Dresden University of Technology
Institute for Theoretical Computer Science
Chair for Automata Theory

LTCS–Report

Complementation and Inclusion of Weighted Automata on Infinite Trees: Revised Version

Stefan Borgwardt Rafael Peñaloza

LTCS-Report 11-02

Complementation and Inclusion of Weighted Automata on Infinite Trees: Revised Version

Stefan Borgwardt   Rafael Peñaloza

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 report 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, 10].

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 [6], has been proven to hold also for weighted automata over finite and infinite words and trees [8, 11, 12, 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 [5].

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 [14].

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 [9]. 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. In particular, 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 closure 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.

2 Automata on Infinite Trees

The main object of our study are weighted automata on infinite trees, whose weights belong to a distributive lattice [13]. We give a brief introduction to lattices before formally defining our automata models.

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; i.e., $a \otimes b$ denotes the infimum of $\{a, b\}$.

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 one De Morgan law, e.g., $\overline{a \otimes b} = \overline{a} \oplus \overline{b}$ for every $a, b \in S$. In every De Morgan lattice the other De Morgan law $\overline{a \oplus b} = \overline{a} \otimes \overline{b}$ also holds and we have $a \le b$ iff $\overline{b} \le \overline{a}$ for all $a, b \in S$. Furthermore, $\overline{0_S} = 1_S$ and $\overline{1_S} = 0_S$.

A *Boolean lattice* is a De Morgan lattice such that $a \otimes \overline{a} = 0_S$ holds for every $a \in S$. This requirement is equivalent to the property $a \oplus \overline{a} = 1_S$. In this case, $\overline{a}$ is called the *complement* of a. Every Boolean lattice is isomorphic to a powerset lattice $(\mathcal{P}(X), \subseteq)$, for some set X.

An element p of a lattice S is called *meet prime* if for every $a, b \in S$, $a \otimes b \le p$ implies that either $a \le p$ or $b \le p$. The dual notion is that of a *join prime* element. Every element of a distributive lattice S can be computed as 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.

In the following we will often deal with finite Boolean lattices. We now prove a few useful facts about these structures.

Lemma 1. *Let S be a finite Boolean lattice.*

a) For all $a, b \in S$, $\overline{a} \oplus b = 1_S$ iff $a \le b$.

b) For every index set I and families $(f_i), (g_i) \in S^I$, the following holds:

$$\left(\bigoplus_{i\in I} f_i\right) \otimes \left(\bigotimes_{j\in I} g_j\right) \le \bigoplus_{i\in I} (f_i \otimes g_i) \quad .$$

Proof. If $a \le b$, then $\overline{a} \oplus b \ge \overline{a} \oplus a = 1_S$. Let now $\overline{a} \oplus b = 1_S$ and assume $a \not\le b$. Then $\overline{b} \not\le \overline{a}$ and thus $\overline{b} \ne \overline{b} \otimes \overline{a}$. But $(\overline{b} \otimes \overline{a}) \otimes b = 0_S$ and $(\overline{b} \otimes \overline{a}) \oplus b = \overline{a} \oplus b = 1_S$, which means that $\overline{b} \otimes \overline{a}$ is another complement of b. This contradicts the fact that the complement in S is unique.

For b), we have

$$\left(\bigoplus_{i\in I} f_i\right) \otimes \left(\bigotimes_{j\in I} g_j\right) = \bigoplus_{i\in I} \left(f_i \otimes \bigotimes_{j\in I} g_j\right) \le \bigoplus_{i\in I} (f_i \otimes g_i)$$

by distributivity of 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 $K := \{1, \ldots, k\}$. Our main object of study is the *full k-ary tree* K^*, where the root is denoted by the empty word ε, and the i-th successor of the node u is identified by ui.[1] For a node u, we denote the full subtree of K^* with root u by $u[K^*]$.

Sometimes we will also speak about (finite) subtrees, i.e., finite prefix-closed subsets of $u[K^*]$.[2] A node in such a (finite) tree T is called *inner node* if all its successors are also elements of T. It is called *leaf* if it has no successors in T. The set of all inner nodes of T is called *interior* of T and is denoted by $\mathrm{int}(T)$. The set of all leaves of T is called *frontier* of T and is denoted by $\mathrm{fr}(T)$. T is called *closed* if $T = \mathrm{int}(T) \cup \mathrm{fr}(T)$.

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 $ui \in p$. $\mathcal{P}ath(u[K^*])$ denotes the set of all infinite paths in $u[K^*]$. A *labeled tree* is a mapping $t : u[K^*] \to \Sigma$ for some labeling alphabet Σ. As usual, the set of all such mappings is denoted by $\Sigma^{u[K^*]}$.

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 maps each labeled tree to a value 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 2. 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 Büchi tree automaton (WBA)* if $n = 1$ and *weighted looping tree automaton (WLA)* if $n = 0$.

[1]One may also want to deal with infinite trees built over a ranked alphabet. However, such trees can always be embedded in a full k-ary tree as long as the arity of the symbols is bounded by k.

[2]Prefix-closed relative to u, i.e., up to and including u, but no prefix of u is considered.

A *run* of the WGBA $\mathcal{A}$ is a labeled tree $r \in Q^{K^*}$. This run is called *successful* if for every path $p \in \mathit{Path}(K^*)$ 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 $\overrightarrow{r(t,u)} := (r(u), t(u), r(u1), \dots, r(uk))$. 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 in the above definitions are restricted to a finitely generated (and thus finite) distributive sublattice. Hence, 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.

Every WGBA can be simulated by a WBA of polynomial size [27, 4], so every result for WBA also holds for WGBA. 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.

In some of our proofs we will make use of known results for more expressive acceptance conditions that we now briefly describe. The *parity* condition is based on a priority function $Q \to \mathbb{N}$. A run is accepted if on every path the minimal priority occurring infinitely often is even. The *Streett* acceptance condition is based on pairs $(E_1, F_1), \dots, (E_n, F_n)$ of state sets. A run is accepted if for every path $p \in \mathit{Path}(K^*)$ there is a pair (E_i, F_i) such that p contains infinitely many states from E_i or only finitely many states from F_i. The *Rabin chain* acceptance condition is also based on pairs $(E_1, F_1), \dots, (E_n, F_n)$ where the strict inclusions $E_1 \subsetneq F_1 \subsetneq E_2 \subsetneq \dots \subsetneq E_n \subsetneq F_n$ hold. A run is accepted if for every path $p \in \mathit{Path}(K^*)$ and every pair (E_i, F_i), p contains infinitely many states from F_i and only finitely many states from E_i.

The corresponding classes of tree automata are denoted by WPA, WSA and WRCA. For a given WBA or WCA it is easy to construct an equivalent WPA, WSA or WRCA. Also, parity, Streett and Rabin chain conditions can be reduced to each other. The class of tree series recognizable by WPA, WSA or WRCA strictly includes those recognizable by WBA or WCA. These in turn strictly include the tree series recognizable by WLA [25].

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. We will denote a generic weighted tree automaton as a tuple $(Q, \Sigma, S, \mathrm{in}, \mathrm{wt}, \mathfrak{F})$, where $\mathfrak{F}$ is the acceptance condition, i.e., it stands for several sets of states, pairs of sets of states or a priority function.

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

Finally, we consider also alternating versions of these automata.

Definition 3. An *alternating tree automaton* is a tuple $\mathcal{A} = (Q, \Sigma, I, \delta, \mathfrak{F})$, where Q, Σ, I, and $\mathfrak{F}$ are defined as for (unweighted) tree automata. The *transition function* $\delta : Q \times \Sigma \to \mathcal{F}(Q \times K)$ maps each state and input symbol to a monotone Boolean formula over $Q \times K$.

Intuitively, an atomic formula (q, i) means that the automaton goes to state q at the i-th successor of the current node. Conjunction $\wedge$ means that the automaton splits up into several copies which each pursues the directions given by the conjuncts. Disjunction $\vee$ means that the automaton can make a non-deterministic choice as to which disjunct to follow.

Starting from the root and an initial state, from one starting automaton many copies can be generated, depending on the non-deterministic choices. Basically, each of these copies consists of a path taken through K^* and an associated sequence of states. An input tree is accepted if it is possible to make each of the non-deterministic choices in such a way that the state sequences generated by the resulting copies all satisfy the acceptance condition $\mathfrak{F}$.

Alternating tree automata are designated by the prefix A to the classification, e.g., ABA stands for the class of all alternating tree automata with a Büchi acceptance condition.

Example 4. A non-deterministic unweighted tree automaton $(Q, \Sigma, I, \Delta, \mathfrak{F})$ can easily be transformed into an alternating one by replacing Δ with the function

$$\delta(q, \alpha) := \bigvee_{(q,\alpha,q_1,\ldots,q_k)\in\Delta} \bigwedge_{i\in K} (q_i, i) \enspace ,$$

i.e., the automaton non-deterministically chooses a transition to take and then sends one copy in every direction.

2.3 Basic Results

A result that will be useful later is that to compute the behavior of a weighted tree automaton on a given input tree t it suffices to consider a finite subtree of K^*. We prove a more general result.

