Tree Automata with Global Constraints for Infinite Trees

Patrick Landwehr
RWTH Aachen University, Germany
landwehr@cs.rwth-aachen.de

Christof Löding
RWTH Aachen University, Germany
loeding@cs.rwth-aachen.de

Abstract

We study an extension of tree automata on infinite trees with global equality and disequality constraints. These constraints can enforce that all subtrees for which in the accepting run a state q is reached (at the root of that subtree) are identical, or that these trees differ from the subtrees at which a state q' is reached. We consider the closure properties of this model and its decision problems. While the emptiness problem for the general model remains open, we show the decidability of the emptiness problem for the case that the given automaton only uses equality constraints.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases Tree automata, infinite trees, global constraints

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.47

Acknowledgements We thank Arnaud Carayol for discussions on the subject.

1 Introduction

In this paper, we start the investigation of a model of finite automata on infinite trees with global equality and disequality constraints. These constraints are specified over the state space of the tree automaton: an equality constraint is of the form $q_1 \approx q_2$ for states q_1 and q_2 of the automaton, and a run satisfies the constraint if the subtrees at all nodes at which q_1 and q_2 appear in the run are equal. Similarly, a disequaltiy constraint $q_1 \not\approx q_2$ requires that the subtrees at q_1-positions in the run are all different from the subtrees at q_2-positions.

Originally, such a model with global constraints has been defined in [7] over finite trees for analysing a logic called TQL for querying semi-structured data [6]. The model (referred to as TAGED, tree automata with global equality and disequality constraints) has then been further investigated in [8, 9] because it turned out to be a useful tool for deciding logics whose expressive power goes beyond that of monadic second-order logic (MSO), the latter being equivalent to standard finite tree automata [21] (see also [23]). In [8, 9] closure properties of TAGED are studied, and decidability results for emptiness on restricted classes of TAGED have been obtained. These results have then been further generalised in [2] to the full class of TAGED (even enriched with arithmetic constraints). Besides the use of TAGED for the logic TQL, there are variants of monadic second-order logic (MSO) with equality and disequality tests that characterise the class of TAGED, and the algorithmic results lead to decision procedures for these versions of MSO [9, 2].

As mentioned above, we want to extend this theory to infinite trees. Automata on infinite trees were originally introduced for solving decision problems for MSO over infinite trees [18]. Since then, many algorithms and decision procedures based on these automata have been developed for solving problems in verification, synthesis, language equations, and set constraints (see [3, 14, 11, 1] for some examples).

While there are many extensions of finite automata over finite trees, e.g., automata with global (dis)equality constraints as described above, automata with (dis)equality constraints between siblings [4], automata over infinite alphabets [12, 25], or automata with counting constraints [13, 20], there is not much work on extended models for infinite trees. One reason for this is that extended automaton models over finite trees are motivated by problems coming from term rewriting or from XML document processing, and thus an extension to infinite trees does not seem to be useful. However, we believe that (dis)equality constraints over infinite trees can be used to express some interesting properties. For example, tree automata over infinite trees can represent sets word functions of the form $f : \Sigma_1^* \to \Sigma_2$ (for alphabets Σ_1, Σ_2). Such a function corresponds to a tree of branching degree $|\Sigma_1|$ with nodes labeled by Σ_2 (a node u in the tree can be identified with the directions one has to take from the root to reach this node, and thus corresponds to a word in Σ_1^*; its label is then the function value $f(u)$). This correspondence is used in algorithms for the synthesis of such functions from logical specifications [19, 17, 14]. In this setting, one can use equality constraints to model, for example, resets of synthesized functions (if $w \in \Sigma^*$ corresponds to an input string that models a reset, then the function should behave the same as from the root of the tree, which can be expressed by a global equality constraint in a tree automaton).

A first step for developing a theory of automata on infinite trees with (dis)equality constraints (we only use the term constraints in the following to refer to (dis)equality constraints) was made in [5], where automata on infinite trees with local constraints are analyzed. It is shown that parity tree automata with constraints for direct subtrees of a node have a decidable emptiness problem and the languages recognized by these automata form a Boolean algebra. In [15] it is shown that these automata with a Büchi condition are closed under projection, and that they can be used to decide satisfiability of MSO formulas enriched with a predicate for testing (dis)equality of direct subtrees of a node.

We continue this line of research and analyze automata on infinite trees with global constraints. We show that (similar to the case of finite trees) the class of languages recognizable by this model is closed under union and intersection, but not under complement. We compare different acceptance conditions and prove - among other results - that the equivalence in expressive power between parity- and Muller-acceptance (see [23]) extends to automata with global constraints. We also consider a few decision problems. To be precise, we show the undecidability of the universality- and the regularity-problem for tree automata with global constraints. (Again, analogous results for finite trees have been shown in [9, 2].)

The main result of this paper is the decidability of the emptiness problem if we restrict the automaton to only use equality constraints. It relies on a construction of a parity automaton that only uses reflexive equality constraints of the form $q \approx q$. This construction is the core of the decidability result because the game-based emptiness algorithm for parity tree automata (see [26] or [23]) also solves the emptiness problem in the case of reflexive equality constraints (using memoryless determinacy of parity games).

For the whole class of tree automata with global constraints on infinite trees, the decidability of the emptiness problem is still an open question. The approaches that have been developed for finite trees do not work on infinite trees. For example, in infinite trees it is possible that a tree is equal to one of its subtrees, which is impossible in finite trees.

This paper is structured as follows: In Section 2, we introduce basic notations and formally define the automaton model. Then in Section 3, we discuss closure properties of the classes of languages recognizable by tree automata with global constraints, and we analyze the expressive power of different restricted models (with regard to acceptance condition and the form of the constraints). In Section 4, we first present undecidability results for the universality and regularity problem (which easily generalize from finite trees). We then show that the emptiness problem for parity tree automata with only global equality constraints, is decidable. We give some concluding remarks in Section 5.

2 Preliminaries

2.1 Trees, Languages and Tree Automata

An *alphabet* is a finite set Σ of letters, Σ^* denotes the set of all finite words over Σ, whereas Σ^ω is the set of infinite words over Σ. The *empty word* is notated by ε.

Let Σ be some finite alphabet. An *infinite Σ-labelled (binary) tree* is a mapping $t : \{0,1\}^* \to \Sigma$. The elements of $\{0,1\}^*$ are called *nodes* and the node ε is called the *root*. We sometimes write dom_t instead of $\{0,1\}^*$. For any node u the node $u0$ is called the *left child* of u and $u1$ is the *right child* of u. A *branch* is an infinite word in $\{0,1\}^\omega$. The set of all infinite Σ-labelled trees is denoted by T_Σ^ω.

Let t be a σ-labelled tree. And let $u \in \{0,1\}^*$ be a node then $t|_u$ denotes the *subtree* of t rooted at u. That is $t|_u(v) := t(uv)$ for all $v \in \{0,1\}^*$.

A *(non-deterministic) tree automaton* for infinite trees over the alphabet Σ is a tuple $\mathcal{A} = (Q, \Sigma, \Delta, q_0, \mathrm{Acc})$ where Q is a finite set of states, Σ is a finite alphabet, $\Delta \subseteq Q \times \Sigma \times Q \times Q$ is the transition relation, $q_0 \in Q$ is the initial state and $\mathrm{Acc} \subseteq Q^\omega$ is the acceptance condition.

