The Inclusion Problem for Weighted Automata on Infinite Trees

Stefan Borgwardt and Rafael Peñaloza
Theoretical Computer Science, TU Dresden, Germany
{penaloza,stefborg}@tcs.inf.tu-dresden.de

Abstract

Weighted automata can be seen as a natural generalization of finite state automata to more complex algebraic structures. The standard reasoning tasks for unweighted automata can also be generalized to the weighted setting. In this paper we study the problems of intersection, complementation and inclusion for weighted automata on infinite trees and show that they are not harder complexity-wise than reasoning with unweighted automata. We also present explicit methods for solving these problems optimally.

1 Introduction

One of the current areas of interest in the field of automata theory is the study of weighted automata. These automata can be seen as a generalization of finite state automata in which the input structures are not accepted or rejected, but rather given a value called their *weight*. More formally, a weighted automaton defines a formal power series over a suitable algebraic structure [22, 11].

The natural question to ask in the presence of such a generalization is whether the properties of the special case still hold. We can find several instances in the literature where this question is answered affirmatively. For example, the relationship between automata and MSO logic, originally shown by Büchi [7], has been proven to hold also for weighted automata over finite and infinite words and trees [9, 12, 13, 21] and some weighted MSO logics. In the area of Model Checking, where Büchi automata are used to model properties of transition systems, weighted Büchi automata have recently been considered for multi-valued model checking [6].

For this purpose, standard tasks like deciding emptiness or complementing automata over finite or infinite words have been generalized to the weighted setting, and algorithms solving these generalized problems have

been developed. One interesting result obtained is that often the complexity of these generalized tasks is not higher than in the unweighted case. For instance, the so-called *emptiness value problem* of weighted automata on infinite words is NLogSpace-complete [15].

Despite reasoning with weighted automata on infinite *words* being well studied, there is a significant lack of results for weighted automata over infinite *trees*. In fact, to the best of our knowledge, the only explicit reasoning algorithm for these automata was given in [4], where a polynomial-time algorithm for computing the emptiness value of automata on infinite *unlabeled* trees, if the weights belong to a distributive lattice, is described. For labeled trees, a method that reduces the problem to several (unweighted) emptiness tests was described in [10]. The execution time of this approach, however, depends on the structure of the lattice.

In this paper we cover this gap by looking at reasoning problems for weighted automata on infinite trees that arise from generalizing standard problems for unweighted tree automata. We show that weighted union, intersection and emptiness of tree automata are computable in polynomial time, independently of the lattice used. We also look at the inclusion and complementation problems, and we show that their complexity remains equal to the unweighted case.

As for automata on infinite words, there are different kinds of automata on infinite trees mainly depending on the acceptance condition used (e.g., Büchi or co-Büchi automata; see Section 2.2). Since some of these classes are not closed under complementation, we analyze several different types of automata with their closure properties relative to each other. Most of these relationships are well-known for unweighted automata, but had not been analyzed for weighted automata. We also present explicit constructions for the complement of some classes of weighted and unweighted tree automata.

Due to a lack of space, some proofs have been left out of this paper. They can be found in the technical report [5].

2 Automata on Infinite Trees

The object of our study are automata on infinite trees with weights from a lattice [14]. We briefly introduce lattices before defining the automata.

2.1 Lattices

A *lattice* is a partially ordered set $(S, \leq)$ where infima and suprema of arbitrary finite subsets of S always exist. The lattice $(S, \leq)$ is *finite* if its carrier set S is finite, it is *distributive* if the infimum and supremum operators distribute over each other, and it is *bounded* if it has a smallest

element 0_S and a greatest element 1_S. In the following, we will often use the carrier set S to denote the lattice $(S, \leq)$. The infimum (supremum) of a finite subset $T \subseteq S$ will be denoted by $\bigotimes_{a \in T} a$ ($\bigoplus_{a \in T} a$). We will also use the infix notation if the set contains only two elements.

A *De Morgan lattice* S is a bounded distributive lattice with a *negation function* $^- : S \to S$ that satisfies the property $\overline{\overline{a}} = a$ for every $a \in S$ and the De Morgan law, $\overline{a \otimes b} = \overline{a} \oplus \overline{b}$ for every $a, b \in S$. As a consequence, the following hold: $\overline{a \oplus b} = \overline{a} \otimes \overline{b}$, $a \leq b$ iff $\overline{b} \leq \overline{a}$, $\overline{0_S} = 1_S$, and $\overline{1_S} = 0_S$.

A *Boolean lattice* is a De Morgan lattice where $a \otimes \overline{a} = 0_S$ (or, equivalently $a \oplus \overline{a} = 1_S$) holds for every $a \in S$; $\overline{a}$ is then called the *complement* of a. Every finite Boolean lattice is isomorphic to a powerset lattice $(\mathcal{P}(X), \subseteq)$.

An element p of a lattice S is called *meet prime* if for every $a, b \in S$, $a \otimes b \leq p$ implies that either $a \leq p$ or $b \leq p$. The dual notion is that of a *join prime* element. Every element of a distributive lattice S is the infimum of all meet prime elements above it. In a De Morgan lattice S, the negation $\overline{a}$ of a meet prime element $a \in S$ is join prime and vice versa. If the finite Boolean lattice S is isomorphic to the powerset lattice over some set X, then there are exactly $|X|$ meet prime and $|X|$ join prime elements in S.