Lemma 5. *Let S be a finite lattice, Σ an input alphabet, $t \in \Sigma^{K^*}$ an input tree, Q a state set and $P : K^* \times (Q \times \Sigma \times Q^k) \to S$ a function that assigns a lattice element to each pair (u, y) consisting of a node and a transition. There is a closed, finite subtree $T \subseteq K^*$ such that for every run $r \in Q^{K^*}$ we have*

$$\bigotimes_{u \in K^*} P(u, \overrightarrow{r(t,u)}) = \bigotimes_{u \in \mathrm{int}(T)} P(u, \overrightarrow{r(t,u)}) \ .$$

Proof. We first construct the infinite tree R of all finite subruns. The root of R is labeled by the empty subrun $r : \emptyset \to Q$ and its direct successors are labeled with all subruns $r : \{\varepsilon\} \to Q$ of depth 0. For each node of R of depth n that is labeled with a subrun r of depth $n-1$, its successors are labeled with all extensions of r to subruns r' of depth n. Since r has k^{n-1} leaves, there are $k^{n-1}|Q|^k$ such extensions. Thus, R is finitely branching.

The tree R' is now constructed from R by pruning it as follows. We traverse R depth-first and check the label $r \in Q^T$ of each node. If there is an extension of r to a finite subrun $r' \in Q^{T'}$ with

$$\bigotimes_{u \in \mathrm{int}(T)} P(u, \overrightarrow{r(t,u)}) >_S \bigotimes_{u \in \mathrm{int}(T')} P(u, \overrightarrow{r'(t,u)}) \ ,$$

then we continue. Otherwise, we remove all nodes below the current node.

Since S is finite, for every run $r \in Q^{K^*}$ the expression $P(u, \overrightarrow{r(t,u)})$ can only yield finitely many different values. Thus, there must be a depth below which the value of the infimum of all $P(u, \overrightarrow{r(t,u)})$ is not changed anymore. Since every infinite path in R uniquely corresponds to a run $r \in Q^{K^*}$, this path must have been pruned in the construction of R', and thus R' can have no infinite paths.

Since R' is still finitely branching, by König's Lemma, R' must be finite and thus have a maximal depth m. Now it is easily seen that the tree $T := \bigcup_{n=0}^{m} K^n$ has the desired property. $\square$

Note that this does not only hold for the infimum of the values $P(u, \overrightarrow{r(t,u)})$. Using the same arguments, an analogous result can be proven where $\bigotimes$ is substituted by $\bigoplus$.

Corollary 6. *For every weighted tree automaton $\mathcal{A} = (Q, \Sigma, S, \mathrm{in}, \mathrm{wt}, \mathfrak{F})$ with finite S and every input tree $t \in \Sigma^{K^*}$, there is a closed, finite subtree $T \subseteq K^*$*

with the property that

$$(\|\mathcal{A}\|, t) = (\|\mathcal{A}\|_T, t) := \bigotimes_{r \in \mathrm{succ}(\mathcal{A})} \mathrm{in}(r(\varepsilon)) \otimes \bigotimes_{u \in \mathrm{int}(T)} \mathrm{wt}(\overrightarrow{r(t,u)}) \ .$$

Proof. Apply Lemma 5 with $P(u, y) := \mathrm{wt}(y)$ for every node $u \in K^*$ and transition $y \in Q \times \Sigma \times Q^k$. □

This means that the computation of $(\|\mathcal{A}\|, t)$ for a given t can be carried out in a finite amount of time, which is of course due to the finiteness of S. We now reformulate the above results for unweighted automata.

Corollary 7. *Let Σ be an input alphabet, $t \in \Sigma^{K^*}$ an input tree, Q a state set and $P \subseteq K^* \times (Q \times \Sigma \times Q^k)$ a predicate on pairs (u, y) of nodes and transitions. There is a closed, finite subtree $T \subseteq K^*$ such that for every run $r \in Q^{K^*}$ we have*

$$\bigforall_{u \in K^*} P(u, \overrightarrow{r(t,u)}) \iff \bigforall_{u \in \mathrm{int}(T)} P(u, \overrightarrow{r(t,u)}).$$

□

Corollary 8. *For every unweighted tree automaton $\mathcal{A} = (Q, \Sigma, I, \Delta, \mathfrak{F})$ and every input tree $t \in \Sigma^{K^*}$, there is a closed, finite subtree $T \subseteq K^*$ with the property that*

$$t \in \mathcal{L}(\mathcal{A}) \iff \bigexists_{r \in \mathrm{succ}(\mathcal{A})} r(\varepsilon) \in I \wedge \bigforall_{u \in \mathrm{int}(T)} \overrightarrow{r(t,u)} \in \Delta \ .$$

□

Since weighted tree automata are a generalization of 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 9. *Let $\mathcal{A}, \mathcal{B}$ be two 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 $\mathcal{B}$ such that $(\|\mathcal{C}\|, t) = (\|\mathcal{A}\|, t) \otimes (\|\mathcal{B}\|, t)$ and $(\|\mathcal{C}'\|, t) = (\|\mathcal{A}\|, t) \oplus (\|\mathcal{B}\|, t)$ for all $t \in \Sigma^{K^*}$.*

Proof. These constructions closely follow the traditional constructions for intersection and union of finite automata, i.e., their state sets consist of the product and union of the original state sets, respectively. The weight functions can be combined in such a way that the desired behaviors are achieved. □

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 WBA.

Lemma 10. *Given a WBA $\mathcal{A}$, the value $\bigoplus_{t\in\Sigma^{K^*}}(\|\mathcal{A}\|, t)$ is computable in time polynomial in the size of $\mathcal{A}$.*

Proof. By combining the input alphabet Σ with the state set of $\mathcal{A}$, we can construct an automaton working over a singleton alphabet whose behavior on the unique input tree is exactly the desired supremum. We can then use the polynomial algorithm from [4] to compute this value. □

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 two tree automata, we are also motivated by the fact that this kind of automata can be used for reasoning tasks in Description Logics (DLs) [1]. DLs are decidable fragments of first-order logic that have been successfully employed in the area of knowledge representation.

Consider for example the DL $\mathcal{ALC}$. Formulae of $\mathcal{ALC}$, also 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. Such an interpretation can be viewed as a labeled directed graph over the node set $\Delta^{\mathcal{I}}$ with edges labeled by role names and nodes labeled by sets of concept names. Complex concept descriptions are interpreted as the sets

- $(C_1 \sqcap C_2)^{\mathcal{I}} := C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$,
- $(\neg C_1)^{\mathcal{I}} := \Delta^{\mathcal{I}} \setminus C_1^{\mathcal{I}}$, and
- $(\forall r.C_1)^{\mathcal{I}} := \{d \in \Delta^{\mathcal{I}} \mid \forall e \in \Delta^{\mathcal{I}} : (d, e) \in r^{\mathcal{I}} \to e \in C_1^{\mathcal{I}}\}$.

A *terminological axiom* of $\mathcal{ALC}$ is of the form $C \sqsubseteq D$ for two concept descriptions C, D. The axiom $C \sqsubseteq D$ 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 terminological axioms if there is a model of all axioms in $\mathcal{T}$ such that $C^{\mathcal{I}} \neq \emptyset$.

The DL $\mathcal{ALC}$ 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. For this purpose, all axioms are labeled with unique propositional variables, and one computes a so-called *pinpointing formula* that is a monotone Boolean formula over these labels identifying the responsible axioms. For example, a pinpointing formula $(\mathrm{ax}_1 \wedge \mathrm{ax}_2) \vee (\mathrm{ax}_2 \wedge \mathrm{ax}_3)$ would mean that ax_1 and ax_2 are enough to cause the unsatisfiability of C, but also ax_2 and ax_3 together have this effect. The set of all these formulae forms a distributive lattice and looping tree automata weighted over this lattice can be used to compute pinpointing formulae [4].

Other applications include the presence of access restrictions on the terminological axioms, where one wants to know whether a certain person has access to certain knowledge based on their access to the axioms [3], or fuzzy DLs, where concept descriptions are interpreted as fuzzy sets and one wants to decide whether a given concept description is satisfiable with at least a given certainty [28, 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 usually 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 of two languages. 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., the concept description $A \sqcap \forall \neg r. \neg A$ 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 of course 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.

This argument can be made for all DLs that do not have the tree model property, as long as the restrictions imposed by the additional constructors can be checked by a tree automaton. For example, the DL $\mathcal{ALCHI}$ allows for roles of the form r^- which are interpreted by $(r^-)^\mathcal{I} := (r^\mathcal{I})^{-1}$ and axioms $r \sqsubseteq s$ asserting that the relation $r^\mathcal{I} \subseteq s^\mathcal{I}$ holds between two roles. This logic also does not have the tree model property. However, role inclusion axioms and inverse roles can easily be checked by tree automata.

In the following sections, we will investigate the complexity of deciding 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 formally 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 for 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 the fact that LA, BA and CA are not closed under complement [25, 17], but, e.g., the complement of the language accepted by an LA can be recognized by a BA (see Theorem 13).

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 constructions [23, 26, 7, 15].

Theorem 11. *The problem $\mathcal{I}_{\mathrm{X,Y}}$ is* ExpTime*-complete for $X, Y \in \{L, B, C, P\}$.*

Proof. Note that we only have to show ExpTime-hardness of $\mathcal{I}_{\mathrm{L,L}}$ and that $\mathcal{I}_{\mathrm{P,P}}$ is in ExpTime since looping tree automata are special cases of all the other models and looping, Büchi, and co-Büchi tree automata can be transformed into parity tree automata in linear time.

