Noname manuscript No.
(will be inserted by the editor)

Automata-based Axiom Pinpointing

Franz Baader · Rafael Peñaloza

the date of receipt and acceptance should be inserted later

Abstract Axiom pinpointing has been introduced in description logics (DL) to help the user understand the reasons why consequences hold by computing minimal subsets of the knowledge base that have the consequence in question (MinA). Most of the pinpointing algorithms described in the DL literature are obtained as extensions of tableau-based reasoning algorithms for computing consequences from DL knowledge bases. In this paper, we show that automata-based algorithms for reasoning in DLs and other logics can also be extended to pinpointing algorithms. The idea is that the tree automaton constructed by the automata-based approach can be transformed into a weighted tree automaton whose so-called behaviour yields a pinpointing formula, i.e., a monotone Boolean formula whose minimal valuations correspond to the MinAs. We also develop an approach for computing the behaviour of a given weighted tree automaton. We use the DL $\mathcal{SI}$ as well as Linear Temporal Logic (LTL) to illustrate our new pinpointing approach.

1 Introduction

Description logics (DLs) [2] are a family of logic-based knowledge representation formalisms, which are employed in various application domains, such as natural language processing, configuration, databases, and bio-medical ontologies, but their most notable success so far is the adoption of the DL-based language OWL [21] as standard ontology language for the semantic web. As the size of DL-based ontologies grows, tools that support improving the quality of such ontologies become more important. DL reasoners [20, 19, 38] can be used to detect inconsistencies and to infer other implicit consequences, such as subsumption relationships between concepts or instance relationships between individuals and concepts. However, for a developer or user of a DL-based ontology, it is often quite hard to understand why a certain consequence computed by the reasoner actually follows from the knowledge base. For example, in the current

First author partially supported by NICTA, Canberra Research Lab., and second author funded by the German Research Foundation (DFG) under grant GRK 446.

Theoretical Computer Science, TU Dresden, Germany
E-mail: {baader,penaloza}@tcs.inf.tu-dresden.de

DL version of the medical ontology SNOMED CT,[1] the concept *Amputation-of-Finger* is classified as a subconcept of *Amputation-of-Arm*. Finding the six axioms that are responsible for this error [10] among the more than 350,000 terminological axioms of SNOMED without support by an automated reasoning tool is not easy.

Axiom pinpointing [34] has been introduced to help developers or users of DL-based ontologies understand the reasons why a certain consequence holds by computing minimal subsets of the knowledge base that have the consequence in question (MinA). There are two general approaches for computing MinAs: the *black-box* approach and the *glass-box* approach. The most naïve variant of the black-box approach considers all subsets of the ontology, and computes for each of them whether it still has the consequence or not. More sophisticated versions [35, 22] use a variant of Reiter's [32] hitting set tree algorithm to compute all MinAs. Instead of applying such a black-box approach to a large ontology, one can also first try to find a small and easy to compute subset of the ontology that contains all MinAs, and then apply the black-box approach to this subset [10]. The main advantage of the black-box approach is that it can use existing highly-optimized DL reasoners unchanged. However, it may be necessary to call the reasoner an exponential number of times. In contrast, the glass-box approach tries to find all MinAs by a single run of a modified reasoner.

Most of the glass-box pinpointing algorithms described in the DL literature (e.g., [4, 34, 33, 27, 25]) are obtained as extensions of tableau-based reasoning algorithms [9] for computing consequences from DL knowledge bases. The pinpointing algorithms and proofs of their correctness in these papers are given for a specific DL and a specific type of knowledge base only, and it is not clear to which of the known tableau-based algorithms for DLs the approaches really generalize. For example, the pinpointing extension described in [25], which can deal with general concept inclusions (GCIs) in the DL $\mathcal{ALC}$, follows the approach introduced in [4], but since GCIs require the introduction of so-called blocking conditions into the tableau-based algorithm to ensure termination [9], there are some new non-trivial problems to be solved.

To overcome the problem of having to design a new pinpointing extension for every tableau-based algorithm, we have introduced in [5] a general approach for extending tableau-based algorithms to pinpointing algorithms. This approach has, however, some annoying limitations. First, it only applies to tableau-based algorithms that terminate without requiring any cycle-checking mechanism such as blocking. Second, termination of the tableau-based algorithm one starts with does not necessarily transfer to its pinpointing extension. Though these problems can, in principle, be solved by restricting the general framework to so-called forest tableaux [8, 7], this solution makes the definitions and proofs quite complicated and less intuitive. Also, the approach can still only handle the most simple version of blocking, usually called subset blocking in the DL literature.

