallel$ and $\parallel \parallel$) 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 with arithmetic constraints.

Proposition 4.3 For all PTAGC $\parallel \parallel$ $\Delta$, one can effectively construct a PTAGC $\Delta'$ such that $\mathcal{L}(\Delta') = \mathcal{L}(\Delta)$.

The first step of this construction consists in rewriting conjunctions of constraints with $\parallel \parallel$ and $\parallel \parallel$ 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 $\parallel \parallel$, $\parallel \parallel$.

Lemma 4.4 Every conjunction $\varphi$ of natural linear inequalities over $Q$ can be effectively rewritten into an equivalent disjunction $\bigvee_{1 \leq i \leq n} p_i \wedge p_{i,\parallel}(q)$ for some $n \in \mathbb{N}$, where $p_i \parallel q$ (resp. $p_{i,\parallel}(q)$) 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 $\varphi_\parallel$ the conjunction of inequalities on occurrences, and $\varphi_{\parallel,\parallel}$ the conjunction of the ones on cardinality. So $\varphi = \varphi_\parallel \wedge \varphi_{\parallel,\parallel}$. Then, we further separate the two parts into $\varphi_\parallel = \chi_\parallel \wedge \psi_\parallel$ and $\varphi_{\parallel,\parallel} = \chi_{\parallel,\parallel} \wedge \psi_{\parallel,\parallel}$ where $\chi_\parallel$ contains all the inequalities of the form $|q| = a$ or $|q| \geq a$ where $q$ does not occur in another inequality of $\varphi_\parallel$, and $\psi_\parallel$ contains all the other inequalities. Note that each state occurring in $\chi_\parallel$ occurs only in this subformula and does not occur in $\psi_\parallel$. Respectively, $\chi_{\parallel,\parallel}$ is the conjunction of all the inequalities of $\varphi_{\parallel,\parallel}$ of the form $|q| = a$ or $|q| \geq a$ where $q$ does not occur in another inequality, and $\psi_{\parallel,\parallel}$ contains all the other inequalities.

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

1. If there exists some $i$ such that $a_{i,k} \neq 0$, then, we define $s_q = \inf_{1 \leq i \leq k, a_{i,k} \neq 0} (a_{i,k} \over a_{i,k})$. If $|q| \geq s_q$, then there exists some inequality which cannot 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 $\varphi_\parallel$ by a disjunction $\bigvee_{0 \leq s \leq s_q} \varphi_{s,\parallel}$, where $\varphi_{s,\parallel} = \chi_\parallel \wedge \psi_{s,\parallel}$.

We define $\chi_{s,\parallel} = \chi_\parallel \wedge |q| = s$ and $\psi_{s,\parallel}$ the following way. First, we replace every inequality $\sum_{q \in Q \setminus \{q\}} a_{i,q} \cdot |q'| \leq a^i$ by $\sum_{q \in Q \setminus \{q\}} a_{i,q} \cdot |q'| \leq a^i - s a_{i,k}$ and every inequality $\sum_{q \in Q \setminus \{q\}} b_{i,q} \cdot |q'| \geq b^i$ by $\sum_{q \in Q \setminus \{q\}} b_{i,q} \cdot |q'| \geq \sup(0, b^i - s b_{i,l})$. 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 $b_{i,l} \neq 0$ and we define $s_q = \sup_{1 \leq j \leq l, b_{i,l} \neq 0} (b_{i,l} / b_{i,l})$. 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 other inequalities with an occurrence of $|q|$ if $|q| \geq s_q$. We replace $\varphi_\parallel$ by a disjunction $\bigvee_{0 \leq s \leq s_q} \varphi_{s,\parallel}$, where $\varphi_{s,\parallel} = \chi_\parallel \wedge \psi_{s,\parallel}$. We define $\chi_{s,\parallel} = \chi_\parallel \wedge |q| = s$ if $s < s_q$, $\chi_{s,\parallel} = \chi_{s,\parallel} \wedge |q| \geq s$ is $s = s_q$ and we define $\psi_{s,\parallel}$ as above (except that there is no $a_{i,k} \neq 0$) if $s < s_q$ and $\psi_{s,\parallel}$ as $\psi_{s,\parallel}$ where every inequality $\sum_{q \in Q} b_{i,q} \geq b^i$ where $b_{i,l} \neq 0$ is deleted.