We show ExpTime-hardness of $\mathcal{I}_{\mathrm{L,L}}$ by reduction of the inclusion problem for finite trees, i.e., given two automata $\mathcal{A}$ and $\mathcal{A}'$ on finite trees, decide whether $\mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A})$. It was shown in [23, Theorem 2.1] that this problem is ExpTime-complete.

The reduction uses a translation of automata on finite trees to looping automata. Given an automaton $\mathcal{A} = (Q, \Sigma, I, \Delta)$ on finite trees, the equivalent LA $\mathcal{B} = (Q', \Sigma', I', \Delta')$ is constructed as follows:

- $Q' := Q \cup \{q_\star, q_0\}$, where $q_\star, q_0$ are new states.
- $\Sigma' := \Sigma \cup \{\star\}$, where $\star$ is a new symbol.
- $I' := I \cup \{q_0\}$.
- $\Delta' := \Delta \cup \{(q, \star, q_\star, \dots, q_\star) \mid q \in Q\} \cup \{(q_\star, \alpha', q_\star, \dots, q_\star) \mid \alpha' \in \Sigma'\} \cup \{(q_0, \alpha, q_1, \dots, q_k) \mid \exists 1 \leq i \leq k. q_i = q_0 \wedge \forall j \neq i. q_j = q_\star\}$.

In this construction, every infinite tree $t' \in \Sigma'^{K^*}$ with a $\star$ on every path represents the finite tree $t \in \Sigma^T$ that is the subtree of t' before the first $\star$ of each path. $\mathcal{B}$ accepts all trees that have an infinite path without a $\star$ (guessed via q_0). On all trees that represent finite trees it behaves the same as $\mathcal{A}$. It is easy to see that $\mathcal{L}(\mathcal{A}') \subseteq \mathcal{L}(\mathcal{A})$ holds for two automata on finite trees iff $\mathcal{L}(\mathcal{B}') \subseteq \mathcal{L}(\mathcal{B})$ holds for their corresponding looping automata.

We will now give an algorithm that decides $\mathcal{I}_{\mathrm{P,P}}$ in time exponential in the size of the input PA $\mathcal{A}$ and $\mathcal{A}'$. Let n and k be the numbers of states and priorities of $\mathcal{A}$ and n' and k' those of $\mathcal{A}'$.

1. We translate $\mathcal{A}$ into an equivalent APA $\mathcal{B}$. The transition function δ of this automaton can be determined as in Example 4. This construction yields an automaton with n states and k priorities.
2. We use [26, Lemma 6.8] to construct an equivalent PA $\mathcal{B}'$.[3] This nondeterministic automaton has a number of states exponential and a number

[3]In [26] alternating automata are defined differently, but the two descriptions can be transformed into each other in polynomial time.

of priorities polynomial in n and k. Let $2^{p(n,k)}$ be a bound on the number of states and $p'(n,k)$ be a bound on the number of priorities of $\mathcal{B}'$ for suitable polynomials p and p'.

3. Now we have to construct an automaton $\mathcal{C}$ recognizing the intersection of $\mathcal{L}(\mathcal{A}')$ and $\mathcal{L}(\mathcal{B}')$. To do this, we use a standard product construction on the automata, where the acceptance conditions have first been rewritten as Streett conditions. For $\mathcal{B}'$, the equivalent Streett condition has at most $p'(n,k)$ pairs and for $\mathcal{A}'$ we need at most k' pairs. The product automaton then has as acceptance condition the conjunction of these two Streett conditions, which is again a Streett condition with at most $p'(n,k) + k'$ pairs. The number of states of $\mathcal{C}$ is bounded by $n'2^{p(n,k)}$.

4. We rewrite the SA $\mathcal{C}$ again as a PA $\mathcal{C}'$. For this we use the construction in [7, Theorem 7]. This construction takes a finite-state Streett game and constructs an equivalent Rabin chain game. Unweighted automata can be interpreted as special finite-state games, so this result also holds for Streett automata and Rabin chain automata (see, e.g., [25]). Rabin chain conditions can equivalently be expressed as parity conditions of the same size.

   We arrive at a PA with $O(n'2^{p(n,k)}(p'(n,k)+k')!)$ states and $O(p'(n,k)+k')$ priorities. Thus, the number of states is bounded by $n'2^{r(n,k,k')}$ and the number of priorities by $r'(n,k,k')$ for polynomials r and r'.[4]

5. By testing emptiness of $\mathcal{L}(\mathcal{C}')$, we effectively decide the inclusion problem for $\mathcal{A}$ and $\mathcal{A}'$. It was shown in [15, Theorem 5.1 (1)] that emptiness of the parity automaton $\mathcal{C}'$ is decidable in time $O\left((n'2^{r(n,k,k')})^{r'(n,k,k')}\right)$, i.e., exponential in the size of the input automata $\mathcal{A}$ and $\mathcal{A}'$. $\square$

We want to present the above construction in more detail for two special cases. The reasons for this are that at some point we may want to implement the inclusion check for certain applications (see Section 3) and later we want to adapt these constructions in order to solve the weighted version of the inclusion problem. For both of these reasons it is useful to have an explicit description of an efficient complementation construction.

The following construction details the complementation step of the above proof for a looping tree automaton $\mathcal{A}$. The resulting automaton $\overline{\mathcal{A}}$ uses a Büchi acceptance condition since complementation of alternating automata involves complementing the acceptance condition (see [19] for details).[5] Removing the alternation is done with a subset construction which results in an exponential blow-up of the number of states.

[4]The factorial $x!$ is bounded by $x^x = 2^{x \log x} \leq 2^{x^2}$.

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

Definition 12. 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

- $Q_c := \mathcal{P}(Q)$,
- $I_c := \{I\}$,
- $(Q_0, \alpha, Q_1, \dots, Q_k) \in \Delta_c$ iff for all $q_0 \in Q_0$ and $(q_0, \alpha, q_1, \dots, q_k) \in \Delta$ 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 13. *If $\mathcal{A}$ is an LA and $\overline{\mathcal{A}}$ its complement automaton (from Definition 12), then $\mathcal{L}(\overline{\mathcal{A}}) = \overline{\mathcal{L}(\mathcal{A})}$.*

Proof. Let $t \in \mathcal{L}(\overline{\mathcal{A}})$. Then there is a successful run $\overline{r} \in Q_c^{K^*}$ of $\overline{\mathcal{A}}$ on t. Assume that there also is a valid run $r \in Q^{K^*}$ of $\mathcal{A}$ on t. We now inductively construct a path $p \in \mathit{Path}(K^*)$ for which $r(u) \in \overline{r}(u)$ holds for all nodes $u \in p$.

- For $u = \epsilon$ we have $r(\epsilon) \in I = \overline{r}(\epsilon)$.
- Let $u \in p$ be a node for which $r(u) \in \overline{r}(u)$ holds. Since r and $\overline{r}$ are valid, we have $(r(u), t(u), r(u1), \dots, r(uk)) \in \Delta$ and $(\overline{r}(u), t(u), \overline{r}(u1), \dots, \overline{r}(uk)) \in \Delta_c$. By definition of Δ_c, there must be an $i \in K$ with $r(ui) \in \overline{r}(ui)$. We now append ui to the path p and continue.

The run $\overline{r}$ cannot fulfill the final state condition $\{\emptyset\}$ of $\overline{\mathcal{A}}$ on the path p, since every label along the path must contain at least one element. This contradicts the fact that $\overline{r}$ is successful, and thus t cannot be accepted by $\mathcal{A}$.

For the other inclusion, let $t \notin \mathcal{L}(\mathcal{A})$. By Corollary 8, there must be a closed, finite subtree $T \subseteq K^*$ on which no valid subrun exists. We now inductively construct a successful run $\overline{r} \in Q_c^{K^*}$ of $\overline{\mathcal{A}}$ on t for which every node $u \in T$ has the following property:

$$P(u) \quad \equiv \quad \bigforall_{r \in Q^{u[K^*]}} \Big[r(u) \in \overline{r}(u) \to \Big(\bigexists_{w \in u[K^*] \cap \mathrm{int}(T)} \overrightarrow{r(t, w)} \notin \Delta \Big) \Big]$$

This means that every mapping $r \in Q^{u[K^*]}$ that starts in a state $q_0 \in \overline{r}(u)$ at u must violate Δ at some node in $\mathrm{int}(T)$ that lies below u.

- If we set $\overline{r}(\epsilon) := \{I\}$, then $P(\epsilon)$ holds because of Corollary 8.
- If u is a leaf of T or $u \notin T$, we set $\overline{r}(ui) := \emptyset$ for each $i \in K$.
- Let now u be an inner node of T where $\overline{r}(u)$ has already been defined and $P(u)$ holds. We initially set $\overline{r}(ui) := \emptyset$ for every $i \in K$. Thus, $P(ui)$ trivially holds for every $i \in K$, but the transition $\overrightarrow{\overline{r}(t,u)}$ need not be valid. We now have to expand the label sets $\overline{r}(ui)$ in such a way that

  1. the transition $\overrightarrow{\overline{r}(t,u)}$ becomes valid and
  2. the properties $P(ui)$ are not violated.

  We do this by checking the conditions of Δ_c step by step.