2.2 Weighted Automata

We consider automata that receive as input infinite trees of a fixed arity k. For a positive integer k, we define $K := \{1, \ldots, k\}$. We focus on the *full k-ary tree* K^*, whose root is denoted by the empty word ε, and the i-th successor of the node u is identified by ui.[1] A *path* p is a prefix-closed set of nodes such that if $u \in p$, then there is at most one $i \in K$ with $ui \in p$. $\mathcal{P}ath(K^*)$ denotes the set of all infinite paths of K^*. A *labeled tree* is a mapping $t : K^* \to \Sigma$ for some labeling alphabet Σ. As usual, the set of all such mappings is denoted by Σ^{K^*}.

For an alphabet Σ and a lattice S, a *formal tree series over Σ and S* is a mapping $\Sigma^{K^*} \to S$; i.e., a function that assigns each labeled tree a weight from S. For a formal tree series $f : \Sigma^{K^*} \to S$, the expression (f, t), called the *coefficient of f at t*, denotes the image of a tree t under f.

Definition 1. A *weighted generalized Büchi tree automaton (WGBA)* is a tuple $\mathcal{A} = (Q, \Sigma, S, \mathrm{in}, \mathrm{wt}, F_1, \ldots, F_n)$ where Q is a finite set of *states*, Σ is the *input alphabet*, S is a distributive lattice, $\mathrm{in} : Q \to S$ is the *initial distribution*, $\mathrm{wt} : Q \times \Sigma \times Q^k \to S$ is the *transition weight function* and $F_1, \ldots, F_n \subseteq Q$ are the sets of *final states*. A WGBA is called a *weighted*

[1] Infinite trees built over a ranked alphabet can always be embedded in a full k-ary tree as long as the arity of the symbols is bounded by k.

Büchi tree automaton (WBA) if $n = 1$ and *weighted looping tree automaton (WLA)* if $n = 0$.

A *run* of the WGBA $\mathcal{A}$ is a labeled tree $r \in Q^{K^*}$. This run is called *successful* if for every path $p \in \mathcal{P}ath(K^*)$ and every $i, 1 \leq i \leq n$ there are infinitely many nodes $u \in p$ such that $r(u) \in F_i$. The set of all successful runs of $\mathcal{A}$ is denoted by $\mathrm{succ}(\mathcal{A})$. We define the *transition* of r on $t \in \Sigma^{K^*}$ at a node $u \in K^*$ as $\overrightarrow{r(t,u)} := (r(u), t(u), r(u1), \ldots, r(uk))$. The *weight* of r on t is the value

$$\mathrm{wt}(t,r) := \mathrm{in}(r(\varepsilon)) \otimes \bigotimes_{u \in K^*} \mathrm{wt}(\overrightarrow{r(t,u)}) \ .$$

The *behavior* of $\mathcal{A}$ on an input tree $t \in \Sigma^{K^*}$ is

$$(\|\mathcal{A}\|, t) := \bigoplus_{r \in \mathrm{succ}(\mathcal{A})} \mathrm{wt}(t,r) \ .$$

Since the images of in and wt are finite, the infima and suprema above are restricted to a finitely generated (and thus finite) distributive sublattice. Thus, even if S is an infinite lattice, the formal tree series $\|A\|$ has a finite image. In consequence, we can always assume w.l.o.g. that S is finite.

We will additionally consider *weighted co-Büchi tree automata (WCA)*. A WCA is like a WBA (i.e., $n = 1$), except that a run is successful if every infinite path contains only finitely many states from $Q \setminus F_1$. Notice that WLA can be seen as special cases of both WBA and WCA in which $F_1 = Q$ and hence every run is successful.

A more expressive acceptance condition is used in *weighted parity tree automata (WPA)*. A *priority function* $\pi : Q \to \mathbb{N}$ assigns natural numbers to the states. A run is accepted if every infinite path fulfills the condition that the minimal priority that occurs infinitely often along this path must be even. Both WBA and WCA can be expressed as WPA with only two priorites. Other acceptance conditions as expressive as the parity condition exist, e.g., the Rabin, Streett, or Muller conditions [25]. However, translating those conditions into each other may involve an exponential blowup [8].

Standard (unweighted) tree automata can be seen as weighted tree automata over the lattice $\mathbb{B} := (\{0,1\}, \leq)$, with $0 < 1$. The supremum and infimum in this lattice are denoted as $\vee$ and $\wedge$, respectively. The behavior of such automata is the characteristic function of the set $\mathcal{L}(\mathcal{A}) := \{t \in \Sigma^{K^*} \mid (\|\mathcal{A}\|, t) = 1\}$ of all trees *accepted* by $\mathcal{A}$. Likewise, the functions in and wt can be seen as a set $I \subseteq Q$ and a relation $\Delta \subseteq Q \times \Sigma \times Q^k$, respectively. The

abbreviations PA, GBA, BA, CA, and LA will be used when the underlying lattice of the automaton is $\mathbb{B}$.

In the following we will use expressions of the form WXA or XA, where X is a placeholder for the different acceptance conditions; i.e., WXA stands for an arbitrary weighted tree automaton.

For the complexity results in this paper, we consider the *size* of an unweighted tree automaton to be the number $|Q|$ of its states and the number $|\mathrm{im}(\pi)|$ of priorities it uses. For Büchi, co-Büchi, and looping acceptance conditions, the number of priorities is constant. The size of a weighted tree automaton is $|Q| \log |S| + |\mathrm{im}(\pi)|$, where $\log |S|$ is the space it takes to store an element of the underlying lattice S. Often, the additional factor $\log |S|$ does not affect the overall complexity and can be dropped.