Let t be an infinite Σ-labelled tree. A *run* of $\mathcal{A}$ on t is a Q-labelled tree ρ such that $\rho(\varepsilon) = q_0$ and for every node $u \in \{0,1\}^*$, $(\rho(u), t(u), \rho(u0), \rho(u1)) \in \Delta$. A run ρ is *accepting* if for every branch $\alpha_1\alpha_2 \ldots \in \{0,1\}^\omega$, the sequence of states $\rho(\varepsilon)\rho(\alpha_1)\rho(\alpha_1\alpha_2)\ldots$ forms an infinite word in Acc. A tree t is accepted by $\mathcal{A}$ if there exists an accepting run ρ of $\mathcal{A}$ on t. The language recognized by $\mathcal{A}$ is defined as $L(\mathcal{A}) := \{t \in T_\Sigma^\omega \mid \mathcal{A} \text{ accepts } t\}$.

Several forms of acceptance conditions appear in the literature. In this paper we mainly focus on so called safety-, Büchi-, parity- and Muller-acceptance:

- If the tree automaton has a *safety* acceptance condition, it holds $\mathrm{Acc} = Q^\omega$. That is every run is accepting.
- For a *Büchi* acceptance condition a subset $F \subseteq Q$ of states is given and an infinite sequence of states $q_1q_2q_3\ldots \in Q^\omega$ is in Acc if and only if there are infinitely many positions $i \in \mathbb{N}$ with $q_i \in F$.
- In the case of a *parity* acceptance condition a mapping $p : Q \to \mathbb{N}$ from states to natural numbers is given and an infinite sequence of states $q_1q_2q_3\ldots \in Q^\omega$ is in Acc if and only if the highest number $p(q_i)$ that appears infinitely often for that sequence is even.
- A *Muller* acceptance condition is given by a set $\mathcal{F} \subseteq 2^Q$ and an infinite sequence of states $q_1q_2q_3\ldots \in Q^\omega$ is in Acc if and only if the set of states that appear infinitely often in that sequence is an element of $\mathcal{F}$.

A language $L \subseteq T_\Sigma^\omega$ is called *regular* if it can be accepted by a parity tree automaton. As a shortened notation, if $\mathcal{A}$ is an automaton, we write $Q_\mathcal{A}$ to denote the set of states of $\mathcal{A}$ and $\Delta_\mathcal{A}$ to denote the transition relation of $\mathcal{A}$.

Even though we present our results only for binary trees, the statements in this paper also hold true for ranked trees with higher branching degree.

2.2 Tree Automata with Global Equality Constraints

A *tree automaton with global constraints (TAGC)* is a tuple $\mathcal{A} = (Q, \Sigma, \Delta, q_0, \mathrm{Acc}, C)$, such that $(Q, \Sigma, \Delta, q_0, \mathrm{Acc})$ is a tree automaton, denoted by $\mathsf{ta}(\mathcal{A})$, and C is a Boolean combination of atomic constraints of the form $q \approx q'$ or $q \not\approx q'$, where $q, q' \in Q$.

A *run* ρ of the TAGC $\mathcal{A}$ on a tree $t \in T_\Sigma^\omega$ is a run of $\mathsf{ta}(\mathcal{A})$ such that ρ satisfies $C_\mathcal{A}$ (denoted by $(t, \rho) \models C_\mathcal{A}$ or $\rho \models C_\mathcal{A}$ if t is clear from the context), where the satisfiability of constraints is defined as follows: For atomic constraints $\rho \models q \approx q'$ holds (resp. $\rho \models q \not\approx q'$

holds), if and only if for all nodes $u, u' \in \mathrm{dom}_t$ with $u \neq u'$, $\rho(u) = q$ and $\rho(u') = q'$, it holds $t|_u = t|_{u'}$ (or $t|_u \neq t|_{u'}$ resp.). This notion of satisfiability is extended to Boolean combinations as usual. In particular, a run satisfies a negated atomic constraint $\neg(q \approx q')$ if there exist two nodes u, u' with $\rho(u) = q$, $\rho(u') = q'$, and $t|_u = t|_{u'}$. So it is important to note that the semantics of $(q \not\approx q')$ is different from $\neg(q \approx q')$ because of the quantifier over the tree nodes.

A run ρ is *accepting* if it is accepting for $\mathsf{ta}(\mathcal{A})$. As usual, the language recognized by $\mathcal{A}$ is the set $L(\mathcal{A})$ of trees $t \in T^\omega_\Sigma$ for which there exists a accepting run of $\mathcal{A}$ on t. Two automata $\mathcal{A}$ and $\mathcal{A}'$ are *equivalent* if they recognize the same language.

Before we give some examples, we define some classes of TAGC with restricted forms of constraints that are used in the later sections. We follow the terminology for TAGC on finite trees as presented in [2] (the model called TAGED in [9] has a slightly different definition, and the notion of 'positive' in [9] differs from the one in [2]). A TAGC is called *positive (PTAGC)* if C is a positive Boolean combination (i.e. it does not use the negation symbol $\neg$ in the Boolean combination of atomic constraints). Note that atomic constraints of the form $(q \not\approx q')$ can be used in PTAGC because they do not use the negation symbol. In a *conjunctive* TAGC, the constraint is a single conjunction of possibly negated atomic constraints. Accordingly, a conjunctive PTAGC has only one conjunction of atomic constraints. A PTAGC is called *rigid* if it is conjunctive and uses only reflexive equality constraints, that is, C is just a single conjunction of atomic constraints, and for each atomic constraint of the form $q \approx q'$, we have $q = q'$ (the term rigid comes from [10]). For some constructions it is helpful if the initial state does not appear in any of the constraints. We call a TAGC that satisfies this restriction *simplified*.

▶ **Example 1.** Let $\Sigma = \{a, b\}$, then the set $\{t \in T^\omega_\Sigma \mid t|_0 = t|_1\}$ is a tree language, which is well known not to be regular. However it is recognized by the safety-PTAGC $\mathcal{A} := (\{q_0, q_1, ☺\}, \Sigma, \Delta, q_0, q_1 \approx q_1)$, where $\Delta := \{(q_0, x, q_1, q_1) \mid x \in \Sigma\} \cup \{(q, x, ☺, ☺) \mid q \in \{q_1, ☺\}, x \in \Sigma\}$.

In example 1 the automaton did not make use of the "globality" of the constraints. Therefore we present an additional example.

▶ **Example 2.** The language of all trees $t \in T^\omega_\Sigma$ that do not contain themselves as a proper subtree is recognized by the Büchi-PTAGC $\mathcal{A} := (\{q_0, q_1\}, \Sigma, \Delta, q_0, \{q_1\}, q_0 \not\approx q_1)$ where $\Delta := \{(q, x, q_1, q_1) \mid q \in \{q_0, q_1\}, x \in \Sigma\}$.

