Automata on Infinite Trees with Equality and Disequality Constraints Between Siblings

Arnaud Carayol
LIGM (Univ. Paris Est & CNRS)
Arnaud.Carayol@univ-mlv.fr

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

Olivier Serre
IRIF (Univ. Paris Diderot & CNRS)
Olivier.Serre@cnrs.fr

Abstract

This article is inspired by two works from the early 90s. The first one is by Bogaert and Tison who considered a model of automata on *finite ranked* trees where one can check equality and disequality constraints between direct subtrees: they proved that this class of automata is closed under Boolean operations and that both the emptiness and the finiteness problem of the accepted language are decidable. The second one is by Niwinski who showed that one can compute the cardinality of any ω-regular language of infinite trees.

Here, we generalise the model of automata of Tison and Bogaert to the setting of *infinite* binary trees. Roughly speaking we consider parity tree automata where some transitions are guarded and can be used only when the two direct sub-trees of the current node are equal/disequal. We show that the resulting class of languages encompasses the one of ω-regular languages of infinite trees while sharing most of its closure properties, in particular it is a Boolean algebra. Our main technical contribution is then to prove that it also enjoys a decidable cardinality problem. In particular, this implies the decidability of the emptiness problem.

Categories and Subject Descriptors Theory of Computation [*Formal languages and automata theory*]: Tree languages

Keywords Automata on infinite trees, Automata with equality and disequality constraints, Emptiness problem, Finiteness problem.

1. Introduction

Finite automata on infinite trees are a powerful tool for decision procedures in logic and synthesis of finite state programs from logical specifications [19–21]. Formulas of monadic second-order logic (MSO) over infinite trees can be translated inductively into equivalent finite automata using the closure properties of this automaton class. Satisfiability of a formula then corresponds to the non-emptiness problem for the equivalent automaton, which is decidable, for example using game-theoretic techniques (see [22]). In [17] it is shown that not only the emptiness problem is decidable but one can even decide whether the language of an automaton on infinite trees is finite, countable, or uncountable (it is shown that no other cardinal is possible).

While finite automata on infinite trees enjoy very good algorithmic and closure properties, their expressive power is limited. One line of research studies extensions of automata on infinite trees, aiming at identifying more expressive models while retaining (most of) the algorithmic and closure properties. One important such extension, which has been studied in recent years, considers models that can express (un)boundedness properties [4–6, 8, 9, 23] that cannot be expressed with classical tree automata or MSO.

In the present paper, we consider a different kind of extension for automata on infinite trees, namely by a mechanism for comparing subtrees for equality.

The theory of tree automata with equality and disequality constraints on finite trees has been developed during the last two decades. The original motivation for such models was to develop tools and algorithms for non-linear term rewrite systems. Over the last two decades, the decidability results have been pushed to stronger and stronger models. As one remarkable result, the theory of these automata has provided tools for solving a long standing open question, namely the decidability of the "HOM problem" [14, 15], which asks for a given regular language T of finite trees and a tree homomorphism h, whether the image $h(T)$ of T under h is a regular tree language. Another motivation is the development of automaton models for capturing constraints in XML specification languages, like monadic key constraints for XML documents [1].

There are two types of equality and disequality constraints: local constraints, specified in the transitions of a tree automaton, and global constraints, evaluated on the level of runs of the automaton. Local constraints are of the form $u = v$ or $u \neq v$, where u and v are tree nodes. For example, a constraint $0 = 10$ is satisfied in a tree if the left subtree (addressed by the node 0) of the root is equal to the left subtree of the right subtree (addressed by the node 10) of the root. In tree automata with equality and disequality constraints (TAED), the transitions are enriched with such constraints, and can be executed at a tree node if all constraints are satisfied by the subtree rooted at this node. For the full class of TAED, the emptiness problem is undecidable (see [10]). For some restricted classes, however, positive algorithmic results have been obtained. In [2] the model restricted to constraints of direct subtrees is analysed and it is shown that emptiness and finiteness of the accepted language are decidable for this model. Furthermore, this class of automata is closed under Boolean operations. Another model with decidable emptiness problem is obtained when restricting the use of equality tests and allowing for arbitrary disequality tests. This class is referred to as reduction automata (see [10] and references therein).

A model with global constraints has been defined in [12, 13]. In these tree automata with global equality and disequality constraints (TAGED), the constraints are specified on the state space of the automaton. A constraint $q_1 = q_2$ (or $q_1 \neq q_2$) is satisfied in a run of the automaton if all pairs of subtrees that evaluate to q_1 and q_2, respectively, are equal (or different). Originally, this model has been defined in [11] for analysing a logic called TQL for querying

LICS '16, July 05-08, 2016, New York, NY, USA

DOI: http://dx.doi.org/10.1145/2933575.2934504

semi-structured data [7]. As it turned out to be a useful tool for deciding logics whose expressive power goes beyond MSO, the model has been further investigated in [12, 13], where decidability results for emptiness on restricted classes of TAGED have been obtained. These results have then been further generalised in [1] 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 with equality and disequality tests that characterise the class of TAGED, and the algorithmic results lead to decision procedures for these versions of MSO [1, 13].

This brief overview on tree automata with equality and disequality constraints over finite trees, shows that this is a very rich theory that has produced powerful decidability results with interesting applications.

In this paper, we make a first step of extending this theory to infinite trees. To the best of our knowledge, there is no existing work on this subject. We study the model with local constraints between siblings, which was the first one for the case of finite trees for which the emptiness problem was shown to be decidable [2].

For infinite trees, we prove that the cardinality problem (that asks for the cardinality of a given language) is decidable for parity tree automata with equality and disequality tests between siblings. We restrict to binary trees to simplify the presentation. In this case, a transition of such an automaton either has no constraint, or it has the constraint $=$ or $\neq$, meaning that the two subtrees of the current node are equal, resp. not equal.

We prove that the class of tree languages defined by these automata forms a Boolean algebra, and we show how equality constraints can be eliminated (making use of alternating tree automata [16]). Our main contributions are decidability results for the emptiness and finiteness problems for automata on infinite trees with equality and disequality constraints. The methods used for finite trees do not generalise to infinite trees. For our solution, we have to develop different methods for dealing with different acceptance conditions.

We first present an algorithm that identifies for each state of a given Büchi tree automaton with constraints whether the unconstrained language accepted from this state is empty, finite, countable or uncountable (the unconstrained language being the one accepted by the automaton in which the equality and disequality constraints are removed). Distinguishing these cases, we then iteratively refine the knowledge on the constrained language for each state: If the unconstrained language is empty or finite, one can directly check which trees are contained in the constrained language. Countable regular languages of infinite trees can be characterised in terms of regular languages of finite trees [17], and thus for this case we can build on the results for automata with constraints on finite trees. This allows us to test whether the constrained language is empty, finite, or countable in case the unconstrained language is countable. Based on the information gathered in such an iteration, we modify the automaton (possibly changing the unconstrained languages of the states but not the constrained ones) and run the test again for each state. If no new states are identified as empty, finite, or countable, we can prove that the remaining states all accept an uncountable constrained language, using a characterisation of uncountable tree languages from [17].

We then develop an algorithm for co-Büchi automata using a fixpoint computation that in each step invokes our algorithm for Büchi automata. Finally we sketch an extension of our ideas to the case of full parity tree automata.

The paper is structured as follows. In Section 2 we introduce basic definitions and some results on automata with equality and disequality constraints concerning closure properties and expressiveness. Section 3 then defines the decision problems concerning the cardinality of languages and gives some examples. In Section 4 we study the cardinality of constrained languages in the case that the unconstrained language is countable. Building on these results we then develop an algorithm for determining the cardinality of the constrained language of a Büchi tree automaton in Section 5. In Section 6 we present the ideas for solving the cardinality problem for co-Büchi automata, and in Section 7 we explain how to extend the ideas to the full class of parity tree automata.

2. Definitions and Preliminary Results

2.1 Words

An **alphabet** is a finite set A of letters. In the sequel A^* denotes the set of finite words over A, and A^ω the set of infinite words over A. The **empty word** is written ε; the length of a word u is denoted by $|u|$. For any $k \geq 0$, we let $A^k = \{u \mid |u| = k\}$, $A^{\leq k} = \{u \mid |u| \leq k\}$ and $A^{\geq k} = \{u \mid |u| \geq k\}$. We let $A^+ = A^* \setminus \{\varepsilon\}$. Let $u \in A^*$ be a finite word and $v \in A^* \cup A^\omega$ be a (possibly infinite) word. Then $u \cdot v$ (or simply uv) denotes the **concatenation** of u and v; the word u is a **prefix** of v, denoted $u \sqsubseteq v$, iff there exists a word w such that $v = u \cdot w$. We denote by $u \sqsubset v$ the fact that u is a strict prefix of v (*ie.* $u \sqsubseteq v$ and $u \neq v$).

2.2 Trees

Let A be some finite alphabet. An ***A*-labelled infinite tree** is a mapping $t : \{0,1\}^* \to A$. Elements of $\{0,1\}^*$ are called **nodes** and we refer to ε as the **root**. For a node u, we refer to $u0$ (*resp.* $u1$) as the **left son** (*resp.* **right son**) of u; the nodes $u0$ and $u1$ are said to be **siblings**. A **branch** is an infinite word in $\{0,1\}^\omega$.

For a given tree t and a node $u \in \{0,1\}^*$ we denote by $t[u]$ the **subtree** of t rooted at u, that is the tree defined by letting $t[u](v) = t(uv)$ for every $v \in \{0,1\}^*$. For a node u, we refer to $t[u0]$ (*resp.* $t[u1]$) as the left (*resp.* right) subtree rooted at u.