Since weighted tree automata generalize unweighted tree automata, a natural question is whether the standard results and constructions available for the latter can be adapted to the former. For unweighted automata, one is often interested in computing the union and intersection of the languages accepted by two automata. These operations correspond to a supremum and an infimum computation, respectively, in the weighed setting. As the following lemma shows, these problems can be solved in polynomial time.

Lemma 2. *Let $\mathcal{A}, \mathcal{B}$ be WXA with $X \in \{L, B, C\}$. Then one can construct WXA $\mathcal{C}$ and $\mathcal{C}'$ of size polynomial in the sizes of $\mathcal{A}$ and $\mathcal{B}$ with $(\|\mathcal{C}\|, t) = (\|\mathcal{A}\|, t) \otimes (\|\mathcal{B}\|, t)$ and $(\|\mathcal{C}'\|, t) = (\|\mathcal{A}\|, t) \oplus (\|\mathcal{B}\|, t)$ for all $t \in \Sigma^{K^*}$.* □

Another important problem for unweighted automata is deciding emptiness of the accepted language; i.e., whether there is at least one tree that is accepted. The natural generalization of this problem is to compute the supremum of the behavior of $\mathcal{A}$ over all possible input trees. Using the ideas developed in [4], it is possible to show that this problem can be solved in polynomial time for WGBA (and hence also for WBA and WLA).

Lemma 3. *Given a WGBA $\mathcal{A}$, the value $\bigoplus_{t \in \Sigma^{K^*}} (\|\mathcal{A}\|, t)$ is computable in time polynomial in the size of $\mathcal{A}$.* □

Before looking at the problem of deciding inclusion of the languages accepted by two tree automata and its generalization to the weighted case, we will motivate our interest in this problem, by showing how it can be used for reasoning in description logics.

3 Motivation

In addition to the theoretical interest in deciding the inclusion of automata, we are motivated by the fact that tree automata can be used for reasoning

in Description Logics (DLs) [1]. DLs are decidable fragments of first-order logic that have been successfully employed for knowledge representation.

Consider for example the DL $\mathcal{ALC}$. Formulae of $\mathcal{ALC}$, called *concept descriptions*, are built using the syntactic rule $C ::= A \mid C_1 \sqcap C_2 \mid \neg C_1 \mid \forall r.C_1$, where A is an element of a fixed set N_C of *concept names*, r is an element of a fixed set N_R of *role names*, and C_1, C_2 are concept descriptions.

The semantics of these formulae is defined using interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consisting of a domain $\Delta^{\mathcal{I}}$ and a function assigning a subset of $\Delta^{\mathcal{I}}$ to every concept name and binary relations over $\Delta^{\mathcal{I}}$ to every role name. An interpretation can be seen as a labeled directed graph with nodes $\Delta^{\mathcal{I}}$ and edges labeled by role names and nodes labeled by sets of concept names. Complex concept descriptions are interpreted as $(C_1 \sqcap C_2)^{\mathcal{I}} := C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$; $(\neg C_1)^{\mathcal{I}} := \Delta^{\mathcal{I}} \backslash C_1^{\mathcal{I}}$; $(\forall r.C_1)^{\mathcal{I}} := \{d \in \Delta^{\mathcal{I}} \mid \forall e \in \Delta^{\mathcal{I}} : (d, e) \in r^{\mathcal{I}} \rightarrow e \in C_1^{\mathcal{I}}\}$.

A *terminological axiom* is of the form $C \sqsubseteq D$ for two concept descriptions C, D. This axiom is *satisfied* by an interpretation $\mathcal{I}$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds. In this case, $\mathcal{I}$ is called a *model* of the axiom. A concept description C is *satisfiable* w.r.t. a given set $\mathcal{T}$ of axioms if there is a model of all axioms in $\mathcal{T}$ such that $C^{\mathcal{I}} \neq \emptyset$.

$\mathcal{ALC}$ has the *tree model property*, i.e., every satisfiable concept description also has a (possibly infinite) tree-shaped model. This gives rise to a characterization of satisfiability in $\mathcal{ALC}$ using the emptiness problem for automata on infinite trees. One can construct a looping tree automaton $\mathcal{A}_C$ that accepts all tree models of a given concept description C [2].

Weighted tree automata over lattices have recently been used for more specific reasoning tasks in $\mathcal{ALC}$. If one has a large set $\mathcal{T}$ of terminological axioms, it often does not suffice to know that a given concept C is unsatisfiable w.r.t. these axioms; to correct errors in the axioms, one may also want to know the specific axioms that are responsible for the unsatisfiability of C. This can be achieved by determining a so-called *pinpointing formula* which can be computed by a lattice-weighted looping tree automaton [4].

Other applications include the presence of access restrictions on the terminological axioms, where one wants to know whether a certain user has access to some knowledge based on their access to the axioms [3], or fuzzy DLs, where concept descriptions are interpreted as fuzzy sets and one wants to find the degree to which a concept description is satisfiable [27, 24].

All these applications motivate us to look at the properties of unweighted and weighted tree automata. In particular, deciding inclusion of tree automata can be used to decide whether a terminological axiom $C \sqsubseteq D$ is a consequence of a set $\mathcal{T}$ of axioms, i.e., $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds in all models of $\mathcal{T}$.