In the present paper, we propose a different general approach for obtaining glass-box pinpointing algorithms, which also applies to DLs for which the termination of tableau-based algorithms requires the use of appropriate blocking conditions. It is well-known that automata working on infinite trees can often be used to construct worst-case optimal decision procedures for such DLs [13, 26, 11, 14, 3]. In this automata-based approach, the input inference problem Γ is translated into a tree automaton $\mathcal{A}_\Gamma$, which is then tested for emptiness. Basically, our approach transforms the tree automaton $\mathcal{A}_\Gamma$ into a weighted tree automaton working on infinite trees, whose so-called behaviour yields

[1] see http://www.ihtsdo.org/our-standards/

a pinpointing formula, i.e., a monotone Boolean formula that encodes all the MinAs of Γ. To obtain an actual pinpointing algorithm, we had to develop an algorithm for computing the behaviour of weighted tree automata working on infinite trees. When we started our work, we could not find such an algorithm in the quite extensive literature on weighted automata. In fact, although weighted automata working on *finite trees* [37] and weighted automata working on *infinite words* [16] have been considered for quite a while, the research on weighted automata working on *infinite trees* has started only recently [23, 15]. During the development of our work, an alternative algorithm for computing the behaviour of weighted tree automata working on infinite trees has independently been developed in [15]. It turns out, however, that using this algorithm in our pinpointing application basically yields a black-box approach for pinpointing, rather than a glass-box approach, as our algorithm does (see Section 5.4).

We will use the DL $\mathcal{SI}$, which extends the basic DL $\mathcal{ALC}$ [36] with transitive and inverse roles, as well as Linear Temporal Logic (LTL) [28, 17] to illustrate our new pinpointing approach. The use of $\mathcal{SI}$ is, on the one hand, motivated by the fact that the presence of inverses in $\mathcal{SI}$ requires tableau-based algorithms to use a blocking condition that is more sophisticated than subset blocking [9]. Consequently, our general results on tableau-based approach for pinpointing [8, 7] do not apply to this DL. On the other hand, the extension of their approach to $\mathcal{SI}$ is mentioned as an open problem in [25]. The automata used to decide satisfiability in $\mathcal{SI}$ are so-called looping automata, which do not use an acceptance condition. Our choice of LTL as a second example is, on the one hand, motivated by the fact that automata-based algorithms for LTL require the use of automata with a Büchi acceptance condition.[2] One the other hand, we believe that pinpointing can also be a useful inference service in applications of LTL. In LTL model checking [12], it does not make sense to check whether a system description satisfies a given LTL formula if this formula or its negation is unsatisfiable. Pinpointing could help the user to find the reasons for the unsatisfiability and thus correct the formula. In LTL synthesis [29, 24] one tries to generate a reactive finite-state system from a formal specification, which is given as an LTL formula. If the formula is unsatisfiable, then the specification is obviously faulty, and needs to be repaired. Pinpointing could be used to support the repair process by clarifying the reasons for unsatisfiability.

In the next section, we first introduce the DL $\mathcal{SI}$ and the temporal logic LTL, and then recall the relevant definitions regarding pinpointing. Section 3 defines generalized Büchi tree automata, their restrictions to Büchi tree automata and looping tree automata, and their generalization to the weighted case. In Section 4, we first present our general approach for automata-based pinpointing, which is based on the notion of an axiomatic automaton and its transformation into a pinpointing automaton. Then, we show that this approach can be applied to $\mathcal{SI}$ and LTL by introducing axiomatic automata for these logics. The pinpointing automaton is a weighted automaton whose behaviour is the pinpointing formula. Thus, to apply our approach in practice, one needs to be able to compute the behaviour of weighted generalized Büchi tree automata. In Section 5, we first show how to compute the behaviour of weighted Büchi tree automata. Second, we explain how this computation can be simplified for the case of weighted *looping* tree automata. For the DL $\mathcal{SI}$, the pinpointing automaton con-

[2] We could, of course, also have used a DL with transitive closure of roles [1] for this purpose. However, such DLs are until now not used in applications, and we also wanted to make clear that our approach for automata-based pinpointing is not restricted to Description Logics.

structed by our approach is such a weighted looping tree automaton. Third, we define a behaviour-preserving polynomial-time reduction of weighted *generalized* Büchi tree automata to weighted Büchi tree automata, which yields an approach for computing the behaviour of weighted generalized Büchi tree automata. For the temporal logic LTL, the pinpointing automaton constructed by our approach is a weighted generalized Büchi tree automaton. Fourth, we compare our approach for computing the behaviour of weighted Büchi tree automata with the one developed in [15]. Section 6 summarizes the results of the paper and gives some perspectives on further research.

This work extends the results in [6] (the conference version of this paper), which apply to looping automata only, to the case of automata with Büchi acceptance conditions.

2 Preliminaries

In this section, we first introduce the DL $\mathcal{SI}$ and the temporal logic LTL, and then recall the relevant definitions regarding pinpointing from [5].

2.1 The Description Logic $\mathcal{SI}$

As mentioned above, $\mathcal{SI}$ extends the basic DL $\mathcal{ALC}$ with transitive and inverse roles. An example of a role that should be interpreted as transitive is has-descendant, while has-ancestor should be interpreted as the inverse of has-descendant. Instead of employing the usual approach of "hard-coding" inverse and transitive roles into the syntax and semantics of concept descriptions, we allow the use of inverse and transitivity axioms in the knowledge base. This enables us to pinpoint also these kinds of axioms as reasons for certain consequences. Thus, the concept descriptions that we consider in this case are simply $\mathcal{ALC}$ concept descriptions.

Definition 1 ($\mathcal{ALC}$ concept descriptions) Let N_C be a set of *concept names* and N_R a set of *role names. The set of $\mathcal{ALC}$ concept descriptions* is the smallest set such that

- all concept names are $\mathcal{ALC}$ concept descriptions;
- if C and D are $\mathcal{ALC}$ concept descriptions, then so are $\neg C$, $C \sqcup D$, and $C \sqcap D$;
- if C is an $\mathcal{ALC}$ concept description and $r \in N_R$, then $\exists r.C$ and $\forall r.C$ are $\mathcal{ALC}$ concept descriptions.