With any A-labelled tree t we associate a unique $(A \times \{=, \neq\})$-labelled tree denoted $t^{\stackrel{?}{=}}$ obtained by annotating every node u in t by an extra information regarding on whether the left and the right subtrees rooted at u are equal or not. More formally, for every $u \in \{0,1\}^*$ one lets

$$t^{\stackrel{?}{=}}(u) = \begin{cases} (t(u), =) & \text{if } t[u0] = t[u1] \\ (t(u), \neq) & \text{if } t[u0] \neq t[u1] \end{cases}$$

A **regular tree** is a tree t that contains finitely many distinct subtrees, *ie.* such that the set $\{t[u] \mid u \in \{0,1\}^*\}$ is finite. Let RegTrees denote the set of regular trees.

Example 1. *Let t be the $\{a,b\}$-labelled tree defined by $t(\varepsilon) = a$, $t(u0) = a$ and $t(u1) = b$ for every node $u \in \{0,1\}^*$. The tree t is regular because $t[u0] = t[0]$ and $t[u1] = t[1]$ for every node $u \in \{0,1\}^*$. One can also remark that $t^{\stackrel{?}{=}}(u) = (t(u), \neq)$ for every node $u \in \{0,1\}^*$.*

An *equivalent* definition of regular trees is as follows (see also [22]). Let $G = (V, E \subseteq V \times \{0,1\} \times V)$ be an edge-labelled finite directed graph, let r be a special vertex called the root and let $\xi \to A$ be a function assigning a label in A to every vertex. Assume moreover that E is deterministic, that is, for every vertex v there is exactly one vertex v_0 (*resp.* v_1) in V such that $(v, 0, v_0) \in E$ (*resp.* $(v, 1, v_1) \in E$). From (G, r, ξ) we define a regular tree t by letting $t(u) = \xi(v_u)$ where v_u is the (unique) vertex that is reached from r in G by following a path labelled by u. See Figure 1 for an example.

2.3 Tree Automata, ω-Regular Tree Languages

A (non-deterministic) **parity tree automaton** over an alphabet A is a tuple $\mathcal{A} = (Q, A, q_{in}, \Delta, \mathrm{Col})$ where Q is a finite set of control states, A is a finite labelling alphabet, $q_{in} \in Q$ is the initial state,

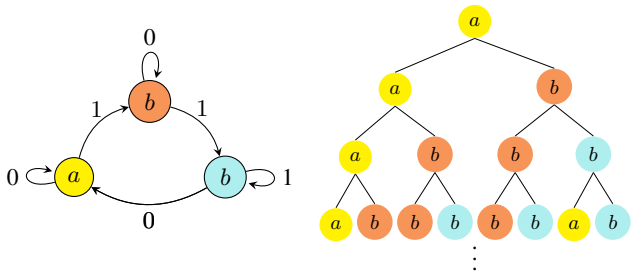

Figure 1: A graph (on the left) G; the regular tree associated with G where the root is the node labelled by a

$\Delta \subseteq Q \times A \times Q \times Q$ is the transition relation and $\mathrm{Col} : Q \to \mathbb{N}$ is a colouring function.

Let t be an infinite A-labelled tree. Then a run of $\mathcal{A}$ on t is a Q-labelled tree r such that $(r(u), t(u), r(u0), r(u1)) \in \Delta$ for every $u \in \{0,1\}^*$. A run r is accepting if $r(\varepsilon) = q_{in}$ and for every branch $\alpha_1\alpha_2 \cdots \in \{0,1\}^\omega$, $\liminf(\mathrm{Col}(r(\alpha_1 \cdots \alpha_i)))_{i\geq 0}$ (the least colour that appears infinitely often) is even. We say that the branch satisfies the parity condition. A tree t is **accepted** by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on t.

In case all states are assigned the same (even) colour by Col, we refer to $\mathcal{A}$ as a **safety** automaton; in case the colour given by Col are either 0 or 1, we refer to $\mathcal{A}$ as a **Büchi** automaton and to the states in $\mathrm{Col}^{-1}(0)$ as **final states** (and we may in that case describe Col by explicitly giving the final states instead of Col). A **co-Büchi** automaton is a parity automaton using colours 1 and 2.

Let q be a state. Say that q is **reachable** if it can be reached from the initial state by transitions. We say that q is **productive** if there is some tree t, some accepting run r over t and some node u such that $r(u) = q$.

We denote by $L(\mathcal{A})$ the set of all trees accepted by $\mathcal{A}$. We refer to such a language as an **ω-regular tree language** and we denote by $\mathbf{REG}$ the set of ω-regular languages. The following is a well-known result [20].

Theorem 1. *The class* $\mathbf{REG}$ *is an effective Boolean algebra.*

2.4 Tree Automata With Equality And Disequality Constraints Between Siblings

A parity **tree automaton with (equality and disequality) constraints** over an alphabet A is a parity tree automaton $\mathcal{A}$ over the alphabet $A \times \{=, \neq\}$. Hence, viewed as a standard tree automaton, it recognises a language of $A \times \{=, \neq\}$-labelled trees. However, it will mainly be used to define languages of A-labelled trees: for that we define

$$L^{con}(\mathcal{A}) = \{t \mid t^{\stackrel{?}{=}} \in L(\mathcal{A})\}$$

Remark 1. *An alternative way of thinking of an automaton with constraints processing an A-labelled tree is by considering it as using guards: a transition $(q, (a, \iota), q_0, q_1)$ can only be fired in a node labelled by a where both subtrees are equal* (resp. *different) in case ι is $=$* (resp. *$\neq$).*

In the following, we will refer to $L^{con}(\mathcal{A})$ as the **language recognised** by $\mathcal{A}$. Sometimes we explicitly refer to $L^{con}(\mathcal{A})$ as the **constrained language** of $\mathcal{A}$ to stress that it satisfies the constraints from the transitions. We denote by $\mathbf{REG}^{\stackrel{?}{=}}$ the class of languages recognised by automata with equality and disequality constraints. The following is an immediate consequence of the definition of $\mathbf{REG}^{\stackrel{?}{=}}$ and of Theorem 1.

Theorem 2. *The class* $\mathbf{REG}^{\stackrel{?}{=}}$ *is an effective Boolean algebra.*

We shall for ease of presentation write $(q, (a, \bot), q_0, q_1) \in \Delta$ to mean that both $(q, (a, =), q_0, q_1)$ and $(q, (a, \neq), q_0, q_1)$ belongs to Δ. Hence, $\bot$ can be understood as a no-constraint symbol.

Obviously, every standard tree automaton can easily be turned into an equivalent tree automaton with constraints (using $\bot$ in all transitions). Furthermore, it is clear that the language of $\{a, b\}$-labelled trees $L = \{t \mid t[0] = t[1]\}$ can be recognised by some tree automaton with constraints but not by a standard tree automaton. This is summarised in the following fact.

Fact 1. *The class* $\mathbf{REG}$ *is strictly included in* $\mathbf{REG}^{\stackrel{?}{=}}$.

Later on, it will be useful to consider the unconstrained version $\widehat{\mathcal{A}}$ of a tree automaton with constraints $\mathcal{A}$. In the spirit of Remark 1, $\widehat{\mathcal{A}}$ mimics $\mathcal{A}$ but ignores the guards. More formally, starting from $\mathcal{A} = (Q, A \times \{=, \neq\}, q_{in}, \Delta, \mathrm{Col})$ we let $\widehat{\mathcal{A}} = (Q, A, q_{in}, \Delta', \mathrm{Col})$ where $\Delta' = \{(q, a, q_0, q_1) \mid (q, (a, =), q_0, q_1) \in \Delta \text{ or } (q, (a, \neq), q_0, q_1) \in \Delta\}$. We refer to $L(\widehat{\mathcal{A}})$ as the **unconstrained language** of $\mathcal{A}$.

Fact 2. *For any automaton with equality and disequality constraints $\mathcal{A}$ one has $L^{con}(\mathcal{A}) \subseteq L(\widehat{\mathcal{A}})$.*

2.5 Extended Tree Automata

We want to allow our automata to directly check, in some node u, that $t[u]$ is equal to some regular tree. This will not add any expressive power to our model (both the constrained and the unconstrained one) but it will later substantially simplify the presentation. We first give the definition in the model without constraints.

An **extended tree automaton** is a tuple $\mathcal{A} = (Q, A, q_{in}, \Delta, \mathrm{Col})$ where Q is a finite set of control states, A is a finite labelling alphabet, $q_{in} \in Q$ is an initial state, $\mathrm{Col} : Q \to \mathbb{N}$ is a colouring function and Δ is a *finite* subset of $Q \times A \times (Q \cup \mathrm{RegTrees}) \times (Q \cup \mathrm{RegTrees})$ called the (extended) transition relation.

A run of $\mathcal{A}$ on t is a $(Q \cup \mathrm{RegTrees} \cup \{\sharp\})$-labelled tree r (where $\sharp$ is a dummy symbol) such that the following holds.

1. For every node $u \in \{0,1\}^*$ with $r(u) \in Q$, one has that $(r(u), t(u), r(u0), r(u1)) \in \Delta$;
2. For every node $u \in \{0,1\}^*$ with $r(u) \in \mathrm{RegTrees}$ one has $r(u) = t[u]$ and for every v such that $u \sqsubset v$ one has $r(u) = \sharp$.

A run r is accepting if $r(\varepsilon) = q_{in}$, and every branch $\alpha_1\alpha_2 \cdots$, such that $r(\alpha_1 \cdots \alpha_i) \in Q$ for all $i \geq 0$, satisfies the parity condition. A tree t is accepted by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on t.

Extended tree automata with equality and disequality constraints are defined in a similar way but working on $A \times \{=, \neq\}$-labelled tree; in particular we can safely assume that any regular trees used in the transition relation is of the form $t^{\stackrel{?}{=}}$ for some A-labelled regular tree t.

For every regular tree t, it is straightforward to build a safety tree automaton that accepts only t. This implies that extended tree automata are not more expressive than classical ones.

Fact 3. *For every extended tree automaton (possibly with constraints) one can build a (non-extended) tree automaton that recognises the same language and uses the same set of colours in its acceptance condition.*

2.6 Getting Rid of Equality Constraints

In this paper we are interested in computing the cardinality of languages of the form $L^{con}(\mathcal{A})$. For that we will consider special cases of automata, **automata with disequality everywhere**, that

correspond to automata with a transition relation $\Delta \subseteq Q \times (A \times \{\neq\}) \times Q \times Q$. Obviously, these automata are strictly less expressive than the full class of automata with equality and disequality constraints. However, Theorem 3 below shows that one can remove equality constraints and even require disequality constraints in all transitions without changing the cardinality of the language.