In both examples we presented PTAGC. The following example uses a negated constraint and also illustrates an interesting interplay between constraints and acceptance conditions:

▶ **Example 3.** Consider the constraint formula $\neg(q_0 \approx q_1 \wedge q_0 \not\approx q_1)$ and note that this formula is not a tautology. If for example the state q_1 does not appear in a run, then both $q_0 \approx q_1$ as well as $q_0 \not\approx q_1$ are satisfied. Thus to satisfy the whole formula, both q_0 and q_1 have to appear in the run. We now use this constraint formula in an automaton:

$$\mathcal{A} := (\{q_0, q_1, ☺\}, \{a, b, c\}, \Delta, q_0, \neg(q_0 \approx q_1 \wedge q_0 \not\approx q_1)) \text{ , where}$$

$$\Delta := \{(q_0, x, q_0, ☺), (q_0, x, ☺, q_0), (q_0, x, q_1, ☺), (q_0, x, ☺, q_1), (☺, x, ☺, ☺) \mid x \in \{a, b, c\}\} \cup \{(q_1, a, ☺, ☺)\}.$$

This safety TAGC starts in q_0, guesses a position at which q_1 appears and checks whether the label at that position is an a. Thus $\mathcal{A}$ accepts precisely those trees over the alphabet

$\{a, b, c\}$, that contain a position labelled with a. This regular language cannot be recognized by a standard safety tree automaton, thus showing that negated constraints actually can introduce stronger acceptance conditions even inside the class of regular languages.

The following examples show some possible applications of TAGC:

▶ **Example 4.** As explained in the introduction, infinite trees can be used to model functions of the form $f : \Sigma_1^* \to \Sigma_2$ for alphabets Σ_1, Σ_2, referred to as input and output alphabets, respectively. We can apply f to infinite input words $\alpha = a_1 a_2 \cdots$ obtaining $f(\alpha) = b_0 b_1 \cdots$ with $b_i = f(a_1 \cdots a_i)$ (with the special case $b_0 = f(\varepsilon)$). In the synthesis problem for such functions, we are given a relation $S \subseteq (\Sigma_1 \times \Sigma_2)^\omega$, called the specification. A function f satisfies the specification if $(\alpha, f(\alpha)) \in S$ for all input words α. The task is to automatically synthesize a function satisfying S from a definition of S by a logical formula or an automaton.

Note that for $\Sigma_1 = \{0, 1\}$, the function f has exactly the same shape as an infinite binary tree (for larger alphabets Σ_1, one can use trees of higher branching degree). Thus, we can identify trees with the functions described above.

As shown in [19, 17] (see also [14]), a specification S defined in MSO logic or linear temporal logic (LTL) over infinite words, can be translated into a tree automaton $\mathcal{A}_S$ accepting precisely those trees that satisfy the specification. The emptiness test for tree automata then yields a tree (and thus a function) satisfying the specification.

Using TAGC, we can model additional properties of such functions. Assume, for example, that we are given a regular language $R \subseteq \Sigma_1^*$ of finite words over the input alphabet that model a reset of the system. The requirement for the function to be synthesized could be that it satisfies the specification S, and whenever the input string processed so far corresponds to a reset in R, then the function should behave in the same way (it moves to a specific reset state and forgets about the past). We can express this property by taking the product of the tree automaton $\mathcal{A}_S$ with a DFA $\mathcal{D}_R$ for the reset words, and impose equality constraints for all product states (q, p) such that p is a final state of $\mathcal{D}_R$. A tree accepted by this product automaton corresponds to a function with the desired reset behavior, and can by synthesized by the methods presented in Section 4.

▶ **Example 5.** Let L be a language of infinite words over the alphabet $\Sigma = \{a_1, \ldots, a_n\}$. We say L is an *ω-power* if $L = U^\omega$ for a regular language U of finite words. It is a *deterministic ω-power* if U is prefix-free (i.e. for all $u \in U$ and v strict prefix of u, $v \notin U$). Here the term 'deterministic' is appropriate, because if U is prefix-free then every word in L has a unique factorization into its U-factors. In general, regular ω-languages can be characterized in terms of concatenations and ω-powers of regular languages of finite words (see [22]). Similarly, deterministic ω-powers can be used to characterize the subclass of deterministic Büchi languages, and are used in [16] to characterize certain regular liveness properties.

We can construct for a given ω-regular language L a PTAGC, whose recognized language is non-empty if and only if L is a deterministic ω-power. We consider n-ary infinite $\{0, 1\}$-labelled trees, where the directions correspond to the n letters of Σ. The constructed parity-PTAGC $\mathcal{A}$ then verifies two conditions on such an input-tree:

- the branches on which infinitely many 1 appear are precisely the words in L.
- The whole tree is equal to all subtrees rooted at positions at which a 1 appears.

If $L = U^\omega$, then the tree t_U is accepted by $\mathcal{A}$, where $t_U(u) = 1$ iff $u \in U^*$. If on the other hand $\mathcal{A}$ accepts a tree, then it also accepts a regular tree t (see Corollary 23 in Section 4). Define U to be the set of all minimal (wrt. prefix order) words u with $t(u) = 1$. Then $L = U^\omega$ since t is accepted by $\mathcal{A}$. As a consequence, the problem whether a given regular ω-language is a deterministic ω-power, is decidable (using Theorem 22).

3 Properties

In this Section we first present some general properties of TAGC and then discuss their closure properties.

3.1 Expressive Power of different Acceptance Conditions

We start by comparing the expressive power of TAGC with different acceptance conditions. The following lemma is useful to convert between different acceptance conditions. The construction is a straightforward adaption of the one for tree automata without constraints commonly found in the literature.

▶ **Lemma 6.** *For a TAGC $\mathcal{A}$ and a deterministic ω-word automaton $\mathcal{B}$, one can construct an automaton $\mathcal{A} \ltimes \mathcal{B}$ that accepts all trees $t \in T_\Sigma^\omega$ for which there is a run ρ of $\mathcal{A}$ on t such that for each branch, the sequence of states along that branch in ρ are accepted by $\mathcal{B}$. The type of acceptance condition of $\mathcal{A} \ltimes \mathcal{B}$ (Büchi, parity, Muller) is the one of $\mathcal{B}$.*

Since all regular ω-languages can be accepted by deterministic parity automata (see, e.g., [23, 24]), we obtain the equivalence of TAGC with parity and Muller conditions.

▶ **Corollary 7.** *A language is recognizable by a parity-TAGC if and only if it is recognizable by a Muller-TAGC. And from a given parity/Muller Automaton one can construct an equivalent automaton with the other acceptance condition.*

3.2 Restricted Constraints

We now consider constructions for simplifying the shape of the constraints. Remember that we call a (P)TAGC conjunctive if its constraint formula is a conjunction of (possibly negated) atomic constraints. And we call a (P)TAGC simplified if its initial state does not appear in its constraint formula. Our goal in this section is to show the following lemma:

▶ **Lemma 8.** *For every Büchi- or parity-TAGC one can construct an equivalent conjunctive and simplified Büchi- or parity-PTAGC (respectively).*