One can construct tree automata $\mathcal{A}_C$ and $\mathcal{A}_D$ such that $\mathcal{L}(\mathcal{A}_C) \subseteq \mathcal{L}(\mathcal{A}_D)$ iff $C \sqsubseteq D$ is a consequence of $\mathcal{T}$. This task can be reduced to checking satisfiability of the concept $C \sqcap \neg D$ w.r.t. $\mathcal{T}$, thus eliminating the need for checking the inclusion. However, in DLs without negation this reduction is not possible and the inclusion test for tree automata has to be applied.

There are also other cases in which it may be necessary to compute such consequences via the inclusion check between tree automata. Consider for example the DL $\mathcal{ALC}(\neg)$, which additionally allows for role expressions of the form $\neg r$ which are interpreted as $(\neg r)^{\mathcal{I}} := (\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}) \setminus r^{\mathcal{I}}$. This DL does not have the tree model property, as, e.g., $A \sqcap \forall \neg r.\neg A$ is satisfiable, but has no tree model. Any domain element belonging to this concept would need to have itself as an r-successor, which forms a loop in the model.

Reasoning in $\mathcal{ALC}(\neg)$ is more complex than in $\mathcal{ALC}$. One may, however, look at the more restricted reasoning task of deciding whether $C \sqsubseteq \forall \neg r.D$ holds for some $\mathcal{ALC}$-concept descriptions C, D w.r.t. to a set of $\mathcal{ALC}$-terminogical axioms $\mathcal{T}$. This is a reasoning task in $\mathcal{ALC}$ and thus can be decided over tree models. It cannot be decided using reasoning methods for $\mathcal{ALC}(\neg)$ since these would have to consider all models of $\mathcal{T}$ and not just the tree models. However, one can easily construct a Büchi automaton checking whether $\forall \neg r.D$ holds, i.e., whether all individuals not connected by an r-edge satisfy D. Thus, this restricted reasoning task can be solved using the inclusion test between two tree automata.

In the following sections, we investigate the complexity of inclusion for tree automata and generalize this problem to weighted tree automata.

4 Deciding Inclusion

We are interested in the inclusion problem of two tree automata which can use different acceptance conditions. This problem is defined as follows.

Problem (Inclusion $\mathcal{I}_{\mathrm{X,Y}}$). Given an XA $\mathcal{A}$ and a YA $\mathcal{A}'$, decide whether $\mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A})$.

One approach to solving this problem is to construct a tree automaton that accepts the complement of $\mathcal{L}(\mathcal{A})$, since the inclusion $\mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A})$ holds iff $\mathcal{L}(\mathcal{A}') \cap \overline{\mathcal{L}(\mathcal{A})} = \emptyset$. If one is able to efficiently decide the emptiness of this intersection, then the inclusion problem can be easily solved. Thus, we look also at the complementation problem.

Problem (Complementation $\mathcal{C}_{\mathrm{X,Y}}$). Given an XA $\mathcal{A}$, construct a YA $\overline{\mathcal{A}}$ with $\mathcal{L}(\overline{\mathcal{A}}) = \overline{\mathcal{L}(\mathcal{A})}$.

Notice that we do not require that the complement automaton has the same acceptance condition as the original one. This is motivated by LA,

BA and CA not being closed under complement [25, 18], but, e.g., the complement of an LA can be recognized by a BA (see Theorem 6).

Despite the difference in expressivity, the inclusion problems $\mathcal{I}_{\mathrm{X,Y}}$ have the same complexity for all $X, Y \in \{L, B, C, P\}$; they are all ExpTime-complete. This follows from known results [23, 26, 8, 16].

Theorem 4. *$\mathcal{I}_{\mathrm{X,Y}}$ is ExpTime-complete for $X, Y \in \{L, B, C, P\}$.* □

We present the above result in more detail for a special case. The reason for this is that we want to adapt these constructions for solving the weighted version of the inclusion problem. It is thus useful to have an explicit description of an efficient complementation construction.

The following construction describes the complementation step for a looping tree automaton $\mathcal{A}$. The resulting automaton $\overline{\mathcal{A}}$ uses a Büchi condition since the construction involves complementing the acceptance condition (see [20] for details)[2] and has an exponential number of states.

Definition 5. Let $\mathcal{A} = (Q, \Sigma, I, \Delta)$ be an LA. The *complement automaton* of $\mathcal{A}$ is the BA $\overline{\mathcal{A}} := (Q_c, \Sigma, I_c, \Delta_c, F_c)$ where

- $Q_c := \mathcal{P}(Q)$,
- $I_c := \{I\}$,
- $(Q_0, \alpha, Q_1, \ldots, Q_k) \in \Delta_c$ iff for all $q_0 \in Q_0$ and $(q_0, \alpha, q_1, \ldots, q_k) \in \Delta$ there is an index $i \in K$ such that $q_i \in Q_i$,
- $F_c := \{\emptyset\}$.

Correctness of this construction follows from the more general results in [26]. However, we will present a direct proof here, which will later be used to show the correctness of the weighted complementation construction by lifting this proof to the more general lattice operations.

Theorem 6. *If $\mathcal{A}$ is an LA and $\overline{\mathcal{A}}$ its complement automaton (from Definition 5), then $\mathcal{L}(\overline{\mathcal{A}}) = \overline{\mathcal{L}(\mathcal{A})}$.*

Proof Sketch. To show $\mathcal{L}(\overline{\mathcal{A}}) \subseteq \overline{\mathcal{L}(\mathcal{A})}$, we assume that there is a tree $t \in \Sigma^{K^*}$ that is accepted by both $\overline{\mathcal{A}}$ and $\mathcal{A}$. Then there are successful runs $r \in Q^{K^*}$ of $\mathcal{A}$ and $\overline{r} \in Q_c^{K^*}$ of $\overline{\mathcal{A}}$. Using the definition of Δ_c, one can show that there is an infinite path p in the tree such that $r(u) \in \overline{r}(u)$ holds for all $u \in p$. This means that $\overline{r}(u) \neq \emptyset$ for all $u \in p$, which contradicts the success of $\overline{r}$.