Theorem 3. *Let $\mathcal{A}$ be a parity automaton with equality and disequality constraints. Then one can build an automaton $\mathcal{B}$ with disequality everywhere (over an alphabet with two new symbols) that is such that $L^{con}(\mathcal{A})$ and $L^{con}(\mathcal{B})$ have the same cardinality. If $\mathcal{A}$ is a safety (*resp. *Büchi) automaton, then so is $\mathcal{B}$.*

Proof sketch. We only sketch the idea.

We use the concept of alternating automata, which can send several states into the same direction in the tree. These states then independently produce runs on the same subtree. In [16] it is shown that this does not increase the expressive power of the automaton model: each alternating parity automaton can be transformed into an equivalent nondeterministic one. Furthermore, when starting from an alternating safety or Büchi automaton, the equivalent nondeterministic one has the same type of acceptance condition.

From a parity automaton with equality and disequality constraints $\mathcal{A}$, we build an alternating automaton that replaces transitions $(q, (a, =), q_0, q_1)$ with a transition $(q, (a, \neq), \{q_0, q_1\}, \zeta)$ that sends both q_0 and q_1 to the left subtree (thus checking whether both can accept the left subtree), and checks that the right subtree is a fixed regular tree ζ over a new alphabet of two symbols. Furthermore, ζ is chosen such that it satisfies disequality everywhere. One can show that the resulting language has the same cardinality as the one of $\mathcal{A}$.

We then apply the result from [16] to obtain an equivalent nondeterministic automaton. □

2.7 The Regular Tree Membership Problem

Let $\mathcal{A}$ be an extended automaton with constraints and let t be a *regular* tree. The following result shows that we can decide whether $t \in L^{con}(\mathcal{A})$.

Proposition 1. *Let $\mathcal{A}$ be an extended automaton with equality and disequality constraints and let t be a regular tree. Then one can decide whether $t \in L^{con}(\mathcal{A})$.*

Proof. Let $(G = (V, E), r, \xi)$ be a finite graph defining t. We can build $t^{\stackrel{?}{=}}$ by checking for each node v whether the two subtrees are equal or different. Checking whether (G, v_0, ξ) and (G, v_1, ξ) are equal for the two successor nodes v_0, v_1 of v is the same as checking (strong) bisimilarity of the associated transitions systems, which can even be done efficiently (see *eg.* [18]). By definition of $L^{con}(\mathcal{A})$, we now need to check whether $t^{\stackrel{?}{=}} \in L(\mathcal{A})$. This can be done, for example, by constructing a safety tree automaton that accepts only $t^{\stackrel{?}{=}}$, and then test the intersection with $\mathcal{A}$ for emptiness, which is decidable (see [22]). □

3. Cardinality Problems for Constrained Languages

One of the most important decision problems for automata is the **emptiness problem** (decide whether the accepted language of a given automaton empty). In terms of logic, this corresponds to the satisfiability problem. Furthermore, many other decision problems reduce to the emptiness problem by applying Boolean operations (for example, the equivalence problem for automata). A generalisation of the emptiness problem is the cardinality problem:

Definition 1 (Cardinality Problem)**.** *The cardinality problem asks, for a given automaton with equality and disequality constraints $\mathcal{A}$ to compute the cardinality of $L^{con}(\mathcal{A})$.*

Obviously, the decidability of the cardinality problem implies the decidability of the emptiness problem. Furthermore, it generalises the finiteness problem, which is to decide for a given automaton whether its language is finite.

3.1 Cardinality Profiles

We prove in the following sections that the cardinality problem is decidable for parity automata with equality and disequality constraints $\mathcal{A}$. We make use of the **cardinality profile** $\kappa_{\mathcal{A}}$ of $\mathcal{A}$, which is a mapping that assigns to each state q of $\mathcal{A}$ the cardinality of $L^{con}(\mathcal{A}_q)$. Denote by $\aleph_0$ the cardinality of the set of natural numbers, and by $2^{\aleph_0}$ the cardinality of the set of the real numbers. We prove that $L^{con}(\mathcal{A}_q)$ is either finite or in $\{\aleph_0, 2^{\aleph_0}\}$, that is, $\kappa_{\mathcal{A}}$ is a mapping $\kappa_{\mathcal{A}} : Q \to \mathbb{N} \cup \{\aleph_0, 2^{\aleph_0}\}$ (for regular languages of infinite trees this is shown in [17]).

In Section 5 we give an algorithm that directly computes the cardinality profile for Büchi automata with disequality everywhere. It relies on computing the cardinalities of the unconstrained languages for each state. In the case where $L(\widehat{\mathcal{A}_q})$ is countable, we propose in Section 4 a technique to compute the cardinality of $L^{con}(\mathcal{A}_q)$ (which works for general parity automata, not only Büchi automata). In case $L^{con}(\mathcal{A}_q)$ is finite, q is replaced in $\mathcal{A}$ by the finitely many regular trees it accepts. Iterating this process finally yields the cardinality profile.

The two examples below illustrate that the cardinality of the constrained and unconstrained language of a state can differ.

Example 2. *Let t_a be the regular tree whose root is labelled by a and such that any left child is labelled by a and any right child is labelled by b,* ie. *$t_a(\varepsilon) = a$, $t_a(u0) = a$ and $t_a(u1) = b$ for any $u \in \{0, 1\}^*$. Let t_b be defined similarly but requiring that the root is labelled by b. Note that $t_a[0] = t_a$ and $t_a[1] = t_b$ (*resp. *$t_b[0] = t_a$ and $t_b[1] = t_b$). Also remark that any siblings in t_a (*resp. *t_b) are distinct.*

Let $\mathcal{A}$ be the (extended) safety automaton with disequality constraints everywhere $(Q_{\mathcal{A}}, \{(a, \neq), (b, \neq)\}, q_{in}, \Delta_{\mathcal{A}}, \mathrm{Col})$ where $Q_{\mathcal{A}} = \{q_{in}, q_b\}$ and transitions $(q_{in}, (a, \neq), q_{in}, t_b)$, $(q_{in}, (a, \neq), q_b, t_b)$, $(q_b, (b, \neq), t_a, t_b)$. Then $L(\widehat{\mathcal{A}})$ consists of those $\{a, b\}$-labelled trees such that every right subtree is equal to t_b, and the leftmost branch contains at most one b (which is not at the root). Thus, $|L(\widehat{\mathcal{A}})| = \aleph_0$.

However, the constrained language consists only of a single tree, namely $L^{con}(\mathcal{A}) = \{t_a\}$, which is the tree that does not contain any b on the leftmost branch. This follows from the fact that a b-node on the leftmost branch would induce a left subtree equal to t_b, violating the disequality constraint at the parent.

This gives an example of a safety automaton such that $|L(\widehat{\mathcal{A}})| = \aleph_0$ while $|L^{con}(\mathcal{A})|$ is finite (in this case equal to 1 but one could easily get any finite cardinality).

Example 3. *Building on top of Example 2, we can construct a safety automaton with disequality constraints everywhere $\mathcal{B}$ such that $|L(\widehat{\mathcal{B}})| = 2^{\aleph_0}$ while $L^{con}(\mathcal{B})$ is empty. Indeed, assume that $\mathcal{B}$ works by checking that the leftmost branch is labelled only by c's and that any right subtree of a node on that branch is such that the root is labelled by c, the left subtree is t_a while the right subtree is accepted by the automaton $\mathcal{A}$ from Example 2:*

$\mathcal{B} = (Q_{\mathcal{B}}, \{(a, \neq), (b, \neq), (c, \neq)\}, q_c, \Delta_{\mathcal{B}}, \mathrm{Col})$ with $Q_{\mathcal{B}} = Q_{\mathcal{A}} \cup \{q_c, q_c'\}$ and $\Delta_{\mathcal{B}} = \Delta_{\mathcal{A}} \cup \{(q_c, (c, \neq), q_c, q_c'), (q_c', (c, \neq), t_a, q_{in})\}$.

Now, because $L(\widehat{\mathcal{A}})$ contains more than one tree different from t_a, it is easily seen that $|L(\widehat{\mathcal{B}})| = 2^{\aleph_0}$ (because for every node in

0^*11 *we have two different possible trees to plug in). But because* $L^{con}(\mathcal{A}) = \{t_a\}$ *it directly follows that* $L^{con}(\mathcal{B}) = \emptyset$.

3.2 Infinity Profiles

In Section 6 we show for co-Büchi automata how to compute a profile that does not distinguish the two infinite cardinalities, we call this the infinity profile.

Formally, the **infinity profile** $\mathbf{p}_\mathcal{A}$ of $\mathcal{A}$ is defined by letting for every state $q \in Q$,

$$\mathbf{p}_\mathcal{A}(q) = \begin{cases} L^{con}(\mathcal{A}_q) & \text{if } |L^{con}(\mathcal{A}_q)| < \infty \\ \infty & otherwise \end{cases}$$

Note that for states with finite languages it does not only assign the cardinality of the language, but the language itself. Since we want to compute with these profiles for parity automata with disequality everywhere, we first show that such languages only consist of regular trees, and thus can be explicitly represented.

Lemma 4. *Let* $\mathcal{A}$ *be a parity automaton with disequality everywhere. If* $L^{con}(\mathcal{A})$ *is finite, then it contains only regular trees.*

Proof. Assume that $L^{con}(\mathcal{A})$ contains a non-regular tree t. Let r be an accepting run of $\mathcal{A}$ on t. Since t is non-regular, it has infinitely many different subtrees. Hence, there is a state q and infinitely many nodes $u_1, u_2, \cdots$ such that all the subtrees $t[u_i]$ are different and $r(u_i) = q$ for all i. This means that $L^{con}(\mathcal{A}_q)$ is infinite. If we now pick one occurrence of q at a node u in r, we can replace the subtree by infinitely many different subtrees that are all accepted from q while retaining the disequality constraints (there are only finitely many nodes above u at which disequality constraints have to be satisfied). Hence, $L^{con}(\mathcal{A})$ is also infinite. □

The next result shows that it is possible to compute the cardinality profile of an automaton from its infinity profile. The proof is a direct construction of either $2^{\aleph_0}$ many different trees or the proof that there are only countably many.