  – Let $q_0 \in \overline{r}(u)$ and $y = (q_0, t(u), q_1, \ldots, q_k) \in \Delta$.

  – Assume that for each index $i \in K$ there is a mapping $r_i \in Q^{ui[K^*]}$ with $r_i(ui) = q_i$ that does not violate Δ below ui in $\mathrm{int}(T)$. Then we could join these mappings into a mapping $r \in Q^{u[K^*]}$ with $r(u) := q_0$ and $r(uiw) := r_i(uiw)$ for all $i \in K$ and $w \in K^*$. This mapping does not violate Δ below u in $\mathrm{int}(T)$, which contradicts $P(u)$.

  – Thus we can find an index $i \in K$ such that $P(ui)$ still holds after we add q_i to $\overline{r}(ui)$.

  After we have done this for every $q_0 \in \overline{r}(u)$ and every matching transition $y \in \Delta$, we have fully determined the successor labels $\overline{r}(ui)$ and $P(ui)$ still holds for every $i \in K$. Additionally, $\overrightarrow{\overline{r}(t,u)}$ now is a valid transition in Δ_c.

To show that $\overline{r}$ is a valid run of $\overline{\mathcal{A}}$ on t, we need to show that every transition is compatible with Δ_c. If the transition fully lies in T or $\overline{T}$, this is clear from the construction.

Let now $u \in \mathrm{fr}(T)$. Since $P(u)$ holds, all mappings $r \in Q^{u[K^*]}$ with $r(u) \in \overline{r}(u)$ must violate Δ in $u[K^*] \cap \mathrm{int}(T) = \emptyset$, which is clearly not possible. This implies that $\overline{r}(u) = \emptyset$, and thus, the transition $\overrightarrow{\overline{r}(t,u)} = (\emptyset, t(u), \emptyset, \ldots, \emptyset)$ is valid in Δ_c.

It is clear that $\overline{r}$ is successful since every infinite path must leave T at some node u and thus has the label $\emptyset$ at every node below u. This implies $t \in \mathcal{L}(\overline{\mathcal{A}})$. □

We will now present another explicit complementation construction: the complement automaton $\overline{\mathcal{A}}$ of a CA is a BA. The difficult step here is again to simulate the alternation by a nondeterministic automaton. In [20] such a simulation was presented for alternating Streett tree automata. However, in this special case we can adapt the construction for alternating Büchi *word* automata from [18].

Definition 14. The *complement automaton* of a CA $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ is the BA $\overline{\mathcal{A}} := (Q_c, \Sigma, I_c, \Delta_c, F_c)$, where

- $Q_c := \mathcal{P}(Q) \times \mathcal{P}(F)$,
- $I_c := \{(I \setminus F, I \cap F)\}$,
- $\chi_F(q) := \begin{cases} 1 & \text{if } q \in F \\ 0 & \text{otherwise} \end{cases}$,
- $\Big((Q_0^{(0)}, \emptyset), \alpha, (Q_0^{(1)}, Q_1^{(1)}), \ldots, (Q_0^{(k)}, Q_1^{(k)})\Big) \in \Delta_c$ iff
  - for all $q \in Q_0^{(0)}$ and $(q, \alpha, q_1, \ldots, q_k) \in \Delta$ there is an index $i \in K$ such that $q_i \in Q^{(i)}_{\chi_F(q_i)}$,
- for $Q_1^{(0)} \neq \emptyset$, $\Big((Q_0^{(0)}, Q_1^{(0)}), \alpha, (Q_0^{(1)}, Q_1^{(1)}), \ldots, (Q_0^{(k)}, Q_1^{(k)})\Big) \in \Delta_c$ iff
  - for all $q \in Q_0^{(0)}$ and $(q, \alpha, q_1, \ldots, q_k) \in \Delta$ there is an index $i \in K$ with $q_i \in Q_0^{(i)}$ and
  - for all $q \in Q_1^{(0)}$ and $(q, \alpha, q_1, \ldots, q_k) \in \Delta$ there is an index $i \in K$ with $q_i \in Q^{(i)}_{\chi_F(q_i)}$,
- $F_c := \{(Q_0, Q_1) \in Q_c \mid Q_1 = \emptyset\}$.

The correctness of this construction can be proven as in [18], taking into account that our automata are working over infinite trees instead of infinite words. The proof we present here follows the main ideas used in the proof of Theorem 13.

Lemma 15. *Let $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ be a CA and $\overline{\mathcal{A}}$ its complement automaton (from Definition 14). Then $\mathcal{L}(\overline{\mathcal{A}}) \subseteq \overline{\mathcal{L}(\mathcal{A})}$.*

Proof. Let $t \in \mathcal{L}(\overline{\mathcal{A}})$, i.e., there is a successful run $\overline{r} \in Q_c^{K^*}$ of $\overline{\mathcal{A}}$ on t, and assume that there also is a successful run $r \in Q^{K^*}$ of $\mathcal{A}$ on t. Then there is a finite tree $T \subseteq K^*$ outside of which no state from $Q \setminus F$ occurs in r, i.e., $r(u) \in F$ for all $u \in K^* \setminus T$. For a node $u \in K^*$, define $\overline{R}(u) := \overline{r}(u)_0 \cup \overline{r}(u)_1$. Then we can inductively construct an infinite path $p \in \mathit{Path}(K^*)$ for which $r(u) \in \overline{R}(u)$ holds for all $u \in p$ and $r(u) \in \overline{r}(u)_1$ holds for all $u \in p \cap u_0[K^*]$ for some node $u_0 \in K^*$:

- Since $r(\varepsilon) \in I$, either $r(\varepsilon) \in I \setminus F = \overline{r}(u)_0$ or $r(\varepsilon) \in I \cap F = \overline{r}(u)_1$ must hold, and thus $r(\varepsilon) \in \overline{R}(\varepsilon)$.
- Let $u \in p \cap T$ be a node with the property $r(u) \in \overline{R}(u)$. Since $\overrightarrow{r(t,u)} \in \Delta$ and $\overrightarrow{\overline{r}(t,u)} \in \Delta_c$, there must be an $i \in K$ such that $r(ui) \in \overline{r}(ui)_j$ holds for some $j \in \{0, 1\}$. Thus $r(ui) \in \overline{R}(ui)$ holds and we can append ui to the path p.

- If $u \in p \setminus T$ is the first node of p outside of T, then we continue the construction from above until we reach a node u' with $\overline{r}(u')_1 = \emptyset$. We must always reach such a node since $\overline{r}$ is successful.
- For u' we know from the definition of Δ_c that we can find $i \in K$ such that $r(u'i) \in \overline{r}(u'i)_1$. We set $u_0 := u'i$.
- Let $u \in p \cap u_0[K^*]$ with $r(u) \in \overline{r}(u)_1$. Since $\overline{r}(u)_1 \neq \emptyset$, we can choose $i \in K$ such that $r(ui) \in \overline{r}(ui)_1$.

Since $r(u) \in \overline{r}(u)_1$ holds for all nodes $u \in p \cap u_0[K^*]$ that occur below u_0 along p, $\overline{r}(u)_1$ can never be empty after u_0. This contradicts the success of $\overline{r}$. □

For this direction, it is easy to see the similarity to the proof of Theorem 13. The other direction is also similar. The property $P(u)$ is replaced by a more complex property **Fail**(u) and the proof is generally more complex to account for the different components of each state. Instead of Corollary 8, we have to use the more general version in Corollary 7 for this proof.

Lemma 16. *Let $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ be a CA and $\overline{\mathcal{A}}$ its complement automaton (from Definition 14). Then $\mathcal{L}(\overline{\mathcal{A}}) \supseteq \overline{\mathcal{L}(\mathcal{A})}$.*

Proof. Let $t \notin \mathcal{L}(\mathcal{A})$. We inductively construct a successful run $\overline{r} \in Q_c^{K^*}$ of $\overline{\mathcal{A}}$ on t. For every node $u \in K^*$ the following property **Fail**(u) will be satisfied.

$$\mathbf{Fail}(u) \equiv \bigwedge_{j=0}^{1} \forall_{r \in Q^{u[K^*]}} r(u) \in \overline{r}(u)_j \to \exists_{w \in u[K^*]} \mathbf{Fail}(w, \overrightarrow{r(t,w)})$$

$$\mathbf{Fail}(u, y = (q_0, \ldots)) \equiv$$

$$y \in \Delta \to \left(q_0 \notin F \wedge \forall_{\substack{r' \in Q^{u[K^*]} \\ r'(u) = q_0}} \neg\mathbf{Valid}(r', u) \vee \neg\mathbf{Success}(r', u) \right)$$

$$\mathbf{Valid}(r, u) \equiv \forall_{w \in u[K^*]} \overrightarrow{r(t,w)} \in \Delta$$

$$\mathbf{Success}(r, u) \equiv \forall_{p \in \mathcal{P}ath(u[K^*])} \mathrm{Inf}(r, p) \setminus F = \emptyset$$

Success(r, u) expresses that a run r is "successful below u", i.e., all infinite paths starting from u must contain only finitely many states from $Q \setminus F$.[6] The

[6]$\mathrm{Inf}(r, p)$ denotes the set of states occurring infinitely often in r along p.

property **Valid**(r, u) ensures that all transitions of a run r below a node u are valid transitions of $\mathcal{A}$. Using these two properties, we formulate **Fail**(u, y) by saying that if y is a valid transition at u, then the current state must be non-final and no valid run starting from this state can be successful. Finally, **Fail**(u) says that every run starting in a state occurring in $\overline{r}(u)_1$ must fail somewhere below u.