An *interpretation* is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where the *domain* $\Delta^{\mathcal{I}}$ is a non-empty set and $\cdot^{\mathcal{I}}$ is a function that assigns to every concept name A a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and to every role name r a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. This function is extended to $\mathcal{ALC}$ concept descriptions as follows:

- $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$, $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$, $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$;
- $(\exists r.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \text{there is a } y \in \Delta^{\mathcal{I}} \text{ with } (x, y) \in r^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$;
- $(\forall r.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \text{for all } y \in \Delta^{\mathcal{I}}, (x, y) \in r^{\mathcal{I}} \text{ implies } y \in C^{\mathcal{I}}\}$.

In this paper we restrict our attention to terminological knowledge, which is given by a so-called TBox.

Definition 2 ($\mathcal{SI}$ TBoxes) An *$\mathcal{SI}$ TBox* is a finite set of axioms of the following form:

(i) $C \sqsubseteq D$ where C and D are $\mathcal{ALC}$ concept descriptions (GCI);
(ii) $\mathsf{trans}(r)$ where $r \in N_R$ (transitivity axiom);
(iii) $\mathsf{inv}(r, s)$, where $r \neq s \in N_R$ (inverse axiom),

such that every $r \in N_R$ appears in at most one inverse axiom.

An interpretation $\mathcal{I}$ is called a *model* of the $\mathcal{SI}$ TBox $\mathcal{T}$ if it satisfies all axioms in $\mathcal{T}$, i.e., if

(i) $C \sqsubseteq D \in \mathcal{T}$ implies $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$;
(ii) $\mathsf{trans}(r) \in \mathcal{T}$ implies that $r^{\mathcal{I}}$ is transitive;
(iii) $\mathsf{inv}(r, s) \in \mathcal{T}$ implies that $(x, y) \in r^{\mathcal{I}}$ iff $(y, x) \in s^{\mathcal{I}}$.

The main inference problems for terminological knowledge are satisfiability and subsumption

Definition 3 (satisfiability, subsumption) Let C and D be $\mathcal{ALC}$ concept descriptions and $\mathcal{T}$ an $\mathcal{SI}$ TBox. We say that C is *satisfiable w.r.t.* $\mathcal{T}$ if there is a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}} \neq \emptyset$. In this case, $\mathcal{I}$ is also called a *model of* C *w.r.t.* $\mathcal{T}$. We call C *unsatisfiable w.r.t.* $\mathcal{T}$ if it does not have a model w.r.t. $\mathcal{T}$. Finally, we say that C *is subsumed* by D w.r.t. $\mathcal{T}$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds in every model $\mathcal{I}$ of $\mathcal{T}$.

We want to pinpoint reasons for unsatisfiability and for subsumption. Since C is subsumed by D w.r.t. $\mathcal{T}$ iff $C \sqcap \neg D$ is unsatisfiable w.r.t. $\mathcal{T}$, it is obviously sufficient to design a pinpointing algorithm for unsatisfiability.

The automata-based approach for deciding (un)satisfiability uses the fact that an $\mathcal{ALC}$ concept description C is satisfiable w.r.t. an $\mathcal{SI}$ TBox $\mathcal{T}$ iff it has a certain tree-shaped model, called Hintikka tree for C and $\mathcal{T}$. It constructs a looping tree automaton working on infinite trees whose runs are exactly the Hintikka trees for C and $\mathcal{T}$ (see [3] and Section 4.2), and then tests this automaton for emptiness.

2.2 Linear Temporal Logic

Linear Temporal Logic (LTL) is an extension of propositional logic that allows reasoning about temporal properties, where time is seen as discrete and linear. The semantics of this logic use the notion of a *computation*, which intuitively correspond to interpretations whose domain is fixed to be the set of natural numbers.

Definition 4 (LTL formulae) Let P be a set of propositional variables. The set of *LTL formulae* is the smallest set such that

- all propositional variables are LTL formulae,
- if ϕ and ψ are LTL formulae, then so are $\neg\phi, \phi \wedge \psi, \bigcirc\phi$, and $\phi \mathcal{U} \psi$.

A *computation* is a function $\pi : \mathbb{N} \to \mathscr{P}(\mathsf{P})$, where $\mathbb{N}$ represents the set of natural numbers. This function is extended to LTL formulae as follows, for every $i \in \mathbb{N}$:

- $\neg\phi \in \pi(i)$ iff $\phi \notin \pi(i)$; $\quad \phi \wedge \psi \in \pi(i)$ iff $\{\phi, \psi\} \subseteq \pi(i)$;
- $\bigcirc\phi \in \pi(i)$ iff $\phi \in \pi(i+1)$; and

– $\phi \mathcal{U} \psi \in \pi(i)$ iff there is a $j \geq i$ such that $\psi \in \pi(j)$ and for all $k, i \leq k < j$, it holds that $\phi \in \pi(k)$.

The LTL formula ϕ is *satisfiable* if there is a computation π such that $\phi \in \pi(0)$.