Note that, at this point of the procedure, $q$ does not occur anymore in any $\psi_{s,\parallel}$ and occurs only once in each $\chi_{s,\parallel}$. While there is still a non-empty conjunction $\psi_{s,\parallel}$, we apply the procedure to $\varphi_{s,\parallel}$. At the end, we have an expression
equivalent to $\varphi_{\mid i}$ that can be written $V_{1 \leq i \leq n} \chi_{i \mid i}$ for some $n$ where each $\chi_{i \mid i}$ is a conjunction of (in)equations of the form $|q| = a$ or $|q| \geq a$.

We apply the same procedure to $\varphi_{\mid i}$, which becomes a disjunction $V_{1 \leq j \leq m} \chi_{i \mid j}$ for some $n$ where each $\chi_{i \mid j}$ is a conjunction of (in)equations of the form $\|q\| = a$ or $\|q\| \leq a$.

Hence, we have that $\phi$ is equivalent to $V_{1 \leq j \leq m} \chi_{i \mid j}$ and $\|q\| = a$ or $\|q\| \leq a$.

Then, we only need to show that a TAGC with a conjunction of simple natural in(equalities) can be effectively transformed into a TAGC[$\approx$, $\neq$].

Lemma 4.5  Given a PTAGC[$\approx$, $\neq$, $\mathbb{N}$] $\mathcal{A}$, whose arithmetic constraints are like in Lemma 4.4, one can effectively compute a TAGC[$\approx$, $\neq$] $\mathcal{A}'$ 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 $\neq$ 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[$\approx$, $\neq$] $\mathcal{A}' = \langle Q', \Sigma, F', C', \Delta' \rangle$ recognizing $L(\mathcal{A})$. Let us define $M = \max(\sup_{q, p_{\|q\|} \mid (\|q\| = b_q)} (b_q), \sup_{q, p_{\|q\|} \mid (\|q\| \geq b_q)} (b_q))$. Then we define a new set of copies of states of $Q'$,

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

and we define $Q' = Q' \times (Q \rightarrow \{0, \ldots, M + 1\})$. So a state of $Q'$ is a pair $\langle q^i, \delta \rangle$ where $q \in Q$, $1 \leq i \leq c(q)$, $q^i \in Q'$ and $\delta$ is a mapping $\delta : Q' \rightarrow \{0, \ldots, M + 1\}$. 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 $m_1 + m_2 = M + 1$.

Then, we define the constraints, rules, and final states of $\mathcal{A}'$. First, we have in $C'$ all the constraints $\langle q^i_1, \delta_1 \rangle \approx (q^i_2, \delta_2)$ such that $(q_1 \approx q_2) \in C$ and all the constraints $\langle q^i_1, \delta_1 \rangle \neq (q^i_2, \delta_2)$ such that $(q_1 \neq q_2) \in C$. 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 $q^i \approx q^j$ for all $1 \leq i \leq b_q$ and $q^i \neq q^j$ for all $1 \leq i < j \leq b_q$.

$\Delta'$ is the set containing all the transition rules $f((q^i_1, \delta_1), \ldots, (q^i_n, \delta_n)) \rightarrow (q, \delta)$ such that $q_1, \ldots, q_n, q \in Q$, $f(q_1, \ldots, q_n) \rightarrow q \in \Delta$, for all $j \leq n$, $1 \leq i_j \leq c(q_j)$, for all $q^i \in Q' \setminus \{q\}$, $\delta(q^i) = \sum_{1 \leq j \leq n} b_j(q^j)$, and $\delta(q) = 1 + \sum_{1 \leq j \leq n} \delta_j(q^j)$.