The property **Fail**(u, y) is clearly of the form required by Corollary 7, and thus **Fail**(u) is equivalent to a property **Fail**(u, T_u) for a closed, finite tree $T_u \subseteq u[K^*]$. This property is the same as **Fail**(u), except that in the conjunct for $j = 1$ "$w \in u[K^*]$" is replaced by "$w \in T_u$".

To start the construction of $\overline{r}$, we set $\overline{r}(\varepsilon) := (I \setminus F, I \cap F)$ and deduce **Fail**(ε) as follows. If **Fail**(ε) was not fulfilled, there would be a run $r \in Q^{K^*}$ with $r(\varepsilon) \in I \setminus F \subseteq I$ for which all transitions are valid and for every $w \in K^*$ with $r(w) \notin F$ there would be a run $r'_w \in Q^{w[K^*]}$ with $r'_w(w) = r(w)$ that is both valid and successful below w. Then we could construct a run $r' \in Q^{K^*}$ by replacing the labels of r on the subtree $w[K^*]$ with those of r'_w at every such node $w \in K^*$.[7] This run r' would be a valid and successful run of $\mathcal{A}$ on t, which contradicts the assumption that $t \notin \mathcal{L}(\mathcal{A})$.

Suppose now that $u \in K^*$ is a node where $\overline{r}(u)$ has already been defined and for which **Fail**(u, T_u) holds for some finite tree $T_u \subseteq u[K^*]$. For every $i \in K$ we construct $\overline{r}(ui)$ from $\overline{r}(u)$ in several steps.

- We initially set $\overline{r}(ui) := (\emptyset, \emptyset)$ for each $i \in K$, and thus **Fail**(ui) holds for our initial definiton of $\overline{r}(ui)$. But clearly, the resulting transition $\overrightarrow{\overline{r}(t, u)}$ need not satisfy the transition relation Δ_c. We now enlarge the sets $\overline{r}(ui)$ in such a way that **Fail**(ui) remains satisfied and $\overrightarrow{\overline{r}(t, u)}$ becomes a valid transition.
- For every $j \in \{0, 1\}$, $q \in \overline{r}(u)_j$ and $y = (q, t(u), q_1, \ldots, q_k) \in \Delta$, we do the following.
  - We choose one index $i \in K$ for which q_i is added to a component of $\overline{r}(ui)$. The index of this new component is determined according to Δ_c as follows:
    - If $\overline{r}(u) = \emptyset$, then q_i is added to $\overline{r}(ui)_0$ (resp. $\overline{r}(ui)_1$) if $q_i \notin F$ (resp. $q_i \in F$).
    - If $\overline{r}(u) \neq \emptyset$, then q_i is added to $\overline{r}(ui)_0$ (resp. $\overline{r}(ui)_1$) if $j = 0$ or $j = 1$ and $q_i \notin F$ (resp. $j = 1$ and $q_i \in F$).

    We choose i such that **Fail**(ui) remains satisfied after we add q_i to $\overline{r}(ui)$ as specified above. As we will show in the following, such an

[7]We only do this replacement for the first occurrence of a state from $Q \setminus F$, not inside a subtree that has already been replaced.

index always exists. For this, we make a case distinction depending on j.

  - Let $j = 0$ and assume that $\mathbf{Fail}(ui)$ is violated by adding q_i to $\overline{r}(ui)$. Then there are subruns $r_i \in Q^{ui[K^*]}$ with the properties
    - $r_i(ui) = q_i$ and
    - $\mathbf{Fail}(w, \overrightarrow{r_i(t, w)})$ is not satisfied for any $w \in ui[K^*]$, i.e., if $w \notin F$, then there is a valid and successful subrun $r'_w \in Q^{w[K^*]}$ with $r'_w(w) = r_i(w)$.

    As in the argument for $\mathbf{Fail}(\varepsilon)$, we can now construct a subrun $r' \in Q^{u[K^*]}$ with $\overrightarrow{r'(t, u)} = y$ which is valid and successful. This means that $\mathbf{Fail}(u, y)$ is not satisfied. If we now construct the subrun $r \in Q^{u[K^*]}$ by concatenating y and the subruns r_i, $\mathbf{Fail}(w, \overrightarrow{r(t, w)})$ is not satisfied for any $w \in u[K^*]$, which is a contradiction to $\mathbf{Fail}(u)$.
  - If $j = 1$, we could use the same argument as above. However, in this case we take a closer look at the finite tree T_{ui} because this will later enable us to show that $\overline{r}$ is successful. We will show that we can choose $i \in K$ such that the property $\mathbf{Fail}(ui, T_{ui})$ remains satisfied if we set $T_{ui} := T_u \cap ui[K^*]$.
    If we assume the converse, we can deduce that there exist subruns $r_i \in Q^{ui[K^*]}$ with the following properties:
    - $r_i(ui) = q_i$.
    - $\mathbf{Fail}(w, \overrightarrow{r_i(t, w)})$ is not satisfied for any $w \in T_u \cap ui[K^*]$.

    If we now construct the subrun $r \in Q^{u[K^*]}$ by concatenating the transition y and the subruns r_i, then it is easily seen that r starts in $r(u) = q$ and no $\mathbf{Fail}(w, \overrightarrow{r(t, w)})$ is satisfied for any $w \in T_u \cap ui[K^*]$ and for any $i \in K$. Furthermore, $\mathbf{Fail}(u, \overrightarrow{r(t, u)})$ is also not satisfied, since $\overrightarrow{r(t, u)} = y \in \Delta$, but $q \in F$. This means that r is a counterexample to $\mathbf{Fail}(u, T_u)$.

After we have done this for every j, q and y, the transition $\overrightarrow{\overline{r}(t, u)}$ is valid and the properties $\mathbf{Fail}(ui)$ still hold.

- Since $\mathbf{Fail}(ui)$ holds, there is a finite tree T_{ui} such that $\mathbf{Fail}(ui, T_{ui})$ holds. This tree can be determined as follows.
  - If $\overline{r}(u)_1 = \emptyset$, then T_{ui} can be determined from $\mathbf{Fail}(ui)$ using Corollary 7.
  - Otherwise, we can set $T_{j,ui} := T_{j,u} \cap ui[K^*]$. This is possible because of the above argument and the fact that we only added states to $\overline{r}(ui)_1$ in the case $j = 1$.

It remains to show that $\overline{r}$ is a successful run of $\overline{\mathcal{A}}$. For this we assume that there is a path $p \in \mathcal{P}ath(K^*)$ such that the set $\overline{r}(u)_1$ is empty only finitely often for nodes $u \in p$. Then there is a node $u \in p$ after which no empty set occurs in the second component of $\overline{r}$ along p. By construction of $\overline{r}$, the property $\mathbf{Fail}(u, T_u)$ must be satisfied for a finit tree $T_u \subseteq u[K^*]$.

Let v be the first node of p that lies outside of T_u. By construction of $\overline{r}$, $\mathbf{Fail}(v, T_v)$ must hold for the finite tree $T_v = T_u \cap v[K^*] = \emptyset$ since no empty set occurs in the second component along the path from u to v. Since $\mathbf{Fail}(v, T_v)$ is satisfied, this means that $\overline{r}(v)_1$ must be empty, which contradicts the assumption. Thus, $\overline{r}$ is a successful run of $\overline{\mathcal{A}}$ on t and $t \in \mathcal{L}(\overline{\mathcal{A}})$. □

We obtain the following theorem.

Theorem 17. *If $\mathcal{A}$ is a CA and $\overline{\mathcal{A}}$ its complement automaton (from Definition 14), then $\mathcal{L}(\overline{\mathcal{A}}) = \overline{\mathcal{L}(\mathcal{A})}$.* □

Notice that the construction of $\overline{\mathcal{A}}$ is again exponential in the size of $\mathcal{A}$. This in particular provides a direct proof of the fact that $\mathcal{I}_{\mathrm{C,B}}$ is in ExpTime.

Unfortunately, a similar construction for the problem $\mathcal{C}_{\mathrm{B,C}}$ is not possible. This was shown in [16] by means of a counterexample, i.e., a tree language that is recognizable by a BA, but whose complement is not recognizable by a CA.

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

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

Related to inclusion is the problem of deciding the equivalence of two unweighted tree automata, which can be decided by two inclusion tests. Generalizing this to Boolean lattices, one can compute the *weighted equivalence* of a WXA $\mathcal{A}$ and a WYA $\mathcal{A}'$ as the infimum of the two weighted inclusions of type $\mathcal{I}_{WX,WY}$ and $\mathcal{I}_{WY,WX}$. This value expresses the degree to which the two automata recognize the same tree series. For Boolean lattices, this value will be 1_S iff these tree series are equal.

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 $t \in \Sigma^{K^*}$.

Similar to the unweighted case, this reduction depends on the feasibility of computing the behavior of the binary infimum of two tree automata. Recall that this task is of polynomial complexity for WBA over distributive lattices (see Lemmata 9 and 10).