Proposition 2. *Let* $\mathcal{A} = (Q, A \times \{\neq\}, q_{in}, \Delta, \mathrm{Col})$ *be a parity automaton with disequality everywhere and let* $\mathbf{p}_\mathcal{A}$ *be its infinity profile. Then for every state* $q \in Q$ *one can compute the cardinality of* $L^{con}(\mathcal{A}_q)$*, which belongs to* $\mathbb{N} \cup \{\aleph_0, 2^{\aleph_0}\}$.

Hence, computing the infinity profile is sufficient in order to solve the cardinality problem.

4. The Case of Countable Unconstrained Languages

In this section, we show that we can determine the cardinality of $L^{con}(\mathcal{A})$ for a parity tree automaton with equality and disequality constraints under the assumption that $L(\widehat{\mathcal{A}})$ is countable. For this, we reduce the computation of the cardinality of $L^{con}(\mathcal{A})$ to a similar question but for automata on finite trees (for which the solution is known from [2]). So we first recall the setting and central results for automata with constraints on finite trees. Then we explain a characterisation of ω-regular tree languages of countable cardinality from [17]. Finally we give the reduction.

4.1 Finite Trees

A **tree domain** is a prefix-closed subset Dom of $\{0,1\}^*$ such that for every $u \in \{0,1\}^*$ one has $u0 \in Dom \Leftrightarrow u1 \in Dom$.

We use two finite alphabets A_2 and A_0 for the internal (binary) nodes and the leafs, respectively. An **(A_2, A_0)-labelled finite tree** is a mapping $t : Dom \to A_2 \cup A_0$ where Dom is a *finite* tree domain, and $t(u) \in A_2$ iff $u0, u1 \in Dom$. Elements of Dom are called **nodes**; we refer to ε as the **root** and to maximal elements (for prefix ordering) in Dom as leaves. For a node u, we refer (when exists) to $u0$ (*resp.* $u1$) as the **left child** (*resp.* **right child**) of u.

For a node $u \in Dom$ we denote by $t[u]$ the subtree of t rooted at u, that is, the finite tree with domain $u^{-1}Dom = \{v \mid uv \in Dom\}$) defined by letting $t[u](v) = t(uv)$ for any $v \in u^{-1}Dom$. For a (non-leaf) u, we refer to $t[u0]$ (*resp.* $t[u1]$) as the left (*resp.* right) subtree rooted at u.

A (non-deterministic) **finite-tree automaton** over (A_2, A_0) is a tuple $\mathcal{A} = (Q, A_2, A_0, q_{in}, \Delta, Acc)$ where Q is a finite set of control states, $q_{in} \in Q$ is an initial state, $\Delta \subseteq Q \times A_2 \times Q \times Q$ is the transition relation and $Acc \subseteq A_0 \times Q$ is an acceptance condition.

Let $t : Dom \to A_2 \cup A_0$ be a finite (A_2, A_0)-labelled tree. Then a run of $\mathcal{A}$ on t is a mapping $r : Dom \to Q$ such that $(r(u), t(u), r(u0), r(u1)) \in \Delta$ for every node $u \in \{0,1\}^*$ that is not a leaf. A run r is accepting if $r(\varepsilon) = q_{in}$, and for every leaf $u \in Dom$ one has $(t(u), r(u)) \in Acc$. A tree t is accepted by $\mathcal{A}$ if there exists an accepting run of $\mathcal{A}$ on t. We denote by $L(\mathcal{A})$ the set of all trees accepted by $\mathcal{A}$. We refer to such a language as a **regular tree language**.

With any (A_2, A_0)-labelled finite tree t we associate an $(A_2 \times \{=, \neq\}, A_0)$-labelled tree denoted $t^{\stackrel{?}{=}}$ obtained by annotating every binary node u in t by an extra information regarding on whether the left and the right subtrees rooted at u are equal or not (or whether u is a leaf). More formally, for every $u \in \{0,1\}^*$ one lets

$$t^{\stackrel{?}{=}}(u) = \begin{cases} t(u) & \text{if } u \text{ is a leaf} \\ (t(u), =) & \text{if } t[u0] = t[u1] \\ (t(u), \neq) & \text{if } t[u0] \neq t[u1] \end{cases}$$

A **finite-tree automaton with (equality and disequality) constraints** over an alphabet A is a finite-tree automaton $\mathcal{A}$ over $(A_2 \times \{=, \neq\}, A_0)$. As for infinite trees, we define the constrained language $L^{con}(\mathcal{A})$ of (A_2, A_0)-labelled finite trees by

$$L^{con}(\mathcal{A}) = \{t \mid t^{\stackrel{?}{=}} \in L(\mathcal{A})\}$$

Bogaert and Tison proved in [2] that one can compute the cardinality of such languages.

Theorem 4 ([2])**.** *Let* $\mathcal{A}$ *be a finite-tree automaton with equality and disequality constraints. Then one can compute* $|L^{con}(\mathcal{A})|$*. Moreover, in case it is finite* $L^{con}(\mathcal{A})$ *can be computed.*

4.2 Countable ω-Regular Tree Languages

We first recall a result due to Niwinski [17] characterising countable ω-regular tree languages.

Let $X = \{x_1, \ldots, x_\ell\}$ be a set of substitution symbols that we use as leaf alphabet A_0. Let $t : Dom \to A \cup X$ be an (A, X)-labelled finite tree and let $t_1, \ldots, t_\ell$ be some A-labelled infinite trees. Then we denote by $t_{[x_1/t_1, \ldots, x_\ell/t_\ell]}$ the infinite A-labelled tree built from t by replacing every leaf labelled by x_i by the tree t_i. Formally: $t_{[x_1/t_1, \ldots, x_\ell/t_\ell]}(u) = t(u)$ if $u \in Dom$ is not a leaf and $t_{[x_1/t_1, \ldots, x_\ell/t_\ell]}[u] = t_i$ if u is a leaf with $t(u) = x_i$.

Theorem 5 ([17])**.** *Let* $\mathcal{B}$ *be a parity tree automaton (without constraints) over an alphabet* A*. Then one can compute the cardinality of* $L(\mathcal{B})$*. Moreover, if this cardinality is at most countable one can compute the following:*

- *A number* ℓ *and a finite set* $\{t_1, \ldots, t_\ell\}$ *of regular trees over an alphabet* A*;*
- *A finite-tree automaton* $\mathcal{C}$ *over* (A, X) *with* $X = \{x_1, \ldots, x_\ell\}$ *such that* $L(\mathcal{B}) = \{t_{[x_1/t_1, \ldots, x_\ell/t_\ell]} \mid t \in L(\mathcal{C})\}$.

Fix an automaton $\mathcal{B} = (Q, A, q_{in}, \Delta, \mathrm{Col})$ that recognises a countable set of infinite trees and a set $B = \{t_1, \ldots, t_\ell\}$ as in Theorem 5. Without loss of generality we can assume that B is

closed under subtrees (*ie.* for every $t \in B$ and every node u, $t[u] \in B$): if needed, one adds the missing regular subtrees to B. Also fix a set $X = \{x_1, \ldots, x_\ell\}$ as in Theorem 5. For every (A, X)-labelled finite tree t, we say that t is **B-reduced** if for every non-leaf node u in t, the infinite tree $t_{[x_1/t_1,\ldots,x_\ell/t_\ell]}[u]$ does not belong to B.

Lemma 5. *For every (A, X)-labelled finite tree t there exists a unique (A, X)-labelled finite B-reduced tree t' such that*

$$t_{[x_1/t_1,\ldots,x_\ell/t_\ell]} = t'_{[x_1/t_1,\ldots,x_\ell/t_\ell]}$$

Proof. The tree t' is built by considering all nodes u in t by length lexicographic ordering and replacing $t[u]$ by a single leaf labelled x_i whenever $t[u] = t_i$ for some $t_i \in B$. Uniqueness comes from the fact that two different B-reduced trees t^1 and t^2 lead to different trees $t^1_{[x_1/t_1,\ldots,x_\ell/t_\ell]}$ and $t^2_{[x_1/t_1,\ldots,x_\ell/t_\ell]}$. □

Lemma 6. *The set of B-reduced trees is regular.*

Proof. Let $a \in A$ and t_0, t_1 be two trees: denote by $a(t_0, t_1)$ the tree with root labelled by a, left son t_0 and right son t_1. Note that a tree t is B-reduced if and only if every node labelled by some a with two children leaves labelled by some x_0,x_1 is such that $a(t_0, t_1) \notin B$ for $t_0, t_1 \in B$. Hence an automaton accepting B-reduced trees simply check this property. □

Lemma 7. *The following set is regular:*

$$T_L = \{t \mid t \text{ is an } (A, X)\text{-labelled finite tree s.t. } t_{[x_1/t_1,\ldots,x_\ell/t_\ell]} \in L(\mathcal{B})\}$$

Proof. For every state q of $\mathcal{B}$, call $\mathcal{B}_q$ the automaton obtained from $\mathcal{B}$ by taking q as the initial state. For every tree t_i let Q_i be the set of states q such that $\mathcal{B}_q$ accepts t_i. An automaton $\mathcal{B}' = (Q, A, q_{in}, \Delta, Acc)$ accepting the set T_L simply mimics $\mathcal{B}$; acceptance at the leaf requiring that a variable x_i corresponds to a state $q \in Q_i$, *ie.* we let $Acc = \bigcup_{i=1\ldots\ell} \{x_i\} \times Q_i$. □

Lemma 8. *There exists a regular language Rd of (A, X)-labelled finite trees such that*

(i) Rd consists only of reduced trees;
(ii) $L(\mathcal{B}) = \{t_{[x_1/t_1,\ldots,x_\ell/t_\ell]} \mid t \in Rd\}$;
(iii) Rd and $L(\mathcal{B})$ have the same cardinal.