One is usually interested in deciding whether a given LTL formula is satisfiable or not. Here, we will look at the satisfiability problem in a more fine-grained manner. We are interested in detecting which parts of the formula actually cause the unsatisfiability. More precisely, we will assume that our formula is a conjunction of LTL formulae, and we want to find out which conjuncts are responsible for the unsatisfiability. We additionally allow some of these conjuncts to be *trusted* in the sense that they will never be considered as the causes for unsatisfiability. Thus, we consider LTL formulae that are conjunctions of a static formula ϕ, which must always be there, and a set of refutable formulae $\mathcal{R}$, which can be removed.

Definition 5 (axiomatic satisfiability) Let ϕ be an LTL formula and $\mathcal{R}$ a finite set of LTL formulae. We say that ϕ is *a-satisfiable* w.r.t. $\mathcal{R}$ if $\phi \wedge \bigwedge_{\psi \in \mathcal{R}} \psi$ is satisfiable, i.e., there is a computation π such that $\mathcal{R} \cup \{\phi\} \subseteq \pi(0)$. In this case, π is called a *computation for* $(\phi, \mathcal{R})$.

We will show in Section 4.3 how one can construct a Büchi tree automaton that has as its successful runs all computations for the input, thus allowing us to reduce a-satisfiability to the emptiness problem for Büchi tree automata.

2.3 Basic Definitions for Pinpointing

Following [5], we define pinpointing not for a specific logic and inference problem, but rather in a more general setting. The type of inference problems that we will consider is deciding a so-called c-property for a given set of axiomatized inputs. To obtain an intuitive understanding of the following definition, just assume that inputs are $\mathcal{ALC}$ concept descriptions, admissible sets of axioms are $\mathcal{SI}$ TBoxes, and the c-property is unsatisfiablility.

Definition 6 (axiomatized input, c-property) Let $\mathfrak{I}$ and $\mathfrak{T}$ be sets of *inputs* and *axioms*, respectively, and let $\mathscr{P}_{admis}(\mathfrak{T}) \subseteq \mathscr{P}_{fin}(\mathfrak{T})$ be a set of finite subsets of $\mathfrak{T}$ such that $\mathcal{T} \in \mathscr{P}_{admis}(\mathfrak{T})$ implies $\mathcal{T}' \in \mathscr{P}_{admis}(\mathfrak{T})$ for all $\mathcal{T}' \subseteq \mathcal{T}$. An *axiomatized input* for $\mathfrak{I}$ and $\mathscr{P}_{admis}(\mathfrak{T})$ is of the form $(\mathcal{I}, \mathcal{T})$ where $\mathcal{I} \in \mathfrak{I}$ and $\mathcal{T} \in \mathscr{P}_{admis}(\mathfrak{T})$.

A *consequence property* (or *c-property* for short) is a set $\mathcal{P} \subseteq \mathfrak{I} \times \mathscr{P}_{admis}(\mathfrak{T})$ such that $(\mathcal{I}, \mathcal{T}) \in \mathcal{P}$ implies $(\mathcal{I}, \mathcal{T}') \in \mathcal{P}$ for every $\mathcal{T}' \in \mathscr{P}_{admis}(\mathfrak{T})$ with $\mathcal{T}' \supseteq \mathcal{T}$.

The reason why we have introduced the set $\mathscr{P}_{admis}(\mathfrak{T})$ of admissible subsets of $\mathcal{T}$ (rather than taking all finite subsets of $\mathcal{T}$) is to allow us to impose additional restrictions on the sets of axioms that must be considered. For instance, $\mathcal{SI}$ TBoxes are not arbitrary finite sets of axioms of the form (i), (ii), and (iii) (see Definition 2). In addition, we require that every role name appears in at most one inverse axiom. Clearly, this restriction satisfies our requirement for admissible sets of axioms.

The problems of *un*satisfiability of $\mathcal{ALC}$ concept descriptions w.r.t. $\mathcal{SI}$ TBoxes and a-*un*satisfiability of sets of LTL formulae are indeed c-properties. More formally, let $\mathfrak{I}$

consist of all $\mathcal{ALC}$ concept descriptions, $\mathfrak{T}$ of all GCIs, transitivity axioms, and inverse axioms, and $\mathscr{P}_{admis}(\mathfrak{T})$ of all $\mathcal{SI}$ TBoxes. The following is a c-property:

$$\mathcal{P} = \{(C, \mathcal{T}) \mid C \text{ is unsatisfiable w.r.t. } \mathcal{T}\}.$$

Likewise, if $\mathfrak{I}$ and $\mathfrak{T}$ both consist of all LTL formulae and $\mathscr{P}_{admis}(\mathfrak{T}) = \mathscr{P}_{fin}(\mathfrak{T})$, then

$$\mathcal{P} = \{(\phi, \mathcal{R}) \mid \phi \text{ is a-unsatisfiable w.r.t. } \mathcal{R}\}$$

is a c-property.

Definition 7 Given an axiomatized input $\Gamma = (\mathcal{I}, \mathcal{T})$ and a c-property $\mathcal{P}$, a set of axioms $\mathcal{S} \subseteq \mathcal{T}$ is called a *minimal axiom set (MinA)* for Γ w.r.t. $\mathcal{P}$ if $(\mathcal{I}, \mathcal{S}) \in \mathcal{P}$ and $(\mathcal{I}, \mathcal{S}') \notin \mathcal{P}$ for every $\mathcal{S}' \subset \mathcal{S}$. The set of all MinAs for Γ w.r.t. $\mathcal{P}$ is denoted by $\mathsf{MIN}_{\mathcal{P}(\Gamma)}$.