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 computations directly over the lattice. This transformation generalizes the logical operators to their lattice counterparts. However, this approach 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. A surprising result is that 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*. The advantage of a glass-box approach is that one has more direct control over the algorithm and can apply optimizations on a deeper level. However, this is also more laborious since it requires intimate knowledge of the inner workings of the original unweighted algorithm.

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 9 and 10). 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 several 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 12 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:

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

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

Definition 18. The *complement automaton* of a WLA $\mathcal{A} = (Q, \Sigma, S, \mathrm{in}, \mathrm{wt})$ 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 $\varphi_0, \dots, \varphi_k \in Q_c$ and $\alpha \in \Sigma$, $\mathrm{wt}_c(\varphi_0, \alpha, \varphi_1, \dots, \varphi_k) :=$
$$\bigotimes_{y=(q_0,\alpha,q_1,\dots,q_k)\in Q\times\{\alpha\}\times Q^k} \overline{\varphi_0(q_0)} \oplus \overline{\mathrm{wt}(y)} \oplus \bigoplus_{i\in K} \varphi_i(q_i) \ .$$
- $F_c := \{\underline{0_S}\}$ where $\underline{0_S} : Q \to S : q \mapsto 0_S$.

We now show that this construction solves the weighted complementation problem $\mathcal{C}_{\mathrm{WL,WB}}$. For this, we fix a WLA $\mathcal{A} = (Q, \Sigma, S, \mathrm{in}, \mathrm{wt})$ and an input tree $t \in \Sigma^{K^*}$ and need to show that $(\|\overline{\mathcal{A}}\|, t) = \overline{(\|\mathcal{A}\|, t)}$ holds. The next two sections are dedicated to proving the two halves of this claim. The proof uses similar ideas to those of Theorem 13, generalized to Boolean lattices.

5.1.1 Proof of $(\|\overline{\mathcal{A}}\|, t) \leq \overline{(\|\mathcal{A}\|, t)}$

We show this direction by proving the inequality $\mathrm{wt}_c(t, \overline{r}) \leq \overline{\mathrm{wt}(t, r)}$ for all $\overline{r} \in \mathrm{succ}(\overline{\mathcal{A}})$ and $r \in Q^{K^*}$. If $\mathrm{wt}_c(t, \overline{r}) = 0_S$ or $\mathrm{wt}(t, r) = 0_S$ this is trivially satisfied, so we fix two runs $\overline{r} \in \mathrm{succ}(\overline{\mathcal{A}})$ and $r \in Q^{K^*}$ with $\mathrm{wt}_c(t, \overline{r}) > 0_S$ and $\mathrm{wt}(t, r) > 0_S$.

We proceed by showing that $\mathrm{wt}_c(t, \overline{r}) \otimes \mathrm{wt}(t, r)$ is smaller than $a \otimes \overline{a} = 0_S$ for some suitably chosen $a \in S$. Looking at Theorem 13 one can already guess that this argument has to do with paths $p \in \mathcal{P}ath(K^*)$ for which $r(u) \in \overline{r}(u)$ holds for all $u \in p$. In the weighted case, this property is replaced by the value $\bigotimes_{u \in p} \overline{r}(u)(r(u))$. To be exact, a has the form

$$\bigoplus_{p \in \mathcal{P}ath(K^*, n)} \bigotimes_{u \in p} \overline{r}(u)(r(u))$$

for some $n \in \mathbb{N}$.

Lemma 19. *There is a depth $m \in \mathbb{N}$ such that*

$$\mathrm{wt}_c(t, \overline{r}) \leq \bigotimes_{p \in \mathcal{P}ath(K^*, m)} \bigoplus_{u \in p} \overline{\overline{r}(u)(r(u))} \ .$$

Proof. Since $\overline{r}$ is successful, there is a minimal depth $m \in \mathbb{N}$ such that any path p visits at least one node labeled by $\underline{0_S}$ before reaching depth m.

Let now $p \in \mathcal{P}ath(K^*, m)$ and assume that $\mathrm{wt}_c(t, \overline{r}) \not\leq \bigoplus_{u \in p} \overline{\overline{r}(u)(r(u))}$. Then $\bigoplus_{u \in p} \overline{\overline{r}(u)(r(u))} < 1_S$ and thus $\overline{r}(u)(r(u)) > 0_S$ holds for every $u \in p$. Hence there cannot be a node labeled with $\underline{0_S}$ along p in $\overline{r}$, which contradicts the above choice of m. □

We now show the second part, which leads to the "contradiction" $\mathrm{wt}(t, r) \otimes \mathrm{wt}_c(t, \overline{r}) \leq 0_S$.

Lemma 20. *For all $n \in \mathbb{N}$ the following inequation holds:*

$$\mathrm{wt}(t, r) \otimes \mathrm{wt}_c(t, \overline{r}) \leq \bigoplus_{p \in \mathcal{P}ath(K^*, n)} \bigotimes_{u \in p} \overline{r}(u)(r(u)) \ .$$

Proof. For $n = 0$ we have

$$\mathrm{wt}(t, r) \otimes \mathrm{wt}_c(t, \overline{r}) \leq \mathrm{in}(r(\epsilon)) \leq \overline{r}(\epsilon)(r(\epsilon)) = \bigoplus_{p \in \mathcal{P}ath(K^*, 0)} \bigotimes_{u \in p} \overline{r}(u)(r(u)) \ .$$

This holds since $\mathrm{wt}_c(t, \overline{r}) > 0_S$ and thus $\mathrm{in}_c(\overline{r}(\epsilon)) > 0_S$ and $\overline{r}(\epsilon)(r(\epsilon)) \geq \mathrm{in}(r(\epsilon))$.

Let now the inequation hold for some $n \in \mathbb{N}$. For $p \in \mathcal{P}ath(K^*, n)$, we let $p = \{p_0, \dots, p_n\}$, where $p_0 = \epsilon$ and the nodes are ordered by the successor relation. For any such path p we know that

$$\mathrm{wt}(t, r) \otimes \mathrm{wt}_c(t, \overline{r}) \leq \mathrm{wt}(\overrightarrow{r(t, p_n)}) \otimes \mathrm{wt}_c(\overrightarrow{\overline{r}(t, p_n)}) \ ,$$

and thus

$$\mathrm{wt}(t, r) \otimes \mathrm{wt}_c(t, \overline{r}) \leq \bigotimes_{p \in \mathcal{P}ath(K^*, n)} \mathrm{wt}(\overrightarrow{r(t, p_n)}) \otimes \mathrm{wt}_c(\overrightarrow{\overline{r}(t, p_n)}) \ .$$

Furthermore,

$$\begin{aligned}
&\overline{r}(p_n)(r(p_n)) \otimes \mathrm{wt}(\overrightarrow{r(t, p_n)}) \otimes \mathrm{wt}_c(\overrightarrow{\overline{r}(t, p_n)}) \\
&\quad = \overline{\left(\overline{\overline{r}(p_n)(r(p_n))} \oplus \overline{\mathrm{wt}(\overrightarrow{r(t, p_n)})}\right) \otimes} \\
&\qquad\quad \bigotimes_{y=(q_0, t(p_n), q_1, \dots, q_k)} \left(\overline{\overline{r}(p_n)(q_0)} \oplus \overline{\mathrm{wt}(y)}\right) \oplus \bigoplus_{i \in K} \overline{r}(p_n i)(q_i) \\
&\qquad\quad \text{(by de Morgan's law)} \\
&\quad \leq \overline{\left(\overline{\overline{r}(p_n)(r(p_n))} \oplus \overline{\mathrm{wt}(\overrightarrow{r(t, p_n)})}\right) \otimes} \\
&\qquad\quad \left(\left(\overline{\overline{r}(p_n)(r(p_n))} \oplus \overline{\mathrm{wt}(\overrightarrow{r(t, p_n)})}\right) \oplus \bigoplus_{i \in K} \overline{r}(p_n i)(r(p_n i))\right) \\
&\qquad\quad \text{(choose } y = \overrightarrow{r(t, p_n)}\text{)} \\
&\quad = \overline{r}(p_n)(r(p_n)) \otimes \mathrm{wt}(\overrightarrow{r(t, p_n)}) \otimes \bigoplus_{i \in K} \overline{r}(p_n i)(r(p_n i)) \\
&\qquad\quad \text{(by distributivity of } S\text{)} \\
&\quad = \bigoplus_{i \in K} \overline{r}(p_n)(r(p_n)) \otimes \mathrm{wt}(\overrightarrow{r(t, p_n)}) \otimes \overline{r}(p_n i)(r(p_n i)) \ .
\end{aligned}$$

Using the above two inequations we get

$$\begin{aligned}
&\mathrm{wt}(t, r) \otimes \mathrm{wt}_c(t, \overline{r}) \\
&\quad \leq \left(\bigoplus_{p \in \mathcal{P}ath(K^*, n)} \bigotimes_{j=0}^{n} \overline{r}(p_j)(r(p_j))\right) \otimes \left(\bigotimes_{p \in \mathcal{P}ath(K^*, n)} \mathrm{wt}(\overrightarrow{r(t, p_n)}) \otimes \mathrm{wt}_c(\overrightarrow{\overline{r}(t, p_n)})\right) \\
&\qquad\quad \text{(by induction hypothesis and the first inequation)} \\
&\quad \leq \bigoplus_{p \in \mathcal{P}ath(K^*, n)} \left(\bigotimes_{j=0}^{n-1} \overline{r}(p_j)(r(p_j))\right) \otimes \overline{r}(p_n)(r(p_n)) \\
&\qquad\qquad \otimes \mathrm{wt}(\overrightarrow{r(t, p_n)}) \otimes \mathrm{wt}_c(\overrightarrow{\overline{r}(t, p_n)}) \\
&\qquad\quad \text{(by Lemma 1 b))}
\end{aligned}$$

$$\leq \bigoplus_{p \in \mathcal{P}ath(K^*,n)} \bigoplus_{i \in K} \left(\bigotimes_{j=0}^{n-1} \overline{r}(p_j)(r(p_j)) \right) \otimes \overline{r}(p_n)(r(p_n))$$

$$\otimes \operatorname{wt}(\overrightarrow{r(t,p_n)}) \otimes \overline{r}(p_n i)(r(p_n i))$$

(by the second inequation and distributivity of S)

$$\leq \bigoplus_{p \in \mathcal{P}ath(K^*,n+1)} \bigotimes_{u \in p} \overline{r}(u)(r(u)) \ .$$

(combining p with $p_n i$)

This completes the proof by induction on n. □

This allows us to conclude the first half of the proof of correctness.