Proof. Define Rd to be the set of B-reduced trees intersected with the language T_L from Lemma 7. It is regular because it is defined as the intersection of two regular languages. Item (i) is by definition, while items (ii) and (iii) follows from Lemma 7 and Lemma 5 (together with the fact that two different B-reduced trees, lead to two different infinite tree after substitution of the x_i's by the t_i's). □

4.3 Back to Automata with Equality and Disequality Constraints

The following fact relates the sibling equalities for a B-reduced tree t and its infinite version $t_{[x_1/t_1,\ldots,x_\ell/t_\ell]}$ after substitution .

Fact 9. *Let t be an (A, X)-labelled finite B-reduced tree. Then for every non-leaf node u in t one has $t^{\stackrel{?}{=}}(u) = t^{\stackrel{?}{=}}_{[x_1/t_1,\ldots,x_\ell/t_\ell]}(u)$.*

We are now ready to go back to automata with equality and disequality constraints and give an analogue of Lemma 8.

Lemma 10. *Let $\mathcal{A}$ be a parity automaton with equality and disequality constraints such that $L(\widehat{\mathcal{A}})$ is countable. One can construct an automaton $\mathcal{C}$ with equality and disequality constraints on finite trees such that*

(i) $L^{con}(\mathcal{C})$ consists only of reduced trees;
(ii) $L^{con}(\mathcal{A}) = \{t_{[x_1/t_1,\ldots,x_\ell/t_\ell]} \mid t \in L^{con}(\mathcal{C})\}$;
(iii) $L^{con}(\mathcal{C})$ and $L^{con}(\mathcal{A})$ have the same cardinality.

Proof. Apply Lemma 8 to $\mathcal{A}$ viewed as a standard parity tree automaton for $(A \times \{=, \neq\})$-labelled trees. The resulting set Rd comes with an automaton $\mathcal{C}$. Fact 9 implies that the properties transfer to the constrained languages of $\mathcal{A}$ and $\mathcal{C}$. □

Combining Lemma 10 and Theorem 4 directly leads to the following result.

Theorem 6. *Let $\mathcal{A}$ be an automaton with equality and disequality constraints on infinite trees. If $L(\widehat{\mathcal{A}})$ is countable, then one can compute the cardinality of $L^{con}(\mathcal{A})$, and in case it is finite $L^{con}(\mathcal{A})$ consists only of regular trees and can be effectively computed.*

5. Büchi Automata

We now turn to the solution of the cardinality problem for Büchi automata. We give an algorithm that takes as input a Büchi automaton $\mathcal{A}$ with disequality constraints everywhere (we can always safely assume this thanks to Theorem 3) and outputs its cardinality profile. The algorithms identifies states with a countable unconstrained language and determines the cardinality of the constrained language based on the techniques developed in Section 4. If the constrained language is finite, it basically replaces the state by the finitely many regular trees in the constrained language. This is iterated until no new states with countable unconstrained language are found.

We start by introducing two operations to be applied for states with a finite constrained language. Then we present the algorithm in more detail, and finally prove its correctness.

5.1 Replacing States by Finite Languages

We now give two constructions used in our algorithm. Each of them takes an (extended) automaton with disequality everywhere $\mathcal{A}$ and produces another automaton $\mathcal{B}$ that is such that (when used in the right context) (i) $L^{con}(\mathcal{A}) = L^{con}(\mathcal{B})$; and (ii) $L(\widehat{\mathcal{B}}) \subseteq L(\widehat{\mathcal{A}})$.

5.1.1 The operation $\mathcal{A} \mapsto \mathcal{A}_{q\mapsto\emptyset}$

Let $\mathcal{A} = (Q, A, q_{in}, \Delta, \mathrm{Col})$ be an extended automaton with disequality everywhere. Let $q \in Q$ be a state such that $L^{con}(\mathcal{A}_q) = \emptyset$. Then we define a new automaton $\mathcal{A}_{q\mapsto\emptyset} = (Q\backslash\{q\}, A, q_{in}, \Delta', \mathrm{Col})$ where Δ' is obtained from Δ by only keeping transitions that do not involve q, *ie.* $\Delta' = \{(p, (a, \neq), p_0, p_1) \in \Delta \mid p, p_0, p_1 \neq q\}$.

The following is immediate.

Fact 11. *For every extended automaton with disequality everywhere $\mathcal{A}$ and for every state q such that $L^{con}(\mathcal{A}_q) = \emptyset$, one has that $L^{con}(\mathcal{A}) = L^{con}(\mathcal{A}_{q\mapsto\emptyset})$ and $L(\widehat{\mathcal{A}_{q\mapsto\emptyset}}) \subseteq L(\widehat{\mathcal{A}})$.*

5.1.2 The operation $\mathcal{A} \mapsto \mathcal{A}_{q\mapsto t_1,\ldots,t_n}$

Let $\mathcal{A} = (Q, A, q_{in}, \Delta, \mathrm{Col})$ be an extended automaton with disequality everywhere. Let $q \in Q$ be a state such that $L^{con}(\mathcal{A}_q) = \{t_1, \ldots, t_n\}$ is a finite set of regular trees. Then we define a new automaton $\mathcal{A}_{q\mapsto t_1,\ldots,t_n} = (Q \setminus \{q\}, A, q_{in}, \Delta', \mathrm{Col})$ where Δ' is obtained from Δ by replacing every transition of the form $(p, (a, \neq), q_0, q_1)$ with q_0 and/or q_1 being equal to q by all the transitions obtained by substituting occurrences of q with elements of $\{t_1, \ldots, t_n\}$, where in case $q_0 = q_1 = q$ the trees substituted for the two occurrences of q have to be different.

The following is immediate

Fact 12. *For every extended automaton with disequality everywhere $\mathcal{A}$ and for every state q such that $L^{con}(\mathcal{A}_q) = \{t_1, \ldots, t_n\}$, one has that $L^{con}(\mathcal{A}) = L^{con}(\mathcal{A}_{q\mapsto t_1,\ldots,t_n})$ and $L(\widehat{\mathcal{A}_{q\mapsto t_1,\ldots,t_n}}) \subseteq L(\widehat{\mathcal{A}})$.*

Algorithm 1 Solve the cardinality problem for Büchi automata

Input: Tree automaton with disequality constraints everywhere $\mathcal{A} = (Q, A, q_{in}, \Delta, Acc)$
Data Structure:
Set $S \leftarrow Q$
Automaton $\mathcal{B} \leftarrow \mathcal{A}$
Function $\kappa : Q \to \mathbb{N} \cup \{\aleph_0, 2^{\aleph_0}\}$; $\kappa(q) \leftarrow 2^{\aleph_0}$ for all q
Code:
1: **while** $\exists q \in S$ s.t. $|L(\widehat{\mathcal{B}_q})| \leq \aleph_0$ **do**
2: $\quad\kappa(q) \leftarrow |L^{con}(\mathcal{B}_q)|$
3: $\quad$**if** $\kappa(q) = 0$ **then**
4: $\quad\quad\mathcal{B} \leftarrow \mathcal{B}_{q \mapsto \emptyset}$
5: $\quad$**else if** $\kappa(q) < \aleph_0$ **then**
6: $\quad\quad$Let $L^{con}(\mathcal{B}_q) = \{t_1, \ldots, t_n\}$
7: $\quad\quad\mathcal{B} \leftarrow \mathcal{B}_{q \mapsto t_1, \ldots, t_n}$
8: $\quad$**end if**
9: $\quad S \leftarrow S \setminus \{q\}$
10: **end while**
11: **return** κ

5.2 Main Algorithm

Our algorithm (Algorithm 1) takes as input an automaton with disequality constraints everywhere $\mathcal{A} = (Q, A, q_{in}, \Delta, Acc)$. The algorithm identifies states whose unconstrained language is countable (which is decidable according to Theorem 5), and then determines the cardinality of the constrained language (Theorem 6). States for which the constrained language is finite are substituted by the regular trees in this language (which can be computed according to Theorem 6) using the operation $\mathcal{A}_{q \mapsto t_1, \ldots, t_n}$, and states with empty constrained language are eliminated using the operation $\mathcal{A}_{q \mapsto \emptyset}$. Note that these states remain in the state set because we want to keep the set fixed. However, they become unreachable by these operations.

The modifications do not change the constrained language accepted by the automaton (Facts 11 and 12). However, it might change the unconstrained languages. In particular, an unconstrained language that was uncountable may become countable by such a modification (see Example 4 below). The algorithm iterates this process until no new states with countable unconstrained language are found, and returns a cardinality profile κ. We prove that if the acceptance condition in $\mathcal{A}$ is a Büchi condition, then κ is the cardinality profile of $\mathcal{A}$. This proof basically amounts to showing that the states for which the unconstrained language is uncountable upon termination of the algorithm, also have an uncountable constrained language.

Example 4. *Consider the following execution of Algorithm 1 on automaton $\mathcal{B}$ of Example 3. It first detects that $|L(\widehat{\mathcal{B}_{q_{in}}})| \leq \aleph_0$, computes $\kappa(q_{in}) = 1$ and therefore modifies $\mathcal{B}$ by changing it to $\mathcal{B}_{q_{in} \mapsto t_a}$. Next (for the new $\mathcal{B}$), $|L(\widehat{\mathcal{B}_{q'_c}})| \leq \aleph_0$ and the algorithm computes $\kappa(q'_c) = 0$. Then, the algorithm detects that $\kappa(q_c) = 0$. Recall that the unconstrained language of q_c is uncountable in the original automaton. Finally, it detects that $\kappa(q_b) = 1$.*

5.3 Correctness

Fix an automaton with disequality constraints everywhere $\mathcal{A} = (Q, A, q_{in}, \Delta, Acc)$ and let $\mathcal{B}$ denote the automaton $\mathcal{B}$ at the end of the execution of Algorithm 1.