$F'$ is the set containing the states $\langle q^i, \delta \rangle$, such that $q^i \in F$ and $\delta$ respect the following 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 $t \in T(\Sigma)$ there exists an accepting run $r$ of $\mathcal{A}$ on $t$ such that $r \models C$ if there exists an accepting run $r'$ of $\mathcal{A}'$ on $t$.

⇒ : let $r$ be an accepting run of $\mathcal{A}$ on $t$. First, we compute a term $\tau$ on $\mathcal{Q}$ such that $Pos(\tau) = Pos(r) = Pos(t)$ and that for all $p \in Pos(t)$, $\tau(p)$ is a copy $q^i \in Q'$ of the state $r(p) = q$. For every $q \in Q$, if $p_{\|q\|} = \top$, then there is only one copy $q^i$ of $q$ in $Q'$. So, for all $p$ with $r(p) = q$ we define $\tau(p) = q^1$. If $p_{\|q\|} = (\|q\| = b_q)$ (resp. $p_{\|q\|} = (\|q\| \geq b_q)$) 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 $t_p = t$, we define $\tau(p) = q^t$. And, if $p_{\|q\|} = (\|q\| \geq b_q)$, 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 $\tau(p) = q^{b_q + 1}$.

Then, we define $r'$ in the following way: for all position $p \in Pos(t)$, $r'(p) = \langle \tau(p), \delta_p \rangle$ where for all $\overline{q} \in Q'$, $\delta_p(\overline{q}) \approx \overline{q}$ wrt to $\tau_p$ 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 $\Delta'$ allow us to define such a run. We know have to check that the constraint $C'$ is satisfied by $r'$ and $t$. 13
This construction satisfies the atomic constraints \( \langle q_1', \delta_1 \rangle \approx \langle q_2', \delta_2 \rangle \) (resp. \( \langle q_1, \delta_1 \rangle \not\approx \langle q_2, \delta_2 \rangle \)), 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 \( q_1' \) and \( q_1 \) we have chosen in \( \tau \) 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 \( p_{\parallel\parallel}(q) = (||q|| = b_q) \), or \( p_{\parallel\parallel\parallel}(q) = (||q|| \geq b_q) \) there are constraints of two types: \( \langle q', \delta_1 \rangle \not\approx \langle q', \delta_2 \rangle \) for \( 1 \leq i < j \leq b_q \), for all \( \delta_1, \delta_2 \) and \( \langle q', \delta_1 \rangle \approx \langle q', \delta_2 \rangle \) for \( 1 \leq i \leq b_q \) and for all \( \delta_1, \delta_2 \). By construction of \( r_B \), \( 1 \leq i \leq b_q \) all the positions \( p \) such that \( r_B(p) = (q', \delta) \), for all \( \delta \), 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 \( A \) we know that \( r_B(\varepsilon) = (q', \delta) \) where \( q_B \in F \) and \( \delta(q') = ||q'|| \) wrt \( r' \) if \( ||q'|| \leq M \) and \( ||q'|| = 1 \) otherwise.

We now have to ensure that \( \sum_{1 \leq i \leq ||q'||} \delta(q_i') \) is either \( ||q|| \) wrt \( r \) if \( ||q|| \leq M \) or \( ||q|| = 1 \) otherwise. Hence, all expressions \( p_{\parallel\parallel}(q) \) of \( C \) are satisfied. And for every expression \( p_{\parallel\parallel\parallel}(q) = (||q|| = b_q) \), we have exactly \( b_q \) copies of \( q \) in \( Q \).

The constraints \( \langle q', \delta_1 \rangle \approx \langle q', \delta_2 \rangle \) and \( \langle q', \delta_1 \rangle \not\approx \langle q', \delta_2 \rangle \) for all \( \delta_1, \delta_2 \), \( i \neq j \) 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 \( p_{\parallel\parallel\parallel}(q) = (||q|| \geq b_q) \), the extra copy \( q^i+1 \) 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 \( A \) on \( t \). \( \square \)

**Proposition 4.6** For all \( \Delta T G C \approx, \not\approx, N \) \( A' \) one can effectively construct a PTAGC \approx, \not\approx, N \( A' \) s.t. \( L(A') = L(A) \).