Proof. Let $\mathcal{A}$ be a TAGC. We separate $\mathcal{A}$ into a set of conjunctive TAGC. Wlog. the constraint formula $C_\mathcal{A}$ is a disjunction of conjunctions $C_1, \ldots, C_k$ of possibly negated atomic constraints. Then $L(\mathcal{A}) = L(\mathcal{A}_1) \cup \ldots \cup L(\mathcal{A}_k)$ where $\mathcal{A}_j$ is obtained from $\mathcal{A}$ by replacing the constraint formula with C_j. Now all $\mathcal{A}_j$ are conjunctive. In the following we show in Lemma 9 how to obtain from a given conjunctive TAGC a set of conjunctive PTAGC. After that we show in Lemma 10 how to construct from a given conjunctive PTAGC a conjunctive and simplified PTAGC. Finally, according to Lemma 11 in Section 3.3, which states effective closure under union, we obtain the desired automaton. ◀

▶ **Lemma 9.** *For every conjunctive Büchi- or parity-TAGC $\mathcal{A}$ one can construct conjunctive Büchi- or parity-PTAGC $\mathcal{B}_1, \ldots, \mathcal{B}_n$ (respectively), such that $L(\mathcal{A}) = L(\mathcal{B}_1) \cup \ldots \cup L(\mathcal{B}_n)$.*

Proof sketch. The proof follows a similar idea to [2], where an analogous statement was proven for finite trees. The elimination of negated atomic constraints is based on the following idea. If we want the automaton to check a negated constraint $\neg(q \approx q')$, we can use copies $\hat{q}$, $\hat{q}'$ of the states that the automaton can use in exactly one position of the run in place of q and q'. With this modification, the constraint $\neg(q \approx q')$ can be rewritten as $(\hat{q} \not\approx \hat{q}')$. The general construction is a bit more technical because the automaton has to guess which negated constraints are checked, and then apply the above idea to these constraints in parallel. ◀

One can show, that the language of example 3, that is the language of all trees containing a node labelled with a, is not recognizable by safety-PTAGC. Therefore, the statement in Lemma 9 does not hold for safety automata.

▶ **Lemma 10.** *For every conjunctive safety-, Büchi- or parity-PTAGC $\mathcal{A}$ one can construct an equivalent conjunctive and simplified safety-, Büchi- or parity-PTAGC $\mathcal{B}$ (respectively).*

Proof sketch. The basic idea is to delay the constraints that compare a subtree with the full tree by one step. To be more precise, for a constraint $q_0 \approx q'$ involving the initial state of $\mathcal{A}$, the automaton $\mathcal{B}$ verifies that at each position $u \in \mathrm{dom}_t$ at which q' appears, the label $t(u)$ is identical to $t(\varepsilon)$, and it also verifies that the right subtree $t|_{u0}$ is equal to $t|_0$ as well as that the left subtree $t|_{u1}$ is equal to $t|_1$ (similarly for disequality constraints). ◀

3.3 Closure Properties

We now analyze the closure properties of tree automata with global constraints for infinite trees. As it turns out, the closure properties with regard to Boolean operations are identical to the case of finite trees ([2]). But the proofs sometimes require a few extra steps, since we use a model with a single initial state.

▶ **Lemma 11.** *The class of languages recognizable by conjunctive and simplified PTAGC is effectively closed under union.*

Proof. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be conjunctive and simplified parity-PTAGC. Wlog. $Q_{\mathcal{A}_1} \cap Q_{\mathcal{A}_2} = \emptyset$. Define the automaton $\mathcal{A}_\cup := (Q, \Sigma, q_{\text{start}}, \Delta, p, C)$, where $Q := \{q_{\text{start}}\} \cup Q_{\mathcal{A}_1} \cup Q_{\mathcal{A}_2}$, $\Delta := \Delta_{\mathcal{A}_1} \cup \Delta_{\mathcal{A}_2} \cup \{(q_{\text{start}}, a, q', q'') \mid (q_0^{\mathcal{A}_1}, a, q', q'') \in \Delta_{\mathcal{A}_1} \vee (q_0^{\mathcal{A}_2}, a, q', q'') \in \Delta_{\mathcal{A}_2}\}$. For each $q \in Q_{\mathcal{A}_1}$, $p(q) := p_{\mathcal{A}_1}(q)$ and for each $q \in Q_{\mathcal{A}_2}$, $p(q) := p_{\mathcal{A}_2}(q)$. ($p(q_{\text{start}})$ can be defined arbitrarily.), and $C := C_{\mathcal{A}_1} \wedge C_{\mathcal{A}_2}$.

Note that $\mathcal{A}_\cup$ is indeed conjunctive and simplified and it holds that $L(\mathcal{A}_\cup) = L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$ (since if $\mathcal{A}_i$ is positive, $C_{\mathcal{A}_i}$ is satisfied if no state form $Q_{\mathcal{A}_i}$ appears in the run). Also note that if $\mathcal{A}_1$ and $\mathcal{A}_2$ are safety- or Büchi-PTAGC, then so is $\mathcal{A}_\cup$. ◀

▶ **Theorem 12.** *The classes of languages recognizable by Büchi- and parity-TAGC are effectively closed under union and intersection, but they are not closed under complement.*

Proof sketch. The closure under union follows from Lemma 8 and Lemma 11. The closure under intersection can be shown by a standard product construction. For showing the non-closure under complement, we can closely follow the ideas from [9] for the corresponding result on finite trees. The idea is to take a language, for which the automaton would have to verify infinitely many independent equalities. On the other hand, to check membership for the complement language it suffices to guess only a single disequality. Concretely, we can show that for $\Sigma = \{a, b\}$, the language $L := \{t \in T_\Sigma^\omega \mid \forall i \in \mathbb{N} : t|_{0^i 10} = t|_{0^i 11}\}$ is not recognizable by a TAGC but its complement can be accepted by a Büchi-TAGC. ◀

Since Lemma 9 does not hold for safety automata, we have to prove closure properties in a different way in this case.

▶ **Theorem 13.** *The class of languages recognizable by safety-TAGC as well as the class of languages recognizable by safety-PTAGC is closed under union and intersection, but not under complement.*

Proof sketch. The proof in the case of PTAGC follows exactly the proof of Theorem 12. In the case of TAGC the we make use of the trick from example 3 (describing a reachability condition with the constraint formula). ◀

4 Decision Problems

In this section, we first present some undecidability results, and then show the decidability of the emptiness problem for PTAGC, which only use equality constraints.

4.1 Undecidability Results

The *universality problem* is to decide, given a TAGC $\mathcal{A}$ over Σ whether $L(\mathcal{A}) = T^\omega_\Sigma$.

The universality problem for tree automata with constraints on finite trees was shown to be undecidable in [8]. Therefore the following theorem is not surprising. In principle, we could give a reduction from the case of finite trees, using the results from [8] and [10]. However, one can also easily provide a direct proof by a reduction from the halting problem of two-register machines.

▶ **Theorem 14.** *The universality problem for Büchi-TAGC is undecidable, even for Büchi-PTAGC without disequality constraints.*

The *regularity problem* is to decide, given an automaton $\mathcal{A}$, whether $L(\mathcal{A})$ is regular. Barguñó et. al. have shown the undecidability of the regularity problem on finite trees.