For the correctness proof, we define a partition $Q_\emptyset \uplus Q_\mathsf{f} \uplus Q_\mathsf{c} \uplus Q_\mathsf{u}$ of Q by letting $Q_\emptyset = \{q \mid \kappa(q) = 0\}$, $Q_\mathsf{f} = \{q \mid 1 \leq \kappa(q) < \aleph_0\}$, $Q_\mathsf{c} = \{q \mid \kappa(q) = \aleph_0\}$ and $Q_\mathsf{u} = \{q \mid \kappa(q) = 2^{\aleph_0}\}$.

Note that the states from $Q_\emptyset$ and Q_f are not reachable anymore in $\mathcal{B}$ because they have been substituted by the regular trees from their languages in the transitions.[1]

Remark that thanks to Fact 11 and Fact 12 we get the following invariant for Algorithm 1.

Lemma 13. *At any moment in the execution of Algorithm 1 one has $L^{con}(\mathcal{B}_q) = L^{con}(\mathcal{A}_q)$.*

The preceding invariant directly implies the correctness of the sets $Q_\emptyset$, Q_f and Q_c.

Lemma 14. *For every $q \in Q_\emptyset \cup Q_\mathsf{f} \cup Q_\mathsf{c}$, one has $\kappa(q) = \kappa_\mathcal{A}(q)$.*

The next lemma completes the picture.

Lemma 15. *If $\mathcal{A}$ is equipped with a Büchi condition, then for every state $q \in Q_\mathsf{u}$ one has $\kappa(q) = \kappa_\mathcal{A}(q) = 2^{\aleph_0}$.*

The proof is a technical construction showing how to build uncountably many trees in the constrained language of each state in Q_c. It is based on a result of Niwinski [17] that characterises uncountable ω-regular languages.

Combining Lemma 15 and Theorem 3 we obtain the following decidability result.

Theorem 7. *The cardinality problem is decidable for Büchi tree automata with equality and disequality constraints.*

6. Co-Büchi Automata

A natural question is whether Algorithm 1 works for other conditions than the Büchi condition, a first candidate here being the co-Büchi condition. We negatively answer this question, by exhibiting below a co-Büchi automaton with disequality constraints everywhere $\mathcal{A}$ such that $|L(\mathcal{A}_q)| = 2^{\aleph_0}$ for all states q of the automaton, while $L^{con}(\mathcal{A}_q) = \emptyset$. On such an automaton Algorithm 1 wrongly declares $\kappa_\mathcal{A}(q) = 2^{\aleph_0}$ for all states q.

More precisely, define $\mathcal{A} = (\{q_a, q_b\}, \{a, b\}, q_a, \Delta, \mathrm{Col})$ where $\mathrm{Col}(q_a) = 2$ and $\mathrm{Col}(q_b) = 1$, and Δ consists of those transitions $(q_x, (x, \neq), q_0, q_1)$ where $x \in \{a, b\}$ and q_0, q_1 are any states. On a given input tree there is at most one possible run, namely the one that assigns q_x to each node labelled by $(x, \neq)$.

The unconstrained language from state q_x is the set of all trees such that the root is labelled by x and such that any branch contains finitely many b's. It is easily seen that this set is uncountable (just think of the trees where there might be a b only at nodes in 0^*1: these trees are all accepted and there are uncountably many different ones).

Now, we claim that $L^{con}(\mathcal{A}_{q_x}) = \emptyset$ for $x \in \{a, b\}$. Indeed, by contradiction, if there was an accepted tree, it would contain at least one node labelled by b because the left and right subtrees of the root must be different. Let u_1 be such a node. Now, in the subtree rooted at u_1, for the same reason there is a node u_2 labelled by b. Hence, by repeating this process, we construct an infinite sequence $u_1 \sqsubset u_2 \sqsubset u_3 \cdots$ of nodes that are all labelled by b. Since all these nodes belong to the same infinite branch, the tree cannot be accepted, which leads to a contradiction.

This shows that we need to use different methods for solving the cardinality problem for co-Büchi automata (and parity automata in general). In this section we develop such methods for co-Büchi automata with disequality everywhere. Note that Theorem 3 does not apply to co-Büchi automata (removing equality constraints from co-Büchi automata might result in a more complex parity condition). However, the goal of this section is to introduce the main ideas

[1] The initial state can be an exception since it is always reachable. This can be dealt with and does not produce any problems but we do not want to obfuscate the presentation with such a minor detail.

in a simple setting. In the next section we briefly explain how to generalise the ideas to parity conditions, for which Theorem 3 applies.

From now on, let $\mathcal{A} = (Q, A \times \{\neq\}, q_{in}, \Delta, \mathrm{Col})$ be a co-Büchi automaton (*ie.* $\mathrm{Col} : Q \to \{1, 2\}$) with disequality everywhere. We say that an infinite tree t is **valid** if we have $t[u0] \neq t[u1]$ for every node u. In the sequel we may focus on valid trees only (as trees belonging to $L^{con}(\mathcal{A})$ are all valid).

6.1 Accepting Traces of $\mathcal{A}$

In the sequel we call a **trace** of $\mathcal{A}$ a pair $\rho = (t_\rho, r_\rho)$ where t_ρ is an A-labelled infinite *valid* tree and r_ρ is a run of $\mathcal{A}$ on $t_\rho^{\stackrel{?}{=}}$ starting from some arbitrary state. The trace is called accepting if the run satisfies the acceptance condition (we do not require the state at the root to be initial in this definition). Depending on the context, we should either think of a trace as a pair made of a tree and a run, or as an $(A \times Q)$-labelled tree.

We define two operations on sets of traces as follows.

- $\mathrm{Attr}(X) = \{(t_\rho, r_\rho)$ trace of $\mathcal{A} \mid \forall$ infinite branch $\pi, \exists u \sqsubset \pi$ s.t. $(t_\rho[u], r_\rho[u]) \in X\}$. Equivalently, by König's lemma it means that traces in $\mathrm{Attr}(X)$ are exactly those obtained by considering a finite prefix of an arbitrary (valid) trace and plug in every leaf a trace of X (starting from the same state as the one at the leaf).
- $\mathrm{Safety}(X) = \{(t_\rho, r_\rho)$ trace of $\mathcal{A} \mid \forall$ infinite branch π, either $\forall u \sqsubset \pi,\ \mathrm{Col}(r_\rho(u)) = 2$, or $\exists u \sqsubset \pi$ s.t. $(t_\rho[u], r_\rho[u]) \in X$ and $\mathrm{Col}(r_\rho(v)) = 2$ for all $v \sqsubset u\}$.

The following fact is immediate.

Fact 16. *For a set X of accepting traces*

1. $X \subseteq \mathrm{Attr}(X)$ *and* $X \subseteq \mathrm{Safety}(X)$*, and*
2. $\mathrm{Attr}(X)$ *and* $\mathrm{Safety}(X)$ *are sets of accepting traces.*

We now define an increasing transfinite sequence $(X_\alpha)_\alpha$ of accepting traces by letting $X_0 = \emptyset$, $X_{\alpha+1} = \mathrm{Attr}(\mathrm{Safety}(X_\alpha))$, and $X_\alpha = \bigcup_{\beta<\alpha} X_\beta$ for limit ordinals α. We call ATraces the limit of this sequence. The definition of the sequence $(X_\alpha)_\alpha$ corresponds to the nesting of greatest ($\mathrm{Safety}(\cdot)$) and least ($\mathrm{Attr}(\cdot)$) fixpoint in the semantics of a co-Büchi condition. Hence, the following lemma is not surprising.

Lemma 17. *The set* ATraces *is the set of accepting traces of $\mathcal{A}$.*

6.2 Computing Infinity Profiles

We now aim at computing the infinity profile $\mathbf{p}_\mathcal{A}$ of $\mathcal{A}$ (see Section 3 for the definition). We approximate it from below, and thus work with mappings $\mathbf{p}$ that associate with any state $q \in Q$ either a finite set of A-labelled valid *regular* trees or the value ∞ (recall that according to Lemma 4 the finite sets used in infinity profiles consist of regular trees). We refer to such a mapping $\mathbf{p}$ simply as a **profile**.

We aim at computing a sequence of profiles that converges to the infinity profile of $\mathcal{A}$. For that we compare profiles by letting $\mathbf{p_1} \leq \mathbf{p_2}$ if for every $q \in Q$ one has $\mathbf{p_1}(q) \preceq \mathbf{p_2}(q)$ where $x \preceq y$ means that either $y = \infty$ or both x and y are sets and $x \subseteq y$. A profile $\mathbf{p}$ is **$\mathcal{A}$-compatible** if one has $\mathbf{p} \leq \mathbf{p}_A$. Note that the profile that maps $\emptyset$ to every state is $\mathcal{A}$-compatible.

We now give a profile counterpart of the $X \mapsto \mathrm{Attr}(X)$ and $X \mapsto \mathrm{Safety}(X)$ operations. Let $\mathbf{p}$ be an $\mathcal{A}$-compatible profile. We define new profiles $\mathrm{Attr}(\mathbf{p})$ and $\mathrm{Safety}(\mathbf{p})$ as follows.

(i) For every state $q \in Q$ such that $\mathbf{p}(q)$ is a finite set of regular trees $\{t_1, \ldots, t_\ell\}$ we choose for every tree t_i an (arbitrary) accepting run r_i of $\mathcal{A}_q$ over $t_i^{\stackrel{?}{=}}$: this leads to a set X_q of accepting traces $\{(t_1, r_1), \ldots, (t_\ell, r_\ell)\}$.

(ii) For every state $q \in Q$ such that $\mathbf{p}(q) = \infty$ we choose an (arbitrary) infinite countable subset $\{t_i \mid i \geq 1\} \subseteq L^{con}(\mathcal{A}_q)$ and for every tree t_i we choose an (arbitrary) accepting run r_i of $\mathcal{A}_q$ over $t_i^{\stackrel{?}{=}}$: this leads to an infinite countable set X_q of accepting traces $\{(t_i, r_i) \mid i \geq 0\}$.