For the other inclusion, we take a tree $t \notin \mathcal{L}(\mathcal{A})$. There must exist a finite prefix $T \subseteq K^*$ on which every run r starting in a state $q_0 \in I$ must

[2] Actually, a reachability or *weak Büchi* condition would suffice.

already contain an invalid transition, i.e., $\overrightarrow{r(t,u)} \notin \Delta$ for some $u \in T$. This can be shown by contradiction: If no such finite prefix existed, we could construct an infinite run of $\mathcal{A}$ on t, which would contradict $t \notin \mathcal{L}(\mathcal{A})$.

The next step is to define a run $\overline{r} \in Q_c^{K^*}$ of $\overline{\mathcal{A}}$ that has only $\emptyset$-labels outside of T and conforms to the transition relation Δ_c. Moreover, this run will satisfy the following property P on all nodes $u \in T$: if $q \in \overline{r}(u)$, then every run r of A with $r(u) = q$ has an invalid transition inside T. More precisely, $\overline{r}$ is defined inductively as follows:

- $\overline{r}(\varepsilon) := \{I\}$. $P(\varepsilon)$ holds because of how we chose the tree T.
- If $\overline{r}(u)$ is already defined and u is not a leaf of T, we consider all states $q \in \overline{r}(u)$ and all possible transitions $(q, t(u), q_1, \ldots, q_k) \in \Delta$. For each transition, we choose a direction $i \in K$ such that every subrun starting in q_i at node ui will contain an invalid transition in T. Such i must exist since otherwise there would be a run starting in q at the node u that had no invalid transition inside T, contradicting $P(u)$.

It is obvious that $\overline{r}$ is successful. All its transitions are valid transitions in Δ_c, because of the construction of $\overline{r}$ and the fact that $P(u)$ holds at the leaves of T. Thus, $t \in \mathcal{L}(\overline{\mathcal{A}})$. □

A similar construction is possible for complementing a CA into a BA. In this case we can adapt the construction for simulating alternating Büchi *word* automata by nondeterministic ones from [19].

Theorem 7. *For every CA $\mathcal{A}$ we can construct a BA $\overline{\mathcal{A}}$ of size exponential in the size of $\mathcal{A}$ with $\mathcal{L}(\overline{\mathcal{A}}) = \overline{\mathcal{L}(\mathcal{A})}$.* □

This provides a direct proof for $\mathcal{I}_{\mathrm{C,B}}$ being in ExpTime. Unfortunately, a construction for $\mathcal{C}_{\mathrm{B,C}}$ is not possible: a tree language recognizable by a BA, whose complement is not recognizable by any CA was shown in [17].

5 The Weighted Inclusion Problem

As mentioned already, unweighted tree automata are weighted tree automata over the Boolean lattice $\mathbb{B}$, whose operators correspond to the logical connectives. De Morgan lattices can be seen as a generalization of Boolean logic, where conjunction, disjunction and negation are translated to infimum $\otimes$, supremum $\oplus$, and complementation $^-$, respectively. We can use this fact to describe generalizations of the decision problems for unweighted tree automata to the weighted setting.

From a low-level point of view, the inclusion problem consists of deciding whether the implication $t \in \mathcal{L}(\mathcal{A}') \Rightarrow t \in \mathcal{L}(\mathcal{A})$ holds for every input tree t. Equivalently, we can express this property using the formula:

$$\bigwedge_{t \in \Sigma^{K^*}} \neg(\|\mathcal{A}'\|, t) \vee (\|\mathcal{A}\|, t) \ ,$$

which can then be generalized to arbitrary De Morgan lattices as follows.

Problem (Weighted Inclusion $\mathcal{I}_{\mathrm{WX,WY}}$). Given a WXA $\mathcal{A}$ and a WYA $\mathcal{A}'$ over the same De Morgan lattice, compute $\bigotimes_{t \in \Sigma^{K^*}} \overline{(\|\mathcal{A}'\|, t)} \oplus (\|\mathcal{A}\|, t)$.

Remark. A more intuitive generalization of the inclusion problem is to decide whether $(\|\mathcal{A}'\|, t) \leq (\|\mathcal{A}\|, t)$ holds for all input trees t. For Boolean lattices, this is only a special case of the above problem, since

$$(\|\mathcal{A}'\|, t) \leq (\|\mathcal{A}\|, t) \Leftrightarrow \overline{(\|\mathcal{A}'\|, t)} \oplus (\|\mathcal{A}\|, t) = 1_S \ .$$

As in the unweighted case, the problem $\mathcal{I}_{\mathrm{WX,WY}}$ can sometimes be reduced to a complementation problem.

Problem (Weighted Complementation $\mathcal{C}_{\mathrm{WX,WY}}$). Given a WXA $\mathcal{A}$, construct a WYA $\overline{\mathcal{A}}$ over the same De Morgan lattice such that $(\|\overline{\mathcal{A}}\|, t) = \overline{(\|\mathcal{A}\|, t)}$ holds for every $t \in \Sigma^{K^*}$.