Note that the notion of a MinA is only interesting if $\Gamma \in \mathcal{P}$; otherwise, the monotonicity requirement for $\mathcal{P}$ entails that $\mathsf{MIN}_{\mathcal{P}(\Gamma)} = \emptyset$. Let us instantiate this definition for the two c-properties we have introduced above.

In our $\mathcal{SI}$ example, consider the axiomatized input $\Gamma = (A \sqcap \forall r.C, \mathcal{T})$ where $\mathcal{T}$ consists of

$$\text{ax}_1\colon\ A \sqsubseteq \exists r.B, \quad \text{ax}_2\colon\ B \sqsubseteq \forall s.\neg A, \quad \text{ax}_3\colon\ C \sqsubseteq \neg B, \quad \text{ax}_4\colon\ \mathsf{inv}(r, s) \tag{1}$$

It is easy to see that $\Gamma \in \mathcal{P}$, and that the set of all MinAs for Γ is $\mathsf{MIN}_{\mathcal{P}(\Gamma)} = \{\{\text{ax}_1, \text{ax}_2, \text{ax}_4\}, \{\text{ax}_1, \text{ax}_3\}\}$.

For the logic LTL, consider the axiomatized input $\Gamma = (q, \mathcal{R})$ where $\mathcal{R}$ is given by

$$\text{ax}_1\colon\ p\mathcal{U}\neg q, \qquad \text{ax}_2\colon\ \bigcirc \neg p, \qquad \text{ax}_3\colon\ \bigcirc q, \qquad \text{ax}_4\colon\ \neg(\bigcirc q \wedge p). \tag{2}$$

The set of all MinAs for Γ is then $\mathsf{MIN}_{\mathcal{P}(\Gamma)} = \{\{\text{ax}_1, \text{ax}_2, \text{ax}_3\}, \{\text{ax}_1, \text{ax}_3, \text{ax}_4\}\}$. Thus, in the LTL formula $q \wedge p\mathcal{U}\neg q \wedge \bigcirc\neg p \wedge \bigcirc q \wedge \neg(\bigcirc q \wedge p)$, the MinAs tell us which minimal combinations of the last four conjuncts are responsible for unsatifiability in the presence of q.

One might think that pinpointing (i.e., the computation of MinAs) can only be applied in the LTL setting if the formula one is interested in is a large conjunction of small formulae. At first sight, it is not clear how a subformula ψ that does not occur as a top-level conjunct could be pinpointed as a culprit for unsatisfiability. This is, however, possible by replacing such a subformula ψ by a new propositional variable p_ψ and adding the "definition" $\Box(p_\psi \Leftrightarrow \psi)$ as a top-level conjunct to the formula obtained this way.[3]

Instead of computing all MinAs, one can also compute a pinpointing formula. To define this formula, we assume that every axiom $t \in \mathfrak{T}$ is labelled with a unique propositional variable, $\mathsf{lab}(t)$. Let $\mathsf{lab}(\mathcal{T})$ be the set of all propositional variables labelling an axiom in $\mathcal{T}$. A *monotone Boolean formula* over $\mathsf{lab}(\mathcal{T})$ is a Boolean formula using variables in $\mathsf{lab}(\mathcal{T})$ and only the connectives conjunction and disjunction. In addition, the constants $\top$ and $\bot$, which always evaluate to true and false, respectively, are monotone Boolean formulae. We identify a propositional *valuation* with the set of propositional variables that it makes true. For a valuation $\mathcal{V} \subseteq \mathsf{lab}(\mathcal{T})$, let $\mathcal{T}_\mathcal{V} = \{t \in \mathcal{T} \mid \mathsf{lab}(t) \in \mathcal{V}\}$. Recall that if $\mathcal{T} \in \mathscr{P}_{admis}(\mathfrak{T})$ then for every $\mathcal{T}' \subseteq \mathcal{T}$ it holds that $\mathcal{T}' \in \mathscr{P}_{admis}(\mathfrak{T})$. In particular this means that $\mathcal{T}_\mathcal{V} \in \mathscr{P}_{admis}(\mathfrak{T})$ for every valuation $\mathcal{V}$.

[3] Here, $\Box\theta$ is an abbreviation for $\neg(\top\mathcal{U}\neg\theta)$ and $\theta_1 \Leftrightarrow \theta_2$ is an abbreviation for $\neg(\theta_1 \wedge \neg\theta_2) \wedge \neg(\neg\theta_1 \wedge \theta_2)$.