(iii) We let $X = \bigcup_{q \in Q} X_q$ and consider $\mathrm{Attr}(X)$. Then for every state $q \in Q$ we consider the set $T_q = \{t \mid \exists (t, r) \in \mathrm{Attr}(X)$ with $r(\varepsilon) = q\}$: if this set is finite (hence, consists only of regular trees, by Lemma 18 below) we let $\mathrm{Attr}(\mathbf{p})(q) = T_q$ and otherwise we let $\mathrm{Attr}(\mathbf{p}) = \infty$.

(iv) We let $X = \bigcup_{q \in Q} X_q$ and consider $\mathrm{Safety}(X)$. Then for every state $q \in Q$ we consider the set $T_q = \{t \mid \exists (t, r) \in \mathrm{Safety}(X)$ with $r(\varepsilon) = q\}$: if this set is finite (hence, consists only of regular trees, by Lemma 18 below) we let $\mathrm{Safety}(\mathbf{p})(q) = T_q$ and otherwise we let $\mathrm{Safety}(\mathbf{p}) = \infty$.

A simple analysis of this process yields the following result.

Lemma 18. *Let $\mathbf{p}$ be an $\mathcal{A}$-compatible profile. The following holds.*

(1) Neither $\mathrm{Attr}(\mathbf{p})$ *nor* $\mathrm{Safety}(\mathbf{p})$ *depends on the choice of the r_i* (resp. (t_i, r_i)) *at step (i)* (resp. *at step (ii)).*
(2) $\mathbf{p} \leq \mathrm{Attr}(\mathbf{p})$ *and* $\mathbf{p} \leq \mathrm{Safety}(\mathbf{p})$.
(3) If $\mathrm{Attr}(\mathbf{p})(q) \neq \infty$ *then it consists only of regular trees.*
(4) If $\mathrm{Safety}(\mathbf{p})(q) \neq \infty$ *then it consists only of regular trees.*

One key ingredient of the decision procedure for co-Büchi automata is given in the next two lemmas, which show that $\mathrm{Attr}(\mathbf{p})$ and $\mathrm{Safety}(\mathbf{p})$ can effectively been computed starting from $\mathbf{p}$.

Lemma 19. *Let $\mathbf{p}$ be an $\mathcal{A}$-compatible profile. Then one can compute* $\mathrm{Attr}(\mathbf{p})$.

Proof sketch. The main idea is to construct an extended Büchi automaton $\mathcal{B}$ that behaves like $\mathcal{A}$ at the beginning but has to do on every branch one of the following actions:

- When being in some state q such that $\mathbf{p}(q) \neq \infty$, check that the current subtree belongs to $\mathbf{p}(q)$.
- When being in some state q such that $\mathbf{p}(q) = \infty$ check that the current subtree belongs to a fixed countable ω-regular set of valid trees (using labels from an alphabet disjoint from A).

The final states are the ones used after stopping simulating $\mathcal{A}$.

The infinity profile of $\mathcal{B}$ is then shown to be equal to $\mathrm{Attr}(\mathbf{p})$ on the states of $\mathcal{A}$. Since $\mathcal{B}$ can be built so that it only requires a Büchi acceptance condition we can use the results from Section 5 and compute its infinity profile. □

The idea for the next lemma is the same as for the proof Lemma 19.

Lemma 20. *Let $\mathbf{p}$ be an $\mathcal{A}$-compatible profile. Then one can compute* $\mathrm{Safety}(\mathbf{p})$.

The following lemma says that $\mathbf{p}_\mathcal{A}$ is the smallest fixpoint of the operator $\mathbf{p} \mapsto \mathrm{Attr}(\mathrm{Safety}(\mathbf{p}))$, which is a direct consequence of the definition of the operator $\mathbf{p} \mapsto \mathrm{Attr}(\mathrm{Safety}(\mathbf{p}))$ and of the characterisation in Lemma 17.

Lemma 21. *Let $\mathbf{p}$ be a profile that is $\mathcal{A}$-compatible. If* $\mathbf{p} = \mathrm{Attr}(\mathrm{Safety}(\mathbf{p}))$ *then* $\mathbf{p} = \mathbf{p}_\mathcal{A}$.

Hence a natural idea from Lemma 19, Lemma 20 and Lemma 21 is to start from the profile that maps $\emptyset$ to every state and to iteratively apply the operator $\mathbf{p} \mapsto \mathrm{Attr}(\mathrm{Safety}(\mathbf{p}))$ until reaching a fixpoint. Unfortunately, it is not clear whether this sequence always converges in a finite number of steps. Hence, we need to speed-up the convergence.

6.3 Speeding-Up Convergence

We first need to introduce some basic definitions (partly borrowed from [17]).

Let $q \in Q$ be a state and let t be a (valid) tree. We say that t is ***q*-good** if there exists an accepting run r of $\mathcal{A}_q$ on $t^{\stackrel{?}{=}}$, a node $u \neq \varepsilon$ such that (i) $t[u] = t$, (ii) $r(u) = q$, and (iii) the smallest colour seen in r between the root and u (included) is even.

Lemma 22. *Given a regular tree t one can decide for any state q whether t is q-good.*

Proof. One first checks that t is valid (see the proof of Proposition 1). Then the problem boils down to a question on standard tree automata and can be solved with classical methods, for example using a two-player game on a finite graph with an ω-regular winning condition. □

Two distinct q-good trees can be combined to build infinitely many accepted trees, as stated in the following lemma.

Lemma 23. *Let q be a state. If there are two trees $t \neq t'$ that are both q-good then $L^{con}(\mathcal{A}_q)$ is uncountable.*

We now describe an algorithm — $\mathrm{InfinityCheck}(q_0, T)$ — that takes as input a state q_0 together with a finite set T of regular trees in $L^{con}(\mathcal{A}_{q_0})$ and checks whether the set T contains enough information to identify $L^{con}(\mathcal{A}_{q_0})$ as being infinite based on Lemma 23. In this case it outputs "infinite", and "do not know" otherwise. The algorithm works as follows:

(1) Define $X = \{t' \mid t' \text{ is a subtree of some } t \in T\}$.

(2) Define $Q_{reach} = \{q \mid \exists r \text{ accepting } t^{\stackrel{?}{=}} \text{ from } q_0 \text{ for some } t \in T \text{ and } \exists u \text{ with } r(u) = q\}$, *ie.* we consider those states that appear in accepting runs over trees in T.

(3) For all states $q \in Q_{reach}$, let $\mathrm{Good}(q) = \{t \in X \mid t \text{ is } q\text{-good}\}$. For all states $q \notin Q_{reach}$, let $\mathrm{Good}(q) = \emptyset$.

(4) If for some $q \in Q_{reach}$, $|\mathrm{Good}(q)| > 1$ then return "Infinite" and stop.

(5) Define the profile $\mathbf{p}_{\mathrm{Good}}$ by letting $\mathbf{p}_{\mathrm{Good}}(q) = \mathrm{Good}(q)$. Compute $\mathrm{Attr}(\mathbf{p}_{\mathrm{Good}})(q_0)$. If it is equal to ∞ then return "Infinite" otherwise return "Do not know".

Now let $\mathbf{p}$ be a profile. We define $\mathrm{SpeedUp}(\mathbf{p})$ by letting for any state q, $\mathrm{SpeedUp}(\mathbf{p})(q) = \infty$ if $\mathbf{p}(q) = \infty$ or if $\mathbf{p}(q) \neq \infty$ and the algorithm $\mathrm{InfinityCheck}(q_0, T)$ returns "Infinite" on $(q, \mathbf{p}(q))$; and $\mathrm{SpeedUp}(\mathbf{p})(q) = \mathbf{p}(q)$ otherwise (*ie.* if the algorithm $\mathrm{InfinityCheck}(q_0, T)$ returns "Do not know"). Obviously, $\mathbf{p} \leq \mathrm{SpeedUp}(\mathbf{p})$.

The next lemma shows that the previous sequence converges in finite time to infinity profile of $\mathcal{A}$.

Lemma 24. *Let $\mathbf{p}_0$ be the profile that maps $\emptyset$ to every state and let $(\mathbf{p}_i)_{i\geq 0}$ be defined by letting, for any $i \geq 0$, $\mathbf{p}_{i+1} = \mathrm{SpeedUp}(\mathrm{Attr}(\mathrm{Safety}(\mathbf{p}_i)))$. Then the sequence $(\mathbf{p}_i)_{i\geq 0}$ converges in a finite number of steps to $\mathbf{p}_{\mathcal{A}}$.*

Lemma 24 implies that we can compute the infinity profile of $\mathcal{A}$. In combination with Proposition 2 we obtain the following decidability result.

Theorem 8. *The cardinality problem is decidable for co-Büchi automata with disequality everywhere.*

7. Parity Automata

We now turn to the general case where $\mathcal{A}$ is a parity automaton with disequality everywhere using colours in $[0, n]$. Our main result is an algorithm to compute the infinity profile of such an automaton. Hence combined with Theorem 3 and Proposition 2 we obtain the following result.

Theorem 9. *The cardinality problem is decidable for parity tree automata with equality and disequality constraints.*

Proof sketch. Computing the infinity profile of $\mathcal{A}$ is quite involved and we only explain here the key ideas and relate to the Büchi and co-Büchi cases presented in Sections 5 and 6.

The algorithm proceeds by induction on the number of colours. More precisely, for $0 \leq k \leq n+1$, we consider runs using colours in $[k, n]$, the base case being the one where $k = n+1$ and the targeted one being the one where $k = 0$.

The algorithm is based on an inductive characterisation of accepted traces generalising the one we gave for the co-Büchi case. Here, it is no longer sufficient to work on traces: in order to manage a larger set of colours than for the co-Büchi setting, we need to consider *contexts* which can be viewed as traces with holes (possibly infinitely many).