This reduction depends on the feasibility of computing the behavior of the infimum of two tree automata. Recall that this task is of polynomial complexity for WBA over distributive lattices (see Lemmata 2 and 3).

We now present two methods for solving the weighted inclusion problem. The first method uses a *glass-box* approach, i.e., it modifies the complementation constructions from the previous section to perform the lattice computations directly. This transformation generalizes the logical operators to their lattice counterparts. However, it only works for Boolean lattices.

The second method uses the algorithm for testing the inclusion of unweighted tree automata as a *black-box* in the sense that this algorithm is called several times in a systematic way until the desired aggregated infimum is found. Surprisingly, the black-box approach turns out to be more efficient than the glass-box method.

5.1 Glass-Box Approach

We now describe a construction that directly computes $\mathcal{I}_{\mathrm{WX,WY}}$ by generalizing the method used for deciding inclusion of unweighted tree automata presented in the previous section; hence the name *glass-box*.

Recall from Section 4 that the procedure deciding inclusion of two tree automata $\mathcal{A}$ and $\mathcal{A}'$ (i.e., whether $\mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A})$) required three steps: first construct an automaton $\overline{\mathcal{A}}$ accepting the complement of $\mathcal{L}(\mathcal{A})$; then, intersect $\mathcal{A}'$ and $\overline{\mathcal{A}}$, and finally decide emptiness of the resulting automaton. We have shown that the last two steps can be solved for WBA in polynomial time (see Lemmata 2 and 3). Thus, if we can solve the problem $\mathcal{C}_{\mathrm{WX,WB}}$, then we will also have a procedure that solves $\mathcal{I}_{\mathrm{WX,WB}}$.

There are some drawbacks to this approach. First, we can only apply it if an explicit and efficient complementation construction is known for the unweighted automata and we have to do it for every construction separately. Second, the construction presented below only works for Boolean lattices.

We will demonstrate this approach by considering the case of looping tree automata. Definition 5 shows us how to build an automaton that accepts the complement language of a given LA. Notice first that the transition relation of the automaton $\overline{\mathcal{A}}$ is equivalent to the following formula:

$$\bigwedge_{(q_0,\alpha,q_1,\ldots,q_k)\in Q\times\Sigma\times Q^k} q_0 \notin Q_0 \vee y \notin \Delta \vee \bigvee_{i\in K} q_i \in Q_i \ .$$

In the weighted complementation construction, we replace the Boolean operators by their lattice counterparts.

Definition 8. The *complement automaton* of a WLA $\mathcal{A} = (Q, \Sigma, S, \mathrm{in}, \mathrm{wt})$ over a Boolean lattice S is the WBA $\overline{\mathcal{A}} = (Q_c, \Sigma, S, \mathrm{in}_c, \mathrm{wt}_c, F_c)$ where

- $Q_c := S^Q$.
- For $\varphi \in Q_c$, $\mathrm{in}_c(\varphi) := \begin{cases} 1_S & \text{if } \varphi(q) \geq \mathrm{in}(q) \text{ for all } q \in Q \\ 0_S & \text{otherwise} \end{cases}$.
- For $\varphi_0, \ldots, \varphi_k \in Q_c$ and $\alpha \in \Sigma$, $\mathrm{wt}_c(\varphi_0, \alpha, \varphi_1, \ldots, \varphi_k) :=$

$$\bigotimes_{y=(q_0,\alpha,q_1,\ldots,q_k)\in Q\times\{\alpha\}\times Q^k} \overline{\varphi_0(q_0)} \oplus \overline{\mathrm{wt}(y)} \oplus \bigoplus_{i\in K} \varphi_i(q_i) \ .$$

- $F_c := \{\underline{0_S}\}$ where $\underline{0_S} : Q \to S : q \mapsto 0_S$.

The next theorem shows that this construction solves the weighted complementation problem $\mathcal{C}_{\mathrm{WL,WB}}$. The proof uses similar ideas to those of Theorem 6, generalized to Boolean lattices.

Theorem 9. *If $\mathcal{A}$ is a WLA over a Boolean lattice and $\overline{\mathcal{A}}$ is its complement automaton (Definition 8), then for all $t \in \Sigma^{K^*}$, $(\|\overline{\mathcal{A}}\|, t) = \overline{(\|\mathcal{A}\|, t)}$.* □

This construction gives us an automaton that has $|S|^{|Q|}$ states, where $|Q|$ is the number of states of the original automaton, and hence solves the problems $\mathcal{C}_{\mathrm{WL,WB}}$ and $\mathcal{I}_{\mathrm{WL,WB}}$ in exponential time.[3] This is optimal with respect to the complexity of the problems, as shown by Theorem 4.

5.2 Black-Box Approach

Since we already have a decision procedure for the unweighted problem $\mathcal{I}_{\mathrm{X,Y}}$ for $X, Y \in \{L, B, C, P\}$, we can use this to construct a *black-box* algorithm for $\mathcal{I}_{\mathrm{WX,WY}}$. This approach reduces the problem $\mathcal{I}_{\mathrm{WX,WY}}$ to several inclusion checks. The main advantage of such an approach is that one can use any procedure deciding the unweighted problem, including any optimizations developed for it, without modification.

The black-box reduction of $\mathcal{I}_{\mathrm{WX,WY}}$ to $\mathcal{I}_{\mathrm{X,Y}}$ is based on an idea from [15, 10] and exploits the fact that every lattice element can be represented as the infimum of all the meet prime elements above it. We demonstrate it first on the case of $\mathcal{I}_{\mathrm{WB,WB}}$.