Definition 8 (pinpointing formula) Given a c-property $\mathcal{P}$ and an axiomatized input $\Gamma = (\mathcal{I}, \mathcal{T})$, the monotone Boolean formula ϕ over $\mathsf{lab}(\mathcal{T})$ is called a *pinpointing formula* for Γ w.r.t. $\mathcal{P}$ if the following holds for every valuation $\mathcal{V} \subseteq \mathsf{lab}(\mathcal{T})$:

$$(\mathcal{I}, \mathcal{T}_\mathcal{V}) \in \mathcal{P} \quad \text{iff} \quad \mathcal{V} \text{ satisfies } \phi.$$

In our $\mathcal{SI}$ example, we can take $\mathsf{lab}(\mathcal{T}) = \{\mathrm{ax}_1, \ldots, \mathrm{ax}_4\}$ as set of propositional variables. It is easy to see that $\mathrm{ax}_1 \wedge ((\mathrm{ax}_2 \wedge \mathrm{ax}_4) \vee \mathrm{ax}_3)$ is a pinpointing formula. In the LTL example, we can take the same set of propositional variables. In this case, $\mathrm{ax}_1 \wedge \mathrm{ax}_3 \wedge (\mathrm{ax}_2 \vee \mathrm{ax}_4)$ is a pinpointing formula.

Valuations can be ordered by set inclusion. The following is an immediate consequence of the definition of a pinpointing formula [4]: if ϕ a pinpointing formula for Γ w.r.t. $\mathcal{P}$, then

$$\mathsf{MIN}_{\mathcal{P}(\Gamma)} = \{\mathcal{T}_\mathcal{V} \mid \mathcal{V} \text{ is a minimal valuation satisfying } \phi\}.$$

This shows that it is enough to design an algorithm for computing a pinpointing formula to obtain all MinAs. However, the reduction suggested by the above identity is not polynomial. One possible way to obtain $\mathsf{MIN}_{\mathcal{P}(\Gamma)}$ from ϕ is to first transform ϕ into disjunctive normal form, and then remove superfluous disjuncts. It is well-known that this can cause an exponential blow-up. This should, however, not be viewed as a disadvantage of approaches computing the pinpointing formula rather than directly $\mathsf{MIN}_{\mathcal{P}(\Gamma)}$. If such a blow-up happens, then the pinpointing formula actually yields a compact representation of all MinAs.

3 Büchi Tree Automata

In this section, we introduce both unweighted and weighted generalized Büchi tree automata. These automata receive infinite trees of a fixed arity k as inputs. For a positive integer k, we denote the set $\{1, \ldots, k\}$ by K. The nodes of our trees can be identified by words in K^* in the usual way: the root node is identified by the empty word ε, and the i-th successor of the node u is identified by ui for $1 \leq i \leq k$. In the case of labelled trees, we will refer to the labelling of the node $u \in K^*$ in the tree r by $r(u)$. We will also use $\overrightarrow{r(u)}$ to denote the tuple $\overrightarrow{r(u)} = (r(u), r(u1), \ldots, r(uk))$. An infinite tree r with labels from a set Q can be represented as a mapping $r : K^* \to Q$.

For our purpose, it is sufficient to use unlabelled infinite trees as inputs for our tree automata. For a fixed arity k, there is exactly one such tree, which we can identify with the set of its nodes, i.e., with K^*. We will also use the concept of a *path* in this tree. A path is a subset $\mathsf{p} \subseteq K^*$ such that $\varepsilon \in \mathsf{p}$ and for every $u \in p$ there is exactly one $i, 1 \leq i \leq k$ with $ui \in \mathsf{p}$.

Definition 9 (Büchi tree automaton) A *generalized Büchi tree automaton* for arity k is a tuple $(Q, \Delta, I, F_1, \ldots, F_n)$, where Q is a finite set of *states*, $\Delta \subseteq Q^{k+1}$ is the *transition relation*, $I \subseteq Q$ is the set of *initial states*, and $F_1, \ldots, F_n \subseteq Q$ are sets of *final states*. A generalized Büchi tree automaton is called *Büchi automaton* if it has only one set of final states; i.e., if $n = 1$. It is called *looping tree automaton* if $n = 0$.

A *run* of a generalized Büchi automaton on the unlabelled tree K^* is a labelled k-ary tree $r : K^* \to Q$ such that $\overrightarrow{r(u)} \in \Delta$ for all $u \in K^*$. This run is *successful* if for

every path p and every $i, 1 \leq i \leq n$, there are infinitely many nodes $u \in \mathsf{p}$ such that $r(u) \in F_i$.

The *emptiness problem* for generalized Büchi tree automata for arity k is the problem of deciding whether a given such automaton has a successful run r with $r(\varepsilon) \in I$ or not.

Let us illustrate the notions introduced in this definition on a simple Büchi automaton.

Example 1 Consider the Büchi tree automaton $\mathcal{A}^{ex} = (Q, \Delta, I, F)$ for arity 2, where

- $Q = \{q_0, q_1, q_2, q_3\}$, $I = \{q_0\}$, and $F = \{q_1, q_3\}$:
- $\Delta = \{(q_0, q_1, q_1), (q_0, q_2, q_2), (q_1, q_1, q_1), (q_2, q_2, q_2), (q_2, q_3, q_3)\}$.

This automaton has two runs that label the root with the initial state q_0: r_1, which labels all the non-root nodes with q_1, and r_2, which labels all the non-root nodes with q_2; the latter is not successful, but the former is. Thus, $\mathcal{A}^{ex}$ has r_1 as a successful run that labels the root with an initial state. The binary tree r_3 that labels the root with q_0 and all the non-root nodes with q_3 is not a run of $\mathcal{A}^{ex}$. Finally, the run r_4, which labels all nodes with q_1, is a successful run of $\mathcal{A}^{ex}$, but it does not label the root with an initial state.