Lemma 21. $(\|\overline{\mathcal{A}}\|, t) \leq \overline{(\|\mathcal{A}\|, t)}$.

Proof. Combining Lemmata 19 and 20, we get $\operatorname{wt}(t,r) \otimes \operatorname{wt}_c(t,\overline{r}) \leq 0_S$. Lemma 1 now implies $\operatorname{wt}_c(t,\overline{r}) \leq \overline{\operatorname{wt}(t,r)}$. Since this holds for all $\overline{r} \in \operatorname{succ}(\overline{\mathcal{A}})$ and all runs r of $\mathcal{A}$, we have $(\|\overline{\mathcal{A}}\|, t) \leq \overline{(\|\mathcal{A}\|, t)}$. □

5.1.2 Proof of $(\|\overline{\mathcal{A}}\|, t) \geq \overline{(\|\mathcal{A}\|, t)}$

From Corollary 6 we know that there must be a closed, finite subtree $T \subseteq K^*$ such that for the computation of the weight $(\|\mathcal{A}\|, t)$, we only need to consider the nodes in T.

Similar to the proof of Theorem 13, we now define a successful run $\overline{r} \in Q_c^{K^*}$ of $\overline{\mathcal{A}}$ with $\operatorname{wt}_c(t,\overline{r}) = \overline{(\|\mathcal{A}\|, t)}$.

Definition 22. Let the run $\overline{r} \in Q_c^{K^*}$ be inductively defined as follows:

- $\overline{r}(\epsilon) := \text{in}$.
- If $u \in \operatorname{fr}(T)$ or $u \notin T$, set $\overline{r}(ui) := \underline{0_S}$ for each $i \in K$.
- If $u \in \operatorname{int}(T)$ is a node where $\overline{r}(u)$ has already been defined, set

$$\overline{r}(ui)(q) := \bigotimes_{\substack{r \in Q^{ui[K^*]} \\ r(ui) = q}} \bigoplus_{w \in ui[K^*] \cap \operatorname{int}(T)} \overline{\operatorname{wt}(\overrightarrow{r(t,w)})}$$

  for each $i \in K$ and $q \in Q$.

From this definition, it is already clear that $\overline{r}$ is a successful run of $\overline{\mathcal{A}}$, since every path will be labeled by $\underline{0_S}$ from some point on.

We additionally define a value $P(u)$ for each node $u \in T$:

$$P(u) := \bigotimes_{r \in Q^{u[K^*]}} \overline{\overline{r}(u)(r(u))} \oplus \bigoplus_{w \in u[K^*] \cap \mathrm{int}(T)} \overline{\mathrm{wt}(\overrightarrow{r(t,w)})}$$

Lemma 23. *The following hold:*

- $P(\epsilon) = \overline{(\|\mathcal{A}\|, t)}$.
- $P(ui) = 1_S$ *for all* $ui \in T$.

Proof. The first claim is easily proven by considering the definitions and Corollary 6.

Additionally, for any $ui \in T$ we have

$$\begin{aligned} P(ui) &= \bigotimes_{r \in Q^{ui[K^*]}} \overline{\overline{r}(ui)(r(ui))} \oplus \bigoplus_{w \in ui[K^*] \cap \mathrm{int}(T)} \overline{\mathrm{wt}(\overrightarrow{r(t,w)})} \\ &\geq \bigotimes_{r \in Q^{ui[K^*]}} \overline{\left(\bigoplus_{w \in ui[K^*] \cap \mathrm{int}(T)} \overline{\mathrm{wt}(\overrightarrow{r(t,w)})} \right)} \oplus \left(\bigoplus_{w \in ui[K^*] \cap \mathrm{int}(T)} \overline{\mathrm{wt}(\overrightarrow{r(t,w)})} \right) \\ &= 1_S \ , \end{aligned}$$

which proves the second claim. □

We now show that the run $\overline{r}$ has the claimed weight.

Lemma 24. *The following hold:*

a) $\mathrm{in}_c(\overline{r}(\epsilon)) = 1_S$.

b) $\mathrm{wt}_c(\overrightarrow{\overline{r}(t,u)}) = 1_S$ *for all* $u \notin T$.

c) $\mathrm{wt}_c(\overrightarrow{\overline{r}(t,u)}) = P(u)$ *for all* $u \in T$.

Proof. a) holds by definition of in_c and $\overline{r}(\epsilon)$ and b) follows from the fact that $\overline{r}(u) = \underline{0_S}$ holds for all $u \notin T$. For c), we consider two cases:

- $\mathrm{wt}_c(\overrightarrow{\overline{r}(t,u)}) = P(u)$ for every $u \in \mathrm{fr}(T)$:

$$\begin{aligned} \mathrm{wt}_c(\overrightarrow{\overline{r}(t,u)}) &= \bigotimes_{y=(q_0,t(u),q_1,\ldots,q_k)} \overline{\overline{r}(u)(q_0)} \oplus \overline{\mathrm{wt}(y)} \\ &= \bigotimes_{r \in Q^{u[K^*]}} \overline{\overline{r}(u)(r(u))} \oplus \overline{\mathrm{wt}(\overrightarrow{r(t,u)})} \\ &= P(u) \ . \end{aligned}$$

The second equation holds because of idempotency of $\otimes$. We consider any transition y at u as the beginning of every run $r \in Q^{u[K^*]}$ with $\overrightarrow{r(t,u)} = y$.

- $\mathrm{wt}_c(\overrightarrow{\overline{r}(t,u)}) = P(u)$ for every $u \in \mathrm{int}(T)$:

$$
\begin{aligned}
&\mathrm{wt}_c(\overrightarrow{\overline{r}(t,u)}) \\
&= \bigotimes_{y=(q_0,t(u),q_1,\dots,q_k)} \overline{\overline{r}(u)(q_0)} \oplus \overline{\mathrm{wt}(y)} \oplus \bigoplus_{i\in K} \overline{r}(ui)(q_i) \\
&= \bigotimes_{y=(q_0,t(u),q_1,\dots,q_k)} \overline{\overline{r}(u)(q_0)} \oplus \overline{\mathrm{wt}(y)} \oplus \bigoplus_{i\in K} \bigotimes_{\substack{r_i\in Q^{ui[K^*]}\\ r_i(ui)=q_i}} \bigoplus_{w\in ui[K^*]\cap \mathrm{int}(T)} \overline{\mathrm{wt}(\overrightarrow{r_i(t,w)})} \\
&= \bigotimes_{y=(q_0,t(u),q_1,\dots,q_k)} \overline{\overline{r}(u)(q_0)} \oplus \overline{\mathrm{wt}(y)} \oplus \\
&\qquad \bigotimes_{\substack{r_1\in Q^{u1[K^*]}\\ r_1(u1)=q_1}} \cdots \bigotimes_{\substack{r_k\in Q^{uk[K^*]}\\ r_k(uk)=q_k}} \bigoplus_{i\in K} \bigoplus_{w\in ui[K^*]\cap \mathrm{int}(T)} \overline{\mathrm{wt}(\overrightarrow{r_i(t,w)})} \\
&\qquad \text{(by distributivity of } S) \\
&= \bigotimes_{y=(q_0,t(u),q_1,\dots,q_k)} \bigotimes_{\substack{r_1\in Q^{u1[K^*]}\\ r_1(u1)=q_1}} \cdots \bigotimes_{\substack{r_k\in Q^{uk[K^*]}\\ r_k(uk)=q_k}} \\
&\qquad \overline{\overline{r}(u)(q_0)} \oplus \overline{\mathrm{wt}(y)} \oplus \bigoplus_{i\in K} \bigoplus_{w\in ui[K^*]\cap \mathrm{int}(T)} \overline{\mathrm{wt}(\overrightarrow{r_i(t,w)})} \\
&\qquad \text{(by distributivity of } S) \\
&= \bigotimes_{r\in Q^{u[K^*]}} \overline{\overline{r}(u)(r(u))} \oplus \bigoplus_{w\in u[K^*]\cap \mathrm{int}(T)} \overline{\mathrm{wt}(\overrightarrow{r(t,w)})} \\
&\qquad \text{(concatenate } y \text{ and } r_1,\dots,r_k \text{ to } r) \\
&= P(u)\ .
\end{aligned}
$$

These two cases account for all the nodes of T, since T is closed. □

This completes the second half of the proof of correctness.