▶ **Theorem 15** ([2])**.** *The regularity problem is undecidable for tree automata with global equality constraints for finite trees.*

▶ **Corollary 16.** *The regularity problem for Büchi-TAGC is undecidable.*

Proof Sketch. From a given tree automaton $\mathcal{A}$ with global equality constraints on finite trees (for a formal definition see eg. [8]) over the alphabet Σ construct a Büchi-TAGC $\mathcal{A}'$ over $\Sigma \cup \{\bot\}$ where $\bot$ is a new symbol, such that $\mathcal{A}'$ accepts precisely those trees $t' \in T^\omega_{\Sigma\cup\{\bot\}}$ for which there exists a tree $t \in L(\mathcal{A})$ with $t'(u) = t(u)$ for all $u \in \mathrm{dom}_t$ and $t'(u) = \bot$ for all $u \notin \mathrm{dom}_t$. Then $L(\mathcal{A}')$ is regular if and only if $L(\mathcal{A})$ is regular. ◀

4.2 Emptiness Problem

We now discuss the decidability of the emptiness problem for TAGC. To be precise, we prove that the emptiness problem for parity-PTAGC without $\not\approx$-constraints is decidable. We do this by a reduction to rigid equality constraints (recall that rigid PTAGC have only reflexive $\approx$-constraints).

We first note in Lemma 17 that for rigid parity-TAGC without $\not\approx$-constraints, the emptiness reduces to the case without constraints, which is known to be decidable (see, e.g., [23, 24]). In Lemma 20 we then present a construction to obtain a rigid automaton (this construction also works in the presence of non-reflexive disequality constraints, while Lemma 17 only works without disequality constraints). The construction in Lemma 20 relies on the existence of memoryless runs for conjunctive parity-PTAGC (to be defined below), which we prove in Lemma 18.

▶ **Lemma 17.** *For each rigid parity-TAGC $\mathcal{A}$ without $\not\approx$-constraints, $L(\mathcal{A}) = \emptyset$ if and only if $L(\mathsf{ta}(\mathcal{A})) = \emptyset$.*

Proof. It trivially holds that $L(\mathcal{A}) \subseteq L(\mathsf{ta}(\mathcal{A}))$. Thus we only have to show that $L(\mathsf{ta}(\mathcal{A})) \neq \emptyset$ implies $L(\mathcal{A}) \neq \emptyset$. Now let $L(\mathsf{ta}(\mathcal{A})) \neq \emptyset$. Then the standard emptiness game (see [23, 24]) for parity tree automata gives us a tree $t \in L(\mathsf{ta}(\mathcal{A}))$ with accepting run ρ such that for all $u, u' \in \mathsf{dom}_t$, $\rho(u) = \rho(u')$ implies $t|_u = t|_{u'}$. Therefore ρ is an accepting run of $\mathcal{A}$ as well, which implies $L(\mathcal{A}) \neq \emptyset$. ◀

Let $\mathcal{A}$ be a parity-PTAGC and $t \in T_\Sigma^\omega$ be a tree. Let ρ be a run of $\mathcal{A}$ on t. We say ρ is *memoryless* if for all nodes $u, u' \in \mathrm{dom}_t$ we have that $\rho(u) = \rho(u')$ and $t|_u = t|_{u'}$ implies $\rho|_u = \rho|_{u'}$. That is, if ρ is in the same state at two nodes with identical subtrees, then ρ is the same on these two subtrees.

▶ **Lemma 18.** *Let $\mathcal{A}$ be a parity-PTAGC and let $t \in L(\mathcal{A})$ be a tree accepted by $\mathcal{A}$. Then there exists a memoryless run of $\mathcal{A}$ on t.*

Proof sketch. A corresponding result is known for standard parity tree automata (see [26, end of Section 7]). The idea is to modify the membership game for an automaton $\mathcal{A}$ and a tree t (called coloring game in [26], Automaton-Pathfinder-Game in [24], and referred to as $\Gamma_{\mathcal{A},t}$ in [23]) by quotienting the game graph w.r.t. equal subtrees. A memoryless (also called positional) winning strategy for the player called "Automaton" in this new game then yields a run of the desired form. In the presence of constraints, we additionally need to start from an accepting run of $\mathcal{A}$, and restrict the game to the positions corresponding to the run. ◀

▶ **Example 19.** Lemma 18 requires the automaton $\mathcal{A}$ to be positive. This requirement is indeed neccessary, since there exists a TAGC that recognizes a non-empty language, but does not have a memoryless run: Define the safety-TAGC $\mathcal{A} := (\{q_0, q_1, q_2\}, \{a\}, q_0, \Delta, C)$ by setting $\Delta := \{(q_0, a, q_1, q_0), (q_0, a, q_2, q_0), (q_1, a, q_0, q_0), (q_2, a, q_0, q_0)\}$ and $C := \neg(q_0 \not\approx q_1) \wedge \neg(q_0 \not\approx q_2)$. For C to be satisfied on the only tree t over the alphabet $\{a\}$, both states q_1 and q_2 have to appear. But there is no memoryless run of $\mathcal{A}$ in which both states appear.

▶ **Lemma 20.** *For each conjunctive and simplified parity-PTAGC $\mathcal{A}$ without reflexive $\not\approx$-constraints, one can construct an equivalent rigid and simplified parity-PTAGC $\mathcal{A}'$.*

Proof. Let $\mathcal{A}$ be the given conjunctive and simplified parity-PTAGC. The automaton $\mathcal{A}'$ that we construct basically guesses a memoryless run of $\mathcal{A}$ on the input tree and verifies that it is indeed an accepting run. The difficulty is that $\mathcal{A}'$ can only use reflexive equality constraints in order to verify the equality constraints imposed by the guessed run of $\mathcal{A}$. Assume, for example, that $q \approx q'$ is a constraint of $\mathcal{A}$, and that q and q' appear in the run ρ of $\mathcal{A}$ that is guessed by $\mathcal{A}'$. In order to check equality of the subtrees at the positions in which q and q' occur in ρ, $\mathcal{A}'$ needs to go into the same state at all these positions. Thus, $\mathcal{A}'$ cannot distinguish anymore between positions with q and q'. However, at the same time, $\mathcal{A}'$ has to check that the parity condition is satisfied along all the branches of the guessed run ρ. For this purpose, we use the existence of memoryless runs, as guaranteed by Lemma 18. Since all subtrees at q positions in the run are the same, in a memoryless run, all subruns at q positions are also the same. We can speak of the run from q. Intuitively, the automaton $\mathcal{A}'$ guesses how these subruns from the constrained states are connected. That is, if in the run from q there is path that leads to q', and k is the maximal priority on this path, then $\mathcal{A}'$ stores this information. If, furthermore, in the run from q' there is a path to q with maximal priority k', then the maximum of k, k' has to be even (otherwise, the underlying run of $\mathcal{A}$ would contain a rejecting path, and $\mathcal{A}'$ has to guess an accepting run of $\mathcal{A}$).

Obviously, the problems and ideas described above are not restricted to pairs of states: If there is an additional constraint of the form $q' \approx q''$, and q, q', q'' all occur in the run, then the states q, q', q'' are in the same class. Formally, for a given subset X of the states, the equality constraints of $\mathcal{A}$ naturally define a relation over X. We take the symmetric and transitive closure of this relation. The class of a state q (in the set X) consists of all states that are related with q in this closure. Note that the class of q is empty if and only if q is not related to any other state from X by an equality constraint. We say that q is a constrained state (again w.r.t. X) if its class is nonempty.