Although a direct algorithm for deciding the emptiness problem for a generalized Büchi automaton is sketched in [40], in the journal version of that paper [41], the idea is simplified by presenting a reduction to the emptiness problem for Büchi automata. Our treatment of weighted automata will follow a similar approach. First, we will show how to compute the behaviour of weighted Büchi automata by an approach that is inspired by the emptiness test for Büchi automata.[4] Then, we will introduce a reduction from weighted generalized Büchi automata to weighted Büchi automata that preserves the behaviour.

We will later extend automata-based decision procedures into algorithms that compute pinpointing formulae by transforming Büchi automata into weighted Büchi automata. The weights of such automata come from a distributive lattice [18].

Definition 10 (distributive lattice) A *distributive lattice* is a partially ordered set $(S, \leq_S)$ such that infima and suprema of arbitrary finite subsets of S always exist and distribute over each other. The distributive lattice $(S, \leq_S)$ is called *finite* if its carrier set S is finite.

Any weighted automaton uses as weights only finitely many elements of the underlying distributive lattice. Since finitely generated distributive lattices are finite [18], the closure of this set under the lattice operations infimum and supremum yields a finite distributive lattice. For this reason, we will in the following assume without loss of generality that the weights of our weighted Büchi automaton come from a *finite* distributive lattice $(S, \leq_S)$.

In the following, we will often simply use the carrier set S to denote the finite distributive lattice $(S, \leq_S)$. The infimum (supremum) of a subset $T \subseteq S$ will be denoted by $\bigotimes_{t \in T} t$ ($\bigoplus_{t \in T} t$). We will often compute the infimum (supremum) $\bigotimes_{i \in I} t_i$ ($\bigoplus_{i \in I} t_i$) over an infinite set of indices I. However, the finiteness of the lattice and the

[4] This emptiness test is sketched in Section 5.1.

idempotency of the operators infimum and supremum ensure that the sets over which the operators are actually applied are finite, and hence infimum and supremum are well-defined in this case. For the infimum (supremum) of two elements, we will also use infix notation, i.e., write $t_1 \otimes t_2$ ($t_1 \oplus t_2$) to denote the infimum (supremum) of the set $\{t_1, t_2\}$. The least element of S (i.e., the infimum of the whole set S) will be denoted by $\mathbf{0}$, and the greatest element (i.e., the supremum of the whole set S) by $\mathbf{1}$.

It should be noted that our assumption that the weights come from a finite distributive lattice is stronger than the one usually encountered in the literature on weighted automata. In fact, for automata working on finite words or trees, it is sufficient to assume that the weights come from a so-called semiring [37]. In order to have a well-defined behaviour also for weighted automata working on infinite objects, the existence of infinite products and sums is required [16, 31]. As mentioned above, our finiteness assumption ensures that such infinite products and sums are actually finite. The additional properties imposed by our requirement to have a distributive lattice (in particular, distributivity and the idempotency of product and sum) are necessary for our approach of computing the behaviour of weighted Büchi automata (see Section 5). These stronger assumptions are not problematic in our pinpointing application: as we will see later, the weights we will encounter in our computation of the pinpointing formula actually come from a finitely generated free distributive lattice.

Definition 11 (weighted Büchi automaton) Let S be a finite distributive lattice. A *weighted generalized Büchi automaton (WGBA)* over S for arity k is a tuple $\mathcal{A} = (Q, in, wt, F_1, \ldots, F_n)$ where Q is a finite set of *states*, $in : Q \to S$ is the *initial distribution*, $wt : Q^{k+1} \to S$ assigns *weights* to transitions, and $F_1, \ldots, F_n \subseteq Q$ are the sets of *final states*. A WGBA is called *weighted Büchi automaton (WBA)* if $n = 1$ and *weighted looping automaton (WLA)* if $n = 0$.

A *run* of the WGBA $\mathcal{A}$ is a labelled tree $r : K^* \to Q$. The *weight* of this run is $wt(r) = \bigotimes_{u \in K^*} wt(\overrightarrow{r(u)})$. This run is *successful* if, for every path p and every $i, 1 \leq i \leq n$, there are infinitely many nodes $u \in \mathsf{p}$ such that $r(u) \in F_i$. Let $\mathsf{succ}_{\mathcal{A}}$ denote the set of all successful runs of $\mathcal{A}$. The *behaviour* of the automaton $\mathcal{A}$ is

$$\|\mathcal{A}\| := \bigoplus_{r \in \mathsf{succ}_{\mathcal{A}}} in(r(\varepsilon)) \otimes wt(r).$$

Let us illustrate this definition on the example of a WBA over the Boolean semiring that simulates an (unweighted) Büchi tree automaton.

Example 2 The Boolean semiring $\mathbb{B} = (\{0, 1\}, \wedge, \vee, 1, 0)$ is a finite distributive lattice, where the partial order is defined as $1 \leq_{\mathbb{B}} 0$. Note that we have defined 1 to be smaller than 0, and thus conjunction yields the supremum (i.e., is the "addition" $\oplus$) and disjunction yields the infimum (i.e., is the "multiplication" $\otimes$). Likewise, 1 is the least element $\mathbf{0}$, and 0 is the greatest element $\mathbf{1}$. The reason for this unorthodox definition is that this makes it easy to transform a given Büchi tree automaton $\mathcal{A} = (Q, \Delta, I, F)$ into a WBA $\mathcal{A}_w$ on $\mathbb{B}$ such that the behaviour of $\mathcal{A}_w$ is 0 iff $\mathcal{A}$ has a successful run that labels the root with an initial state. In $\mathcal{A}_w$, the initial distribution maps initial states to 0 and all other states to 1; a tuple in Q^{k+1} gets weight 0 if it belongs to Δ, and weight 1 otherwise.