Lemma 25. $(\|\overline{\mathcal{A}}\|, t) \geq \overline{(\|\mathcal{A}\|, t)}$.

Proof. We deduce

$$(\|\overline{\mathcal{A}}\|, t) \geq \mathrm{wt}_c(t, \overline{r}) = \mathrm{in}_c(\overline{r}(\epsilon)) \otimes \bigotimes_{u\in K^*} \mathrm{wt}_c(\overrightarrow{\overline{r}(t,u)}) = \overline{(\|\mathcal{A}\|, t)}$$

from Lemmata 23 and 24. □

Theorem 26. *If $\mathcal{A}$ is a WLA and $\overline{\mathcal{A}}$ is its complement automaton (from Definition 18), then for all $t \in \Sigma^{K^*}$, $(\|\overline{\mathcal{A}}\|, t) = \overline{(\|\mathcal{A}\|, t)}$.*

Proof. Since the construction of $\overline{\mathcal{A}}$ does not depend on the input tree t, this follows from Lemmata 21 and 25. $\square$

This construction gives us an automaton that has $|S|^{|Q|}$ states, where $|Q|$ is the number of states of the original automaton, and hence can be used to solve the problems $\mathcal{C}_{\mathrm{WL,WB}}$ and $\mathcal{I}_{\mathrm{WL,WB}}$ in exponential time. This is optimal with respect to the complexity of the problems, as shown by Theorem 11.

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, since this procedure needs not be modified.

The black-box reduction of $\mathcal{I}_{\mathrm{WX,WY}}$ to $\mathcal{I}_{\mathrm{X,Y}}$ is based on an idea from [14, 9] 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 show that this property is transferred to the behavior of the weighted automata as follows. We have $(\|\mathcal{A}\|, t) \leq p$ iff $t \notin \mathcal{L}(\mathcal{A}_p)$ and $(\|\mathcal{A}'\|, t) \geq \overline{p}$ iff $t \in \mathcal{L}(\mathcal{A}'_{\overline{p}})$ for all $t \in \Sigma^{K^*}$. From this it follows that $\bigotimes_{t \in \Sigma^{K^*}} (\|\mathcal{A}\|, t) \oplus \overline{(\|\mathcal{A}'\|, t)} \leq p$ holds iff $\mathcal{L}(\mathcal{A}'_{\overline{p}}) \nsubseteq \mathcal{L}(\mathcal{A}_p)$.

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,[8] 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 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. This yields the next theorem.

Theorem 27. *Let S be a De Morgan lattice and X, Y be two acceptance conditions such that the problem $\mathcal{I}_{\mathrm{X,Y}}$ is decidable in some complexity class* C *that includes* ExpTime. *Then the problem of deciding whether a given value $a \in S$ solves an instance of $\mathcal{I}_{\mathrm{WX,WY}}$ is also in* C. □

Corollary 28. *Let S be a De Morgan lattice. The problem of deciding whether a given value $a \in S$ solves an instance of $\mathcal{I}_{\mathrm{WX,WY}}$ is* ExpTime*-complete for $X, Y \in \{L, B, C, P\}$.* □

5.3 Complexity Comparison

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[9] of automata of size $2^{|Q|}$.

On the other hand, the glass-box approach will apply a polynomial time algorithm to an automaton of size $(2^n)^{|Q|}$. Additionally, n cannot be considered independently from $|Q|$, but, given our assumption that the lattice S is generated by the input automata, n actually grows proportionally to $|Q|$. This means that the bigger the input automata become, the more expensive the glass-box approach is, relative to the black-box procedure. This is surprising because 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. In fact, by generalizing the construction from Definition 14

[8]Each meet prime element can be expressed as the supremum of some generating elements and complements of generating elements.

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

to arbitrary Boolean lattices as we did for Definition 12, 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. 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] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003.

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

[3] Franz Baader, Martin Knechtel, and Rafael Peñaloza. A generic approach for large-scale ontological reasoning in the presence of access restrictions to the ontology's axioms. In Abraham Bernstein, David R. Karger, Tom

Heath, Lee Feigenbaum, Diana Maynard, Enrico Motta, and Krishnaprasad Thirunarayan, editors, *Proc. of the 8th Int. Semantic Web Conf. (ISWC'09)*, volume 5823 of *Lecture Notes in Computer Science*, pages 49–64. Springer-Verlag, 2009.

[4] Franz Baader and Rafael Peñaloza. Automata-based axiom pinpointing. *Journal of Automated Reasoning*, 45(2):91–129, August 2010. Special Issue: Selected Papers from IJCAR'08.

[5] Glenn Bruns and Patrice Godefroid. Model checking with multi-valued logics. In *Proc. of the 31st Int. Coll. on Automata, Languages and Programming (ICALP'04)*, volume 3142 of *Lecture Notes in Computer Science*, pages 281–293. Springer-Verlag, 2004.

[6] J. Richard Büchi. On a decision method in restricted second order arithmetic. In E. Nagel et al., editors, *Proc. of the Int. Congr. on Logic, Methodology and Philosophy of Science (CLMPS'60)*, pages 1–11. Stanford University Press, 1960.

[7] Nils Buhrke, Helmut Lescow, and Jens Vöge. Strategy construction in infinite games with streett and rabin chain winning conditions. In Tiziana Margaria and Bernhard Steffen, editors, *Proc. of the 7th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01)*, volume 1055 of *Lecture Notes in Computer Science*, pages 207–224. Springer-Verlag, 1996.

[8] Manfred Droste and Paul Gastin. Weighted automata and weighted logics. In Luis Caires, Giuseppe Italiano, Luis Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Proc. of the 32nd Int. Coll. on Automata, Languages and Programming (ICALP'05)*, volume 3580 of *Lecture Notes in Computer Science*, pages 513–525, Lisbon, Portugal, 2005. Springer-Verlag.

[9] Manfred Droste, Werner Kuich, and George Rahonis. Multi valued MSO logics over words and trees. *Fundamenta Informaticae*, 84(3-4):305–327, 2008.

[10] Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer-Verlag, 1st edition, 2009.

[11] Manfred Droste and George Rahonis. Weighted automata and weighted logics on infinite words. In Oscar H. Ibarra and Zhe Dang, editors, *Proc. of the 10th Int. Conf. on Developments of Language Theory (DLT'06)*, volume 4036 of *Lecture Notes in Computer Science*, pages 49–58, Santa Barbara, CA, USA, 2006. Springer-Verlag.

[12] Manfred Droste and Heiko Vogler. Weighted tree automata and weighted logics. *Theoretical Computer Science*, 366(3):228–247, 2006.

[13] George Grätzer. *General Lattice Theory, Second Edition.* Birkhäuser Verlag, 2003.

[14] Orna Kupferman and Yoad Lustig. Lattice automata. In *Proc. of the 8th Int. Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI'07)*, volume 4349 of *Lecture Notes in Computer Science*, pages 199–213. Springer-Verlag, 2007.

[15] Orna Kupferman and Moshe Y. Vardi. Weak alternating automata and tree automata emptiness. In *Proc. of the 30th Annual ACM Symp. on Theory of Computing (STOC'98)*, pages 224–233, New York, NY, USA, 1998. ACM.

[16] Salvatore La Torre and Aniello Murano. Reasoning about co-Büchi tree automata. In *Proc. of the 1st Int. Coll. on Theoretical Aspects of Computing (ICTAC'04)*, pages 527–542, 2004.

[17] Salvatore La Torre, Aniello Murano, and Margherita Napoli. Weak Muller acceptance conditions for tree automata. volume 2294 of *Lecture Notes in Computer Science*, pages 285–288. Springer-Verlag, 2002.

[18] Satoru Miyano and Takeshi Hayashi. Alternating finite automata on omega-words. *Theoretical Computer Science*, 32:321–330, 1984.

[19] David E. Muller and Paul E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54(2-3):267–276, 1987.

[20] David E. Muller and Paul E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141(1-2):69–107, 1995.

[21] George Rahonis. Weighted Muller tree automata and weighted logics. *Journal of Automata, Languages and Combinatorics*, 12(4):455–483, 2007.

[22] Marcel Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2-3):245–270, September 1961.

[23] Helmut Seidl. Deciding equivalence of finite tree automata. In B. Monien and R. Cori, editors, *Proc. of the 6th Annual Symp. on Theoretical Aspects of Computer Science (STACS'89)*, volume 349 of *Lecture Notes in Computer Science*, pages 480–492. Springer-Verlag, 1989.

[24] Umberto Straccia. Reasoning within fuzzy description logics. *Journal of Artificial Intelligence Research*, 14:137–166, 2001.

[25] Wolfgang Thomas. Languages, automata, and logic. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 389–455. Springer-Verlag, 1997.

[26] Moshe Y. Vardi and Thomas Wilke. Automata: From logics to algorithms. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata*, volume 2 of *Texts in Logic and Games*, pages 629–736. Amsterdam University Press, 2008.

[27] Moshe Y. Vardi and Pierre Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Sciences*, 32(2):183–221, 1986.

[28] John Yen. Generalizing term subsumption languages to fuzzy logic. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 472–477, 1991.