The automaton $\mathcal{A}'$ guesses the set X of states occurring in the run of $\mathcal{A}$, for each class a set $S \subseteq X$ of states that can occur at a subtree corresponding to that class, and for all (S, q) and (S', q') with $q \in S$ and $q' \in S'$, how these states are connected in the subruns on the subtrees. The information about the connections is stored in a graph G. This is a fixed information (X, G) that is guessed at the beginning of the run and does not change anymore.

Then $\mathcal{A}'$ starts in the initial state of $\mathcal{A}$ and guesses a run of $\mathcal{A}$. When the guessed run enters a constrained state q, then $\mathcal{A}'$ goes to a state that starts simulating runs from all the states that can occur at the root of the subtree of q. Whenever one of the simulated runs enters a constrained state, then this is only possible if the other simulated runs agree with the information on the connection of subruns stored in G. In this case, the simulation is restarted from the set of the new constrained state.

Formally, let $\mathcal{A} = (Q, \Sigma, \Delta, q_0, p, C)$ be a conjunctive and simplified parity-PTAGC. We start by defining the graphs that where informally explained above. $\mathbb{G}_{\mathcal{A}}$ is defined as the set of vertex- and edge-labelled directed graphs $G = (V_G, E_G, \nu_G, \mu_G)$ with

1A. vertex labels $\nu_G : V_G \to \{(S, q) \in 2^Q \times Q \mid q \in S\}$ (as explained above, a set S corresponds to the set of states that appear at the root of some subtree that is constrained by an equality constraint in the run of $\mathcal{A}$).

1B. Each two vertices have different labels, and for each vertex with label (S, q) and each $q' \in S$ there is a vertex labelled (S, q').

1C. For each $q \approx q' \in C$, whenever labels (S, q) and (S', q') appear, then $S = S'$ (each class is contained in a unique set). And for each vertex labelled (S, q) there are states $q', q'' \in S$ with $q' \approx q'' \in C$ (each set contains at least one class).

1D. For each $q \not\approx q' \in C$ with $q \neq q'$, there is no vertex labelled (S, q) with $q' \in S$ (states with a disequality constraint cannot appear at the root of the same subtree).

2A. edge labels $\mu_G : E_G \to (2^{p(Q)} \setminus \emptyset)$. (Intuitively, an edge label for an edge $(S, q) \to (S', q')$ encodes the allowed maximal priorities on the paths from q to q' in the run of $\mathcal{A}$.)

2B. for every cycle $e_1 e_2 \dots e_k$ with $e_i \in E_G$, and all $x_1, x_2, \dots, x_k \in p(Q)$ with $x_i \in \mu_G(e_i)$, we have $\max\{x_i \mid 1 \leq i \leq k\}$ is even (concatenating the paths encoded in the edges of G leads to an accepting path in the run of $\mathcal{A}$).

Note that properties 1A. and 1B. imply that $\mathbb{G}_{\mathcal{A}}$ is finite. These graphs are used to define "allowed transitions" such that the underlying guessed run of $\mathcal{A}$ satisfies the parity condition.

▶ **Example 21.** We give an example for such a graph G. Let $\mathcal{A} := (Q, \Sigma, \Delta, q_0, p, C)$ be the parity-PTAGC with

- $Q := \{q_0, q_1, q_1', q_2, q_2', q_3, q_4\}$
- $\Delta := \Big\{(q_0, x, q_1, q_1'),\ (q_1, x, q_1, q_4),\ (q_1', x, q_4, q_2),\ (q_2, x, q_3, q_3),$
  $(q_3, x, q_2', q_2'),\ (q_2', x, q_3, q_3)\ (q_4, x, q_4, q_4) \mid\ x \in \Sigma\Big\}$
- $p(q_0) = p(q_1) = p(q_1') = p(q_4) = 0$ and $p(q_2) = p(q_2') = 1$ and $p(q_3) = 2$
- $C := \ (q_1 \approx q_1') \wedge (q_2 \approx q_2')$

One possible graph in $\mathbb{G}_{\mathcal{A}}$ is depicted in Figure 1. The constructed automaton $\mathcal{A}'$ then can guess $X = Q$ as the set of states that are allowed to appear in the run. And it can guess the graph from Figure 1 to remember allowed transitions.

For the formal construction of $\mathcal{A}'$, we use the following notation. For each $S \subseteq Q$ we enumerate the states as $S = \{q_1^S, \dots, q_{|S|}^S\}$, where $q_j^S < q_{j+1}^S$ for all $j \leq |S|$ and $<$ is an arbitrary but fixed total order on Q.

Now define $\mathcal{A}' := (Q', \Sigma, \Delta', q_0', \mathrm{Acc}', C')$ where

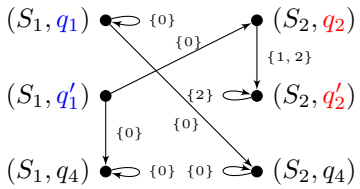

Figure 1 One possible graph from $\mathbb{G}_{\mathcal{A}}$ where $\mathcal{A}$ is the parity-PTAGC from example 21. Here S_1 is the set $\{q_1, q_1', q_4\}$ and S_2 is the set $\{q_2, q_2', q_4\}$. Note that every circle can produce only a maximum priority which is even.

- $Q' := \{q_I\} \cup (Q_{\text{state}} \times Q_{\text{fix}})$, where

$$Q_{\text{state}} \subseteq 2^Q \times \bigcup_{k \leq |Q|} (Q \times p(Q))^k \text{ and } Q_{\text{fix}} \subseteq 2^Q \times \mathbb{G}_{\mathcal{A}}$$

  such that $((S, \overline{w}), (X, G)) \in (Q_{\text{state}} \times Q_{\text{fix}})$ if and only if the following properties are satisfied (the intuition of the components is explained below):
  - Either $S = \{q_0\}$ or $S \subseteq Q$ such that there is a vertex labelled (S, q) in G for some (and therefore all) $q \in S$.
  - $\overline{w} \in (X \times p(X))^{|S|}$.
  - For each vertex labelled (S', q') in G, we have $q' \in X$ (and thus also $S' \subseteq X$).

  Informally, Q_{state} contains two kinds of information: The set for the most recently visited class, and a vector which contains for each element of that set a state and the highest priority visited up to now. This information is updated in every step. As explained earlier, Q_{fix} contains information that is guessed once at the very beginning and then remains unchanged for the remainder of the run.
- $q_0' := q_I$
- $\Delta' := \Delta_{\text{init}} \cup \Delta_{\text{trans}}$, where Δ_{init} contains the transitions that are applicable at the initial state, and Δ_{trans} contains all other transitions. We first define Δ_{trans}: Let $((S, \overline{w}), (X, G)) \in Q'$ be a state and let $a \in \Sigma$ be a letter. The automaton chooses for each entry $w_j = (q_j, p_j)$ in $\overline{w} = (w_1, \ldots, w_{|S|})$ a transition $(q_j, a, q_j^0, q_j^1) \in \Delta$, such that $q_j^i \in X$ for both $i \in \{0, 1\}$, and one of the following holds:
  - Either $\{q_j^i \mid j \leq |S|\} \cap \{q \in X \mid \exists q' \in X : q \approx q' \in C\} = \emptyset$. That is, none of the guessed successor states in direction i is a constrained state in X. In this case, the i-successor is $((S, \overline{w}'), (X, G))$, where $\overline{w}' := \big((q_1^i, \max(p_1, p(q_1^i))), \ldots, (q_{|S|}^i, \max(p_{|S|}, p(q_{|S|}^i)))\big)$.
  - Or there is a constrained state q_j^i and a vertex label (S', q_j^i) of G, such that
    - $S = \{q_0\}$ (the run enters a class for the first time), or
    - $S \neq \{q_0\}$, and for each $j \leq |S|$, the edge label $\mu(e_j)$ of the edge from (S, q_j^S) to (S', q_j^i) contains the priority p_j (the path segments of the simulated runs are encoded in G).