**Proof.** Let \( A = \langle Q, \Sigma, F, C, \Delta \rangle \). We can assume wlog that \( C = C_+ \land C_- \land C_\not\approx_\parallel \) 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 \( \lnot(q \approx q') \) or \( \lnot(q \not\approx q') \), and \( C_\not\approx_\parallel \) 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 \( A_i = (Q, F, C_i, \Delta) \), obtaining \( A'_i \) and let \( A' \) be the disjoint union of the \( A'_i \).s.

We construct below a positive \( \Delta T G C \approx, \not\approx, N \) \( A' = \langle Q', \Sigma, F', C', \Delta' \rangle \) recognizing \( L(A) \). The idea for the construction of \( A' \) 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 \( \lnot(q_1 \approx q_2) \). By definition, for every successful run \( r \) of \( A \) on a term \( t \in T(\Sigma) \), 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 \( \Delta' \) by adding to \( \Delta \) all the transitions obtained from \( \Delta \) 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 \( C_\not\approx_\parallel \) from \( C_+ \) and \( C_\not\approx_\parallel \). Also let \( F' = F \cup \{q_1', q_2' \} \). Finally, we let \( C' = C_+ \land C_\not\approx_\parallel \not\approx_\parallel \land C_\not\approx_\parallel \land \lnot(q_1' \approx q_2') = 1 \land ||q_1'|| = 1 \).

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 \( C_- = \lnot(q_1 \approx q_2) \land \lnot(q_1 \approx q_3) \). For a successful run \( r \) of \( A \) on \( t \), there exist positions \( p_1, p_2, p_3 \in Pos(t) \) such that \( r(p_1) = r(p_2) = q_1, r(p_3) = q_2 \) and \( t_{p_1} \neq t_{p_2} \) and \( t_{p_2} \neq t_{p_3} \). 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 \( p_1 \neq p_2 \) then we can choose two copies \( q_1^1 \) and \( q_2^1 \) of \( q_1 \), that will reach respectively \( p_1 \) and \( p_2 \) in a computation of \( A' \) on \( t \). With constraints \( q_1^1 \not\approx q_2^1, ||q_1^1|| = 1 \) and \( ||q_2^1|| = 1 \) as above, we are done. However, if \( p_1 = p_2 \), then we must have only one copy of \( q_1 \), because only one state can reach this position in a computation of \( A' \) 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 $C = d_1 \land \ldots \land d_m \land e_1 \land \ldots \land e_n$ where for all $k \leq m, d_k = \neg(q_{u_k} \approx q_{v_k})$ and for all $\ell \leq n, e_\ell = \neg(q_{v_\ell} \neq q_{v_\ell})$. Let us associate the following closed first order formula to $A$:

$$
\phi_A = \exists x_{u_1}^d, y_{u_1}^d, \ldots, x_{u_m}^d, y_{u_m}^d, x_{v_1}^e, y_{v_1}^e, \ldots, x_{v_n}^e, y_{v_n}^e. \bigwedge_{k=1}^m x_{u_k}^d \neq y_{u_k}^d \land \bigwedge_{\ell=1}^n x_{v_\ell}^e \neq y_{v_\ell}^e.
$$