Let $\mathcal{A} = (Q, \Sigma, S, \mathrm{in}, \mathrm{wt}, F)$ and $\mathcal{A}' = (Q', \Sigma, S, \mathrm{in}', \mathrm{wt}', F')$ be two WBA over the same De Morgan lattice S and $p \in S$ a meet prime element. We define the *cropped automata* $\mathcal{A}_p$ and $\mathcal{A}'_{\overline{p}}$ as the BA $(Q, \Sigma, I, \Delta, F)$ and $(Q', \Sigma, I', \Delta', F')$, respectively, where the initial state sets and transition relations are as follows:

- $I := \{q \in Q \mid \mathrm{in}(q) \not\leq p\}$, $\Delta := \{y \in Q \times \Sigma \times Q^k \mid \mathrm{wt}(y) \not\leq p\}$,
- $I' := \{q' \in Q' \mid \mathrm{in}'(q') \geq \overline{p}\}$, $\Delta' := \{y' \in Q' \times \Sigma \times Q'^k \mid \mathrm{wt}'(y') \geq \overline{p}\}$.

The transitions allowed in $\mathcal{A}_p$ (resp. $\mathcal{A}'_{\overline{p}}$) are exactly those transitions of $\mathcal{A}$ (resp. $\mathcal{A}'$) having weight $\not\leq p$ (resp. $\geq \overline{p}$). It is easy to see that this property transfers to the behavior of the weighted automata as follows. For all $t \in \Sigma^{K^*}$, $(\|\mathcal{A}\|, t) \leq p$ iff $t \notin \mathcal{L}(\mathcal{A}_p)$ and $(\|\mathcal{A}'\|, t) \geq \overline{p}$ iff $t \in \mathcal{L}(\mathcal{A}'_{\overline{p}})$. From this it follows that $\bigotimes_{t \in \Sigma^{K^*}} (\|\mathcal{A}\|, t) \oplus \overline{(\|\mathcal{A}'\|, t)} \leq p$ holds iff $\mathcal{L}(\mathcal{A}'_{\overline{p}}) \not\subseteq \mathcal{L}(\mathcal{A}_p)$.

We have assumed that the De Morgan lattice S is generated by the elements in the images of the initial distribution and transition weight functions of $\mathcal{A}$ and $\mathcal{A}'$. Since the number of meet prime elements in any distributive lattice is at most exponential in the number of elements generating it,[4] S has at most exponentially many meet prime elements measured in the sizes of $\mathcal{A}$ and $\mathcal{A}'$. Thus, this black-box approach requires at most exponentially many inclusion tests, each of which is itself exponential in the sizes of

[3] Recall that the size of $\mathcal{A}$ is $|Q| \log |S|$.

[4] Each meet prime element is the supremum of some generating elements and complements of generating elements.

these automata. This means that this algorithm solves the problem $\mathcal{I}_{\mathrm{WB,WB}}$ in exponential time, and hence is also optimal w.r.t. complexity.

Notice, additionally, that the reduction we used depends only on the number of meet prime elements and on the existence of an exponential-time inclusion test for the unweighted version of the automata, but not on the specific acceptance condition used. In other words, if $\mathcal{I}_{\mathrm{X,Y}}$ can be decided in exponential time, then $\mathcal{I}_{\mathrm{WX,WY}}$ is computable in exponential time, too.

Theorem 10. *Let S be a De Morgan lattice and X, Y two acceptance conditions such that $\mathcal{I}_{\mathrm{X,Y}}$ is in a complexity class* C *that includes* ExpTime. *Deciding whether $a \in S$ solves an instance of $\mathcal{I}_{\mathrm{WX,WY}}$ is also in* C. □

Corollary 11. *Let S be a De Morgan lattice. Deciding whether $a \in S$ solves an instance of $\mathcal{I}_{\mathrm{WX,WY}}$ is* ExpTime*-complete for $X, Y \in \{L, B, C, P\}$.* □

We presented a glass-box and a black-box approach to solve $\mathcal{I}_{\mathrm{WL,WB}}$, both of which are of optimal (ExpTime) complexity. However, a more fine-grained analysis of the algorithms shows that the black-box approach is in fact more efficient than the glass-box approach. The main consideration is that the number of meet prime elements of any Boolean lattice is logarithmic in the size of the lattice. Hence, if there are n meet prime elements, then the black-box approach involves n emptiness tests[5] of automata of size $2^{|Q|}$.

On the other hand, the glass-box approach applies a polynomial time algorithm to an automaton of size $(2^n)^{|Q|}$. Additionally, n is not independent from $|Q|$ but, given our assumption that the lattice S is generated by the input automata, n grows proportionally to $|Q|$. Thus, the bigger the input automata become, the more expensive the glass-box approach is, relative to the black-box procedure. This is surprising as it shows that an all-purpose procedure performs better than a specifically designed algorithm.

Obviously, looping tree automata are not the only ones that can be used in a glass-box approach. By generalizing the complementation construction for co-Büchi tree automata (see the technical report [5]) to Boolean lattices as we did for Definition 5, we could obtain a method for solving $\mathcal{C}_{\mathrm{WC,WB}}$. However, this would again result in an automaton having $|S|^{|Q|}$ states, which is less efficient than the black-box approach.

6 Conclusions

We have investigated some of the standard problems for unweighted automata on infinite trees and their generalizations to weighted tree automata.