    In these cases, let $\overline{w_{S'}} := \big((q_1^{S'}, p(q_1^{S'})), \ldots, (q_{|S'|}^{S'}, p(q_{|S'|}^{S'}))\big)$, and the i-successor be the state $((S', \overline{w_{S'}}), (X, G))$. (The simulation of the runs is reset to the states in S'.)

  This fully defines Δ_{trans}. We conclude the definition of Δ' by setting

$$\Delta_{\text{init}} := \Big\{(q_I, a, P', P'') \mid \Big(\big(\big(\{q_0\}, ((q_0, p(q_0)))\big), (X, G)\big), a, P', P''\Big) \in \Delta_{\text{trans}}, (X, G) \in Q_{\text{fix}}\Big\}.$$

- Let $\mathbb{P} \subseteq Q'$ be the set of states $((S, \overline{w_S}), (X, G))$ with $\overline{w_S} := \big(\big(q_1^S, p(q_1^S)\big), \ldots, \big(q_{|S|}^S, p(q_{|S|}^S)\big)\big)$, which are used when a class is entered (see the second case in the definition of Δ_{trans}). By a slight abuse of notation, we say for $q \in Q$ and $P \in Q'$ that $q \in P$ if $P = ((S, \overline{w}), (X, G))$ and $\overline{w} = ((q_1, p_1), \ldots, (q_{|S|}, p_{|S|}))$ with $q \in \{q_1, \ldots, q_{|S|}\}$. Then define $C' := \bigwedge_{P \in \mathbb{P}} P \approx P \ \wedge \ \bigwedge_{P,P' \in Q', q \not\approx q' \in C, q \in P, q' \in P'} P \not\approx P'$.
- Acc′ should verify, that each branch is accepting. For each branch in a run, there are two possibilities: Either there are infinitely many states in $\mathbb{P}$ or not. If there are, then the allowed transitions in G verify that each branch of the run of $\mathcal{A}$ that is simulated in this branch of the run of $\mathcal{A}'$ was accepting. In the case that there are only finitely many states from $\mathbb{P}$, the acceptance condition has to verify that in each of the branches of the run of $\mathcal{A}$ that are simulated along this branch of the run of $\mathcal{A}'$, the maximum priority visited infinitely often is even.
  Instead of directly defining Acc′, we use an approach similar to Lemma 6. Here we have to be very careful to not reintroduce new irreflexive constraints. First note that it is possible to construct a deterministic parity word automaton, that reads the states along the branches in a run of $\mathcal{A}'$, and verifies that (assuming no additional states from $\mathbb{P}$ are read) all described paths of $\mathcal{A}$ satisfy the acceptance condition of $\mathcal{A}$. This can easily be seen if one constructs a nondeterministic automaton for the complement language, i.e. guess an index $j \in 1, \ldots, |Q|$, and check whether the highest priority along this branch is odd. This automaton then can be determinized and complemented (see [23]). Let this deterministic word automaton be $\mathcal{B} = (S, Q, s_0, \delta, p_{\mathcal{B}})$. Define the deterministic parity word automaton $\mathcal{B}' := (S \cup \{\hat{s}\}, Q, s_0, \delta_{\mathcal{B}'}, p_{\mathcal{B}'})$ where

$$\delta_{\mathcal{B}'}(s,q) := \begin{cases} \hat{s} & \text{if } q \in \mathbb{P} \\ s_0 & \text{if } q \notin \mathbb{P} \text{ and } s = \hat{s} \\ \delta(s,q) & \text{if } q \notin \mathbb{P} \text{ and } s \neq \hat{s} \end{cases},$$

$$p_{\mathcal{B}'}(s) := p_{\mathcal{B}}(s) \text{ for all } s \in S \ \text{ and } \ p_{\mathcal{B}'}(\hat{s}) := \begin{cases} \max(p(S)) & \text{if } \max(p(S)) \text{ is even} \\ \max(p(S)) + 1 & \text{if } \max(p(S)) \text{ is odd.} \end{cases}$$

  That is, $\mathcal{B}'$ behaves like $\mathcal{B}$ on states in $Q \setminus \mathbb{P}$, and whenever a state from $\mathbb{P}$ is read, it transitions into $\hat{s}$. Thus this automaton accepts precisely those sequences of states that satisfy the acceptance condition Acc′. We then construct the automaton $\mathcal{A}' \ltimes \mathcal{B}'$ according to Lemma 6. A closer look at the construction of $\mathcal{A}' \ltimes \mathcal{B}'$ reveals, that this did introduce non-reflexive $\approx$-constraints of the form $(q, s) \approx (q, s')$ for $q \in \mathbb{P}$ and $s \in S \cup \{\hat{s}\}$. But by the definition of $\mathcal{B}'$, $\delta_{\mathcal{B}'}(s, q) = \hat{s}$ for all $q \in \mathbb{P}, s \in S \cup \{\hat{s}\}$, and therefore states (q, s) with $q \in \mathbb{P}$ and $s \neq \hat{s}$ are unreachable. Removing these states from $\mathcal{A}' \ltimes \mathcal{B}'$ does not change the recognized language, but eliminates all non-reflexive $\approx$-constraints. ◀

It turns out that Lemma 20 does not hold, if we allow reflexive $\not\approx$-constraints. One can show, that the language $\{t \in T_\Sigma^\omega \mid t|_0 = t|_1 \text{ and } t|_{0^i} \neq t|_{0^j} \text{ for all } i \neq j\}$ is not recognizable by a PTAGC with reflexive $\approx$-constraints.

Lemma 8, Lemma 20 and Lemma 17 combined with the fact that the emptiness problem for parity tree automata without constraints is decidable, imply the following theorem:

▶ **Theorem 22.** *The emptiness problem for parity-PTAGC without $\not\approx$-constraints is decidable.*

In order to compute a concrete witness for the non-empiness, the following result is important.