Let $N$ be the number of variables in $\phi_A$. Every variable of $\phi_A$ is uniquely associated to an occurrence of a state of $Q$ in $C$. For instance, $x_{u_k}^d$ (resp. $y_{u_k}^d$) is associated to the occurrence of $q_{u_k}$ is the left (resp. right) hand side of $d_k$. For $A$ with the above example of $C$, we have $\phi_A = \exists x_{u_1}^d, y_{u_1}^d, x_{v_1}^e, y_{v_1}^e, x_{u_2}^d, y_{u_2}^d, x_{v_2}^e, y_{v_2}^e$. For each $1 \leq i \leq p$, we define $X_i$ as the set of variables associated to occurrences of $q_i$: $X_i = \{x_{u_k}^d | 1 \leq k \leq m, u_k = i\} \cup \{y_{u_k}^d | 1 \leq k \leq m, u_k = i\} \cup \{x_{v_\ell}^e | 1 \leq \ell \leq n, v_\ell = i\} \cup \{y_{v_\ell}^e | 1 \leq \ell \leq n, v_\ell = i\}$. It is clear that $\phi_A$ is satisfiable if and only if $A$ is a finite model with at most $N$ elements. A solution of $\phi_A$ is a mapping $\sigma$ from $\{x_{u_1}^d, y_{u_1}^d, \ldots\}$ to $D$ making true the conjunction of $\phi_A$. Let $sol(\phi_A)$ be the set of solutions of $\phi_A$. To each $\sigma \in sol(\phi_A)$, we associate a TAGC $[\approx, \neq, N]$ $A_{\sigma} = \langle Q_{\sigma}, F_{\sigma}, C_{\sigma}, A_{\sigma} \rangle$, where $Q_{\sigma}$ 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 $A_{\sigma}$ contains all the transitions of $A$ plus additional transitions obtained from $A$ 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 $F_{\sigma} = \{q_i^j | q_i \in Q, q_i \in F\}$. Finally, we let

$$
C_{\sigma} = C_+^Q \land C_+^Q \land \bigwedge_{i \leq p} \bigwedge_{j \leq n_i, j \neq i} (|q_i^j| = 1 \land |q_i^j| \neq q_i^{j'})
$$

where $C_+^Q$ contains all the positive constraints of $C$, where some (possibly zero or more) instances of states of $Q$ are replaced by copies, and $C_+^Q$ is obtained from $C_+$ 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 $A_{\sigma}$ such that there states sets are pairwise disjoint, and let $A'$ be the disjoint union of all the $A_{\sigma}$. It is clear that $A'$ is positive. Let us show that $L(A') = L(A)$.

Let $t \in L(A')$. By construction, there exists a solution $\sigma \in sol(\phi_A)$ such that $t \in L(A_\sigma)$. Let $r'$ be a successful run of $A_\sigma$ on $t$. By definition of $C_\sigma$, 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. $r^{-1}(p_i^j) = \{p_i^j\}$). Moreover, still by definition of $C_\sigma$, the subterms $t_{p_i^j}$ are pairwise disjoint. The mapping $\theta$ which associate to each variable in $X_i$ the corresponding position $p_i^j$ is isomorphic to $\sigma$. Let $r$ be obtained from $r'$ by replacement of every subrun $r_i^j$ by a run of $ta(A)$ headed by $q_i$. It is clear by construction of $\Delta_\sigma$ that $r$ is a run of $ta(A)$. Moreover, following the property of $\theta$ above, $t_r = C_\sigma$ and $t_r = C_+ \land C_\sigma$. It follows that $t_r$ is a successful run of $A$ on $t$.

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

Lemma 3.1 in Section 3 is a consequence of Propositions 4.3 and 4.6. Indeed, given a TAGC $[\approx, \neq, N]$ $A'$ with $L(A') = L(A)$ and then Proposition 4.3 permits to construct a PTAGC $[\approx, \neq, N]$ $A''$ with $L(A'') = L(A)$ and then Proposition 4.3 permits to construct a PTAGC $A'''$ with $L(A''') = L(A)$. Let $A''' = \langle Q'', \Sigma, F'', C'', \Delta'' \rangle$, and let us put the constraint $C''$ in disjointive normal form $C_1 \lor \ldots \lor C_n$ where every $C_i, 1 \leq i \leq n$ is a conjunction of atomic constraints $q \approx q'$, $q \neq q'$. The PCTAGC $A_1 = \langle Q''', \Sigma, F'', C', \Delta'' \rangle, for 1 \leq i \leq n$, is such that $L(A) = L(A_1) \lor \ldots \lor L(A_n)$.