[5] The emptiness of Büchi automata can be tested in quadratic time.

In particular, we have looked at the inclusion and complementation problems for parity tree automata. Despite the class of Büchi tree automata not being closed under complementation, for every looping or co-Büchi tree automaton it is possible to build a Büchi tree automaton of exponential size accepting the complement language. We demonstrated that these constructions can be generalized to the weighted setting, thus giving exponential time solutions to the weighted inclusion and complementation problems. Additionally, we described a black-box approach that solves these problems by performing several (unweighted) inclusion tests.

Since automata on infinite trees provide a clear characterization of reasoning in logics with the tree model property (e.g., some description logics), in our future work we will study the relation between the generalized problems for weighted automata and some non-standard inferences in these logics. In particular, we will study their application to uncertainty and multi-valued reasoning.

Acknowledgements

We are grateful to Christof Löding for sharing his ideas on the complexity of the inclusion problem for tree automata.

References

[1] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[2] F. Baader, J. Hladik, and R. Peñaloza. Automata can show PSPACE results for description logics. *Information and Computation*, 206(9,10):1045–1056, 2008.

[3] F. Baader, M. Knechtel, and R. Peñaloza. A generic approach for large-scale ontological reasoning in the presence of access restrictions to the ontology's axioms. In *Proc. ISWC'09*, volume 5823 of *LNCS*, pages 49–64. Springer, 2009.

[4] F. Baader and R. Peñaloza. Automata-based axiom pinpointing. *J. Autom. Reasoning*, 45(2):91–129, August 2010. Special Issue: Selected Papers from IJCAR'08.

[5] S. Borgwardt and R. Peñaloza. Complementation and inclusion of weighted automata on infinite trees: Revised version. LTCS-Report 11-02, TU Dresden, Germany, 2011. See http://lat.inf.tu-dresden.de/research/reports.html.

[6] G. Bruns and P. Godefroid. Model checking with multi-valued logics. In *Proc. ICALP'04*, volume 3142 of *LNCS*, pages 281–293. Springer, 2004.

[7] J. R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. CLMPS'60*, pages 1–11. Stanford University Press, 1960.

[8] N. Buhrke, H. Lescow, and J. Vöge. Strategy construction in infinite games with streett and rabin chain winning conditions. In *Proc. TACAS'01*, volume 1055 of *LNCS*, pages 207–224. Springer, 1996.

[9] M. Droste and P. Gastin. Weighted automata and weighted logics. In *Proc. ICALP'05*, volume 3580 of *LNCS*, pages 513–525, Lisbon, Portugal, 2005. Springer.

[10] M. Droste, W. Kuich, and G. Rahonis. Multi valued MSO logics over words and trees. *Fund. Inform.*, 84(3-4):305–327, 2008.

[11] M. Droste, W. Kuich, and H. Vogler. *Handbook of Weighted Automata*. Springer, 1st edition, 2009.

[12] M. Droste and G. Rahonis. Weighted automata and weighted logics on infinite words. In *Proc. DLT'06*, volume 4036 of *LNCS*, pages 49–58, Santa Barbara, CA, USA, 2006. Springer.

[13] M. Droste and H. Vogler. Weighted tree automata and weighted logics. *Theor. Comput. Sci.*, 366(3):228–247, 2006.

[14] G. Grätzer. *General Lattice Theory, Second Edition*. Birkhäuser Verlag, 2003.

[15] O. Kupferman and Y. Lustig. Lattice automata. In *Proc. VMCAI'07*, volume 4349 of *LNCS*, pages 199–213. Springer, 2007.

[16] O. Kupferman and M. Y. Vardi. Weak alternating automata and tree automata emptiness. In *Proc. STOC'98*, pages 224–233, New York, NY, USA, 1998. ACM.

[17] S. La Torre and A. Murano. Reasoning about co-Büchi tree automata. In *Proc. ICTAC'04*, pages 527–542, 2004.

[18] S. La Torre, A. Murano, and M. Napoli. Weak Muller acceptance conditions for tree automata. volume 2294 of *LNCS*, pages 285–288. Springer, 2002.

[19] S. Miyano and T. Hayashi. Alternating finite automata on omega-words. *Theor. Comput. Sci.*, 32:321–330, 1984.

[20] D. E. Muller and P. E. Schupp. Alternating automata on infinite trees. *Theor. Comput. Sci.*, 54(2-3):267–276, 1987.

[21] G. Rahonis. Weighted Muller tree automata and weighted logics. *J. Autom. Lang. Comb.*, 12(4):455–483, 2007.

[22] M. P. Schützenberger. On the definition of a family of automata. *Inform. Control*, 4(2-3):245–270, September 1961.

[23] H. Seidl. Deciding equivalence of finite tree automata. In *Proc. STACS'89*, volume 349 of *LNCS*, pages 480–492. Springer, 1989.

[24] U. Straccia. Reasoning within fuzzy description logics. *J. Artif. Intell. Res.*, 14:137–166, 2001.

[25] W. Thomas. Languages, automata, and logic. In *Handbook of Formal Languages*, volume 3, pages 389–455. Springer, 1997.

[26] M. Y. Vardi and T. Wilke. Automata: From logics to algorithms. In *Logic and Automata*, volume 2 of *Texts in Logic and Games*, pages 629–736. Amsterdam University Press, 2008.

[27] J. Yen. Generalizing term subsumption languages to fuzzy logic. In *Proc. IJCAI'91*, pages 472–477, 1991.