Table 1 Summary of our closure and decidability results.

	safety-		Büchi-	parity = Muller
	PTAGC	TAGC	PTAGC = TAGC	PTAGC = TAGC
Union ($\cup$)	✓	✓	✓	✓
Intersection ($\cap$)	✓	✓	✓	✓
Complement ($\neg$)	×	×	×	×
Emptiness ($= \emptyset?$)	✓without $\not\approx$	?	✓PTAGC without $\not\approx$	✓PTAGC without $\not\approx$
Universality ($= T^{\omega}_{\Sigma}?$)	?	×	×	×
Regularity	?	×	×	×

▶ **Corollary 23.** *Each nonempty language recognizable by a parity-PTAGC without disequality constraints contains a regular tree.*[1]

Proof. This Corollary follows from the proof of Lemma 17, since the regular tree obtained from the membership game of standard tree automata (see [23, 24]) is also contained in the language of the parity-PTAGC. ◀

5 Conclusion

We have made a first step in extending the theory of TAGC from finite to infinite trees. The conversion of acceptance conditions works in the same way as for tree automata without constraints. Our results on closure and decidability properties of TAGC are summarized in Table 1. As shown in the table, the models of TAGC and PTAGC have the same expressive power for Büchi- and parity conditions. Safety conditions are not preserved by our construction of PTAGC. In particular, we conjecture that Lemma 9 does not hold for safety-TAGC. In Example 3 we present a language that is recognizable by a safety-TAGC that makes use of Boolean negation in its constraint formula. We are convinced that this language is not recognizable by safety-PTAGC, but currently we only have a proof in the case that we disallow reflexive $\not\approx$-constraints. Showing that this language is not recognizable by safety-PTAGC would also show that safety-PTAGC are not closed under complement.

The main open problem that remains, is the emptiness problem in the presence of disequality constraints. The pumping arguments that are used in the case of finite trees [2], do not (directly) generalize to infinite trees because an infinite tree can be equal to one of its subtrees. So it seems that one needs to develop new methods to deal with disequality constraints on infinite trees.

References

1 Franz Baader and Alexander Okhotin. Solving Language Equations and Disequations with Applications to Disunification in Description Logics and Monadic Set Constraints. In *Logic for Programming, Artificial Intelligence, and Reasoning - 18th International Conference, LPAR-18*, volume 7180 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2012. doi:10.1007/978-3-642-28717-6.

2 Luis Barguñó, Carles Creus, Guillem Godoy, Florent Jacquemard, and Camille Vacher. The Emptiness Problem for Tree Automata with Global Constraints. In *Proceedings of the 25th*

[1] A regular tree is the unfolding of a finite graph, see [23, 24] for more precise definitions.

Annual IEEE Symposium on Logic in Computer Science, LICS 2010, pages 263–272. IEEE Computer Society, 2010.

3 Orna Bernholtz, Moshe Y. Vardi, and Pierre Wolper. An Automata-Theoretic Approach to Branching-Time Model Checking. In *Proceedings of the 6th International Conference on Computer Aided Verification, CAV '94*, volume 818 of *Lecture Notes in Computer Science*, pages 142–155. Springer, 1994.

4 Bruno Bogaert and Sophie Tison. Equality and Disequality Constraints on Direct Subterms in Tree Automata. In *Proceedings of STACS 92, 9th Annual Symposium on Theoretical Aspects of Computer Science*, volume 577 of *Lecture Notes in Computer Science*, pages 161–171. Springer, 1992.

5 Arnaud Carayol, Christof Löding, and Olivier Serre. Automata on Infinite Trees with Equality and Disequality Constraints Between Siblings. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 227–236, 2016. doi:10.1145/2933575.2934504.

6 Luca Cardelli and Giorgio Ghelli. TQL: a query language for semistructured data based on the ambient logic. *Mathematical Structures in Computer Science*, 14(3):285–327, 2004. doi:10.1017/S0960129504004141.

7 Emmanuel Filiot, Jean-Marc Talbot, and Sophie Tison. Satisfiability of a Spatial Logic with Tree Variables. In *Proceedings of Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL*, volume 4646 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2007. doi:10.1007/978-3-540-74915-8_13.

8 Emmanuel Filiot, Jean-Marc Talbot, and Sophie Tison. Tree Automata with Global Constraints. In *Proceedings of Developments in Language Theory, 12th International Conference, DLT 2008*, volume 5257 of *Lecture Notes in Computer Science*, pages 314–326. Springer, 2008. doi:10.1007/978-3-540-85780-8_25.

9 Emmanuel Filiot, Jean-Marc Talbot, and Sophie Tison. Tree Automata with Global Constraints. *Int. J. Found. Comput. Sci.*, 21(4):571–596, 2010. doi:10.1142/S012905411000743X.

10 Florent Jacquemard, Francis Klay, and Camille Vacher. Rigid tree automata and applications. *Inf. Comput.*, 209(3):486–512, 2011. doi:10.1016/j.ic.2010.11.015.

11 Barbara Jobstmann and Roderick Bloem. Optimizations for LTL Synthesis. In *Formal Methods in Computer-Aided Design, 6th International Conference, FMCAD 2006*, pages 117–124. IEEE Computer Society, 2006.

12 Michael Kaminski and Tony Tan. Tree Automata over Infinite Alphabets. In *Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, volume 4800 of *Lecture Notes in Computer Science*, pages 386–423. Springer, 2008.

13 Felix Klaedtke and Harald Rueß. Monadic Second-Order Logics with Cardinalities. In *Automata, Languages and Programming, 30th International Colloquium, ICALP 2003, Eindhoven, The Netherlands, June 30 - July 4, 2003. Proceedings*, volume 2719 of *Lecture Notes in Computer Science*, pages 681–696. Springer, 2003.

14 Orna Kupferman and Moshe Y. Vardi. Safraless Decision Procedures. In *46th Annual IEEE Symposium on Foundations of Computer Science, FOCS'05, Proceedings*, pages 531–542. IEEE Computer Society, 2005.

15 Patrick Landwehr and Christof Löding. Projection for Büchi Tree Automata with Constraints Between Siblings. In *Developments in Language Theory, DLT 2018*, volume 11088 of *Lecture Notes in Computer Science*, pages 478–490. Springer, 2018. doi:10.1007/978-3-319-98654-8.

16 Frank Nießner and Ulrich Ultes-Nitsche. A complete characterization of deterministic regular liveness properties. *Theor. Comput. Sci.*, 387(2):187–195, 2007. doi:10.1016/j.tcs.2007.07.038.

17 Amir Pnueli and Roni Rosner. On the Synthesis of a Reactive Module. In *Proceedings of the Symposium on Principles of Programming Languages, POPL'89*, pages 179–190, 1989.

18 Michael O. Rabin. Decidability of Second-Order Theories and Automata on Infinite Trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

19 Michael O. Rabin. *Automata on Infinite Objects and Church's Problem*. American Mathematical Society, Boston, MA, USA, 1972.

20 Helmut Seidl, Thomas Schwentick, and Anca Muscholl. Numerical document queries. In *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 9-12, 2003, San Diego, CA, USA*, pages 155–166. ACM, 2003.

21 James W. Thatcher and Jesse B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

22 Wolfgang Thomas. Automata on Infinite Objects. In *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 133–192. Elsevier Science Publishers, Amsterdam, 1990.

23 Wolfgang Thomas. Languages, Automata, and Logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Language Theory*, volume III, pages 389–455. Springer, 1997.

24 Moshe Y. Vardi and Thomas Wilke. Automata: from logics to algorithms. In *Logic and automata - history and perspectives*, volume 2 of *Texts in Logic and Games*, pages 629–724. Amsterdam University Press, 2007.

25 Margus Veanes and Nikolaj Bjørner. Symbolic tree automata. *Inf. Process. Lett.*, 115(3):418–424, 2015. doi:10.1016/j.ipl.2014.11.005.

26 Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998.