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

**Theorem 4.7** Emptiness is decidable for TAGC $[\approx, \neq, N]$. 

**Proof.** Given a TAGC $[\approx, \neq, N]$ $A$, we can construct as above, using Proposition 4.6 and Proposition 4.3, n PC-TAGC $A_1, \ldots, A_n$ such that $L(A) = L(A_1) \lor \ldots \lor L(A_n)$. Then we can apply Theorem 3.21 to each $A_i, 1 \leq i \leq n$, in order to decide the emptiness of $L(A)$. □

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 $A = \langle Q, \Sigma, F, \Delta \rangle$ where $Q, F, \Sigma$ are defined like for TA and the transitions rules of $\Delta$ have the form: $f(q_1, \ldots, q_n) \rightarrow q$, 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 \( A \) on a term \( t \in \mathcal{T}(\Sigma) \) is a function \( r \) from \( \text{Pos}(t) \) into \( Q \) such that, for all \( p \in \text{Pos}(t) \), there exists a rule \( t(p)(r(p.1), \ldots, r(p.m)) \xrightarrow{\Delta} r(p) \in \Delta \) 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\([\approx, \neq, \ldots]\) 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 \( q_i \) to \( L_0, L \) and \( M \), so that the first argument \( q_i \) will characterize the theoretical time that was needed to cook the dish. Let us consider the transitions with \( L_0, L \) and \( M \) in input by \( L_0(q_{id}, q_i, q_t) \xrightarrow{2 \to 3} q_L \), \( L(q_{id}, q_i, q_t) \xrightarrow{2 \to 3} q'_L \), \( L_M(q_{id}, q_i, q_t) \xrightarrow{2 \to 3} q'_M \), where \( q'_L \) is a new state meaning that there was an anomaly. We also also add a transition \( L(q_{id}, q_i, q_t, q'_L) \rightarrow q'_L \) to propagate \( q'_L \) and \( M(q_{id}, q_i, q_t, q'_M) \rightarrow q'_M \).

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.

The emptiness decision algorithm of Section 3 still works for this extension of TAGC with local brother constraints. This is because a global pumping \( r[r_1(t(p_1)) \ldots r_{m}(t(p_m))] \) 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\([\approx, \neq, \mathbb{N}]\).

### 4.3 Unranked Ordered Trees

Our tree automata models and results can be generalized from ranked to unranked ordered terms. In this setting, \( \Sigma \) 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 \( \Sigma \).

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 \( A = (Q, \Sigma, F, C, \Delta) \) where \( Q, F \) and \( C \) are as in the definition of TAGC in Section 2.2 and the transitions of \( \Delta \) have the form \( a(L) \rightarrow q \) where \( a \in \Sigma, q \in Q \) and \( L \) is a regular (word) language over \( Q^* \), assumed given by a NFA with input alphabet \( Q \).

A run of \( A \) on an unranked ordered \( t \in \mathcal{U}(\Sigma) \) is a function \( r \) from \( \text{Pos}(t) \) into \( Q \) such that for all position \( p \in \text{Pos}(t) \) with \( n \) child, there exists a transition \( t(p)(L) \rightarrow r(p) \in \Delta \) 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\([\approx, \neq, \ldots]\) where the \( \tau_i \)'s can be \( \approx, \neq, \| \neq \|, \|\|n \).

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 \( \Sigma \) the (ranked) signature \( \Sigma_0 := \{ a : 0 \mid a \in \Sigma \} \cup \{ \tau : 2 \} \) where \( \tau \) is a new symbol. The following operator \( \text{curry} \) defines a bijection from \( \mathcal{U}(\Sigma_0) \) into \( \mathcal{T}(\Sigma_0) \): \( \text{curry}(a) := a \) and \( \text{curry}(a(t_1, \ldots, t_n)) := \tau(\text{curry}(a(t_1, \ldots, t_{n-1})), \text{curry}(t_n)) \). An example of application of this operator is presented in Figure 6.