Of special interest are those contexts $\mathrm{CParity}([k, n])$ that are accepting (meaning that every infinite branch is accepting) and only use colours in $[k, n]$ (but remember we can have infinitely many holes). It is important here to stress that we do not deal with constraints for the moment (as we did also in the co-Büchi case). A key ingredient is an inductive characterisation of $\mathrm{CParity}([k, n])$ using $\mathrm{CParity}([k+1, n])$ in terms of two operators on sets of context denoted $X \mapsto \mathrm{Iter}_*^{\{k\}}(X)$ and $X \mapsto \mathrm{CoBuchi}^{[k-1,n]}(X)$, which is inspired by the characterisation of winning regions from [24]. Unsurprisingly, the characterisation is dissymmetric depending on whether k is even or odd.

Next, we define *$\mathcal{A}$-pre-profiles* which are an analogue of profiles as defined in the co-Büchi case. These are mapping that approximate from below the infinity profile of $\mathcal{A}$. A difficulty here is that, in order to deal with the induction, we need more information than the sole pre-profile. For this reason we have an operation $\mathrm{Cnt}(k, \mathbf{p})$ that builds a (compact but possibly infinite representation of a) set of *valid accepting* contexts (a context is valid if every infinite subtree in it is valid). In a nutshell, a context belongs to $\mathrm{Cnt}(k, \mathbf{p})$ if it is valid and can be obtained by completing an accepting context (that only uses colours $\geq k$) using the informations from the pre-profile $\mathbf{p}$. An important fact is that $L^{con}(\mathcal{A}) = \mathrm{Cnt}(0, \mathbf{p}_\emptyset)$ where $\mathbf{p}_\emptyset$ is the pre-profile that maps $\emptyset$ to any state.

To compute the profile of $L^{con}(\mathcal{A}) = \mathrm{Cnt}(0, \mathbf{p}_\emptyset)$ we need two ingredients. First we derive from the inductive characterisation of $\mathrm{CParity}([k, n])$ an inductive characterisation of $\mathrm{Cnt}(k, \mathbf{p})$ depending on $\mathrm{Cnt}(k+1, \mathbf{p})$ and using two operators $X \mapsto \mathrm{Iter}_*^R(X)$ and $X \mapsto \mathrm{CoBuchi}^R(X)$, which are a slight modification of the previously mentioned ones that were putting constraints on a set of colours C to be used while here the constraints is on a set of states R to be used. The second (easy) ingredient is a notion of profile $\mathrm{Prof}_{\mathbf{p}}(\mathrm{X})$ for a set X of contexts knowing a pre-profile $\mathbf{p}$ (used to derive informations at holes). As this notion coincides with the one on trees when considering context without holes, our original problem is reduced to compute $\mathrm{Prof}_{\mathbf{p}_\emptyset}(\mathrm{Cnt}(0, \mathbf{p}_\emptyset))$ and this latter task is made possible thanks to the previous inductive characterisation of $\mathrm{Cnt}(k, \mathbf{p})$.

Therefore we define an algorithm $\mathrm{Profiler}(k, \mathbf{p})$ that takes as input an integer $k \in [0, n+1]$ and a pre-profile $\mathbf{p}$ and outputs $\mathrm{Prof}_{\mathbf{p}}(\mathrm{Cnt(k, \mathbf{p})})$. The algorithm works by case inspection. The base case is when $k > n$. For the case where k is odd it follows the same approach as for the co-Büchi case, constructing an increasing sequence of under-approximations that is ensured to converge thanks to (a slight variant of) the previous $\mathrm{SpeedUp}$ operator. For the case where k is even we have to deal with the $X \mapsto \mathrm{Iter}_*^R(X)$ operator and the first step is to prove that it suffices to apply this

operator to a subset of contexts which we prove to be regular.Then we successively apply this operator as long as we discover countable (regular) set of contexts: for each of them we can compute their cardinality in the constrained version using the same technique as in Section 4. Finally, when no more countable sets are found we prove that for the remaining states their constrained version has cardinality $2^{\aleph_0}$: this is done by generalising the technique of Lemma 15 used in the Büchi case. □

8. Discussion

In this paper we have started an investigation of automata with equality and disequality constraints on infinite trees, by analysing the model with constraints between siblings. Besides showing that the resulting language class forms a Boolean algebra, our main contribution is a solution for the cardinality problem (and hence for the emptiness and finiteness problems). While this shows that this extension of classical parity tree automata preserves many of the good properties, we do not have a result concerning the projection of languages accepted by automata with constraints between siblings. For establishing connections to logic, projection is an essential operation for dealing with quantifiers. Nevertheless, our results can be used to solve some decision problems for MSO over infinite trees. First of all, given an MSO formula φ and a parity tree automaton with constraints between siblings $\mathcal{A}$, it is decidable whether φ has a model accepted by $\mathcal{A}$ (transform φ into an equivalent parity automaton, take the intersection with $\mathcal{A}$, and test the resulting automaton for emptiness). Furthermore, it is also possible to solve satisfiability for extended MSO queries, for example, Boolean combinations of queries of the following form, where $\varphi, \varphi_0, \varphi_1$ are standard MSO formulas:

$$\forall x.(t[x] \models \varphi) \implies (t[x0] \neq t[x1] \wedge t[x0] \models \varphi_0 \wedge t[x1] \models \varphi_1)$$

A tree t satisfies this query if for each node x at which the subtree $t[x]$ satisfies φ, the two subtrees of x are different and satisfy φ_0 and φ_1, respectively. The satisfiability of such a property can be checked by building a parity automaton with sibling constraints that guesses a node at which the property fails, and then complementing this automaton and testing it for emptiness.

In future work we plan to consider further decision problems for this automaton class, for example the regularity problem, asking for a given automaton with constraints whether the accepted language is regular (for the case of finite trees this is known to be decidable [3]). Furthermore, we want to investigate models with global constraints. In particular, we are interested in finding models with decidable emptiness problem that capture extensions of MSO with isomorphism tests.

References

[1] L. Barguñó, C. Creus, G. Godoy, F. Jacquemard, and C. Vacher. The emptiness problem for tree automata with global constraints. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010*, pages 263–272. IEEE Computer Society, 2010.

[2] B. Bogaert and S. Tison. Equality and disequality constraints on direct subterms in tree automata. In *Proceedings of the 19th Symposium on Theoretical Aspects of Computer Science (STACS 2002)*, volume 577 of *Lecture Notes in Computer Science*, pages 161–171. Springer-Verlag, 1992.

[3] B. Bogaert, F. Seynhaeve, and S. Tison. The recognizability problem for tree automata with comparisons between brothers. In *FoSSaCS'99, Proceedings*, volume 1578 of *LNCS*. Springer, 1999.

[4] M. Bojańczyk. A bounding quantifier. In *Proceedings of Computer Science Logic, 18th International Workshop, CSL 2004, 13th Annual Conference of the EACSL*, volume 3210 of *LNCS*, pages 41–55. Springer, 2004.

[5] M. Bojańczyk. Weak MSO with the unbounding quantifier. *Theory of Computing Systems*, 48(3):554–576, 2011. ISSN 1432-4350.

[6] M. Bojanczyk. Weak MSO+U with path quantifiers over infinite trees. In *Proceedings of Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014*, volume 8573 of *LNCS*, pages 38–49. Springer Berlin Heidelberg, 2014.

[7] L. Cardelli and G. Ghelli. TQL: a query language for semistructured data based on the ambient logic. *Mathematical Structures in Computer Science*, 14(3):285–327, 2004.

[8] T. Colcombet and C. Löding. The non-deterministic mostowski hierarchy and distance-parity automata. In *Proceedings of the 35th International Colloquium on Automata, Languages, and Programming (ICALP 2008)*, volume 5126 of *Lecture Notes in Computer Science*, pages 398–409. Springer-Verlag, 2008.

[9] T. Colcombet, D. Kuperberg, C. Löding, and M. Vanden Boom. Deciding the weak definability of büchi definable tree languages. In *"Proceedings of Computer Science Logic, 27th Annual Conference of the EACSL (CSL 2013)"*, volume 23 of *LIPIcs*, pages 215–230. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013.

[10] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. `http://tata.gforge.inria.fr/`, 2007. release October, 12th 2007.

[11] E. Filiot, J. Talbot, and S. 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 *LNCS*, pages 130–145. Springer, 2007.

[12] E. Filiot, J. Talbot, and S. Tison. Tree automata with global constraints. In *Proceedings of Developments in Language Theory, 12th International Conference, DLT 2008*, volume 5257 of *LNCS*, pages 314–326. Springer, 2008.

[13] E. Filiot, J. Talbot, and S. Tison. Tree automata with global constraints. *Int. J. Found. Comput. Sci.*, 21(4):571–596, 2010.

[14] G. Godoy and O. Giménez. The HOM problem is decidable. *Journal of the Association for Computing Machinery (ACM)*, 60(4):23, 2013.

[15] G. Godoy, O. Giménez, L. Ramos, and C. Àlvarez. The HOM problem is decidable. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010*, pages 485–494. ACM, 2010.

[16] D. E. Muller and P. E. Schupp. Simulating alternating tree automata by nondeterministic automata. *Theoretical Computer Science*, 141(1&2): 69–107, 1995.

[17] D. Niwinski. On the cardinality of sets of infinite trees recognizable by finite automata. In *Proceedings of the 16th Symposium, Mathematical Foundations of Computer Science (MFCS 1991)*, volume 520 of *Lecture Notes in Computer Science*, pages 367–376. Springer-Verlag, 1991.

[18] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987.

[19] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the Symposium on Principles of Programming Languages, POPL'89*, pages 179–190, 1989.

[20] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141: 1–35, 1969.

[21] M. O. Rabin. *Automata on infinite objects and Church's problem*. American Mathematical Society, Providence, R.I., 1972. Conference Board of the Mathematical Sciences Regional Conference Series in Mathematics, No. 13.

[22] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Language Theory*, volume III, pages 389–455. Springer-Verlag, 1997.

[23] M. Vanden Boom. Weak cost monadic logic over infinite trees. In *Proceedings of Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011*, pages 580–591, 2011.

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