**Proposition 4.10** For all HAGC\([\approx, \neq, \mathbb{N}]\) \( A \) over \( \Sigma \), one can construct effectively in \( \text{PTIME} \) a TAGC\([\approx, \neq, \mathbb{N}]\) \( A' \) over \( \Sigma_0 \) such that \( \mathcal{L}(A') = \text{curry}(\mathcal{L}(A)) \).

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

We can assume \( wlog \) that \( \Delta \) contains at most one transition \( a(L) \rightarrow q \) for each pair \( \langle a, q \rangle \in \Sigma \times Q \). Otherwise, we can replace two transition \( a(L) \rightarrow q \) and \( a(L') \rightarrow q \) by the unique \( a(L \cup L') \rightarrow q \), preserving the language recognized. Let \( B_{a,q} \) be the NFA recognizing \( L \) in the (unique) transition \( a(L) \rightarrow q \) of \( \Delta \) 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 \( \Delta \) of the state sets of the automata \( B_{a,q} \).

![Figure 6. Currying of an unranked term](image-url)
The transitions of the automaton $A'$, when computing on 
$\text{curry}(t)$ for some $t \in U(\Sigma)$, will simulate both the 
transitions of $A$ (vertical transitions) and the transitions of 
the NFAs $B_{a,q}$ (horizontal transitions).

Let $A' = \langle Q \cup P, \Sigma, F, C, \Delta' \rangle$ where $\Delta'$ contains the 
transitions: $a \to q$ if $B_{a,q}$, $g$ recognizes the empty word, $a \to g$ 
for all $q \in Q$ such that $\text{init}_{a,q}$ is the initial state of 
$B_{a,q}$. $0(\Delta) \to p'$ if there is a transition $p \to p'$ in some 
$B_{a',q'}$ and $\text{init}_{a,q}$, $g$, $a \to q'$ if there is a transition $p \to p'$ in 
some $B_{a',q'}$, $g$ is a final state.

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

**Theorem 4.11** Emptiness is decidable for HAGC[$\approx, \neq, \mathbb{N}$].

### 5 Monadic Second Order Logic

A ranked term $t \in T(\Sigma)$ over $\Sigma$ can be seen as a model 
for logical formulae, with an interpretation domain which 
is the set of positions $\text{Pos}(t)$. We consider monadic second 
order formulae interpreted on such models, with quantific-
tions 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 formul-
ae 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 $\Sigma$, 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 $X \neq Y$, 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 $\sum a_i \cdot |X_i| \geq a \sum a_i \cdot |X_i|^\| \geq a$, 
where every $a_i$ and $a \in \mathbb{Z}$ such that $|X_i|$ is interpreted 
as the cardinality of $X_i$ and $|X|$ as the cardinality of 
the set of models of $\phi$. The decidability of the logic follows 
then from Theorem 4.7.

**Theorem 5.2** EMSO[$\approx, \neq, \mathbb{N}, |\cdot|, |\cdot|^+1$] is decidable on 
ranked terms.

**Proof.** We show that for every formula $\phi$ in 
$\text{EMSO}[^{\approx, \neq, \mathbb{N}, |\cdot|, |\cdot|^+1]}$, we can construct a 
$\text{TAGE}$[$\approx, \mathbb{N}$] recognizing exactly 
the set of models of $\phi$. 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 suc-
cessor predicates of category 3 are not sufficient. In order 
describe unranked ordered terms as models, we replace the 
above predicates $S_i$ by:

- $S_i(x, y)$ which holds on every pair of positions of 
the form $(p, p.i)$ (i.e. $y$ is a child of $x$),
- $S_{+i}(x, y)$ which holds on every pair of positions of 
the form $(p, p.i + 1)$ (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 The-
orem 5.2 to EMSO over unranked ordered terms.

**Theorem 5.3** EMSO[$\approx, \neq, \mathbb{N}, |\cdot|, |\cdot|^+1$] is decidable on 
unranked ordered terms.
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 $q \not\equiv q'$ 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[=] 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