Consider the WBA $\mathcal{A}_w^{ex}$ that is obtained by applying this construction to the Büchi tree automaton $\mathcal{A}^{ex}$ of Example 1. The run r_1 has weight 0 since all the transitions it uses have weight 0, and these weights are multiplied with each other, i.e., connected by

disjunction. Since this run is successful, it contributes the summand $in(q_0) \otimes wt(r_1) = 0 \vee 0 = 0$ to the behaviour of $\mathcal{A}_w^{ex}$. Since addition is conjunction, this causes the behaviour of $\mathcal{A}_w^{ex}$ to be 0. Let us nevertheless consider some other runs. The run r_2 also has weight 0 and starts with the initial state q_0. However, since this run is not successful, $in(q_0) \otimes wt(r_2)$ is not used as a summand when computing the behaviour of $\mathcal{A}_w^{ex}$. The tree r_3 is a successful run of $\mathcal{A}_w^{ex}$, but it is not a run of $\mathcal{A}^{ex}$. Since it uses the transition (q_3, q_3, q_3), whose weight is 1, its overall weight is 1 as well. Thus, it contributes the summand $in(q_0) \otimes wt(r_3) = 0 \vee 1 = 1$ to the behaviour of $\mathcal{A}_w^{ex}$, but this summand is "eaten up" by the summand 0 contributed to the sum (i.e., conjunction) by the run r_1. Finally, the run r_4, is a successful run of $\mathcal{A}_w^{ex}$, which has weight 0. Since q_1 is not an initial state of $\mathcal{A}^{ex}$, it contributes the summand $wt(q_1) \otimes wt(r_4) = 1 \vee 0 = 1$ to the behaviour of $\mathcal{A}_w^{ex}$.

By generalzing the observations we have made for the runs r_1, r_2, r_3, r_4 of $\mathcal{A}_w^{ex}$, it is easy see that the following holds for any Büchi tree automaton $\mathcal{A}$: the behaviour of $\mathcal{A}_w$ is 0 iff $\mathcal{A}$ has a successful run that labels the root with an initial state.

In Section 5, we will develop an approach for computing the behaviour of weighted (generalized) Büchi tree automata that generalizes the emptiness test for (generalized) Büchi tree automata. But first, we show how to reduce the problem of computing the pinpointing formula to the problem of computing the behaviour of a WGBA.

4 Automata-based Pinpointing

In this section, we first introduce our general approach for automata-based pinpointing, and then show how it can be applied to finding a pinpointing formula for unsatisfiability in $\mathcal{SI}$ and LTL.

4.1 The General Approach

Basically, the automata-based approach for deciding a c-property $\mathcal{P}$ takes axiomatized inputs $\Gamma = (\mathcal{I}, \mathcal{T})$ and translates them into automata $\mathcal{A}_\Gamma$ such that $\Gamma \in \mathcal{P}$ iff $\mathcal{A}_\Gamma$ does *not* have a successful run. For example, the automaton constructed from a concept description C and a TBox $\mathcal{T}$ has a successful run iff C is satisfiable w.r.t. $\mathcal{T}$, where the c-property is *un*satisfiability. If the translation from Γ to $\mathcal{A}_\Gamma$ is an arbitrary function, then we have no way of knowing how the axioms in $\mathcal{T}$ influence the behaviour of the automaton, and thus it is not clear how to construct a corresponding pinpointing automaton. For this reason, we will assume that the automaton $\mathcal{A}_\Gamma$ for $\Gamma = (\mathcal{I}, \mathcal{T})$ in a certain sense also contains automata for all axiomatized inputs $(\mathcal{I}, \mathcal{T}')$ with $\mathcal{T}' \subseteq \mathcal{T}$,[5] which can be obtained by appropriately restricting the states and transitions of $\mathcal{A}_\Gamma$. To be more precise, let $\mathcal{A} = (Q, \Delta, I, F_1, \ldots, F_n)$ be a generalized Büchi automaton for arity k and $\Gamma = (\mathcal{I}, \mathcal{T})$ an axiomatized input. The functions $\Delta\mathsf{res} : \mathcal{T} \to \mathscr{P}(Q^{k+1})$ and $I\mathsf{res} : \mathcal{T} \to \mathscr{P}(Q)$ are respectively called a *transition restricting function* and an *initial restricting function*. The restricting functions $\Delta\mathsf{res}$ and $I\mathsf{res}$ can be extended to sets of axioms $\mathcal{T}' \subseteq \mathcal{T}$ as follows:

$$\Delta\mathsf{res}(\mathcal{T}') := \bigcap_{t \in \mathcal{T}'} \Delta\mathsf{res}(t) \quad \text{and} \quad I\mathsf{res}(\mathcal{T}') := \bigcap_{t \in \mathcal{T}'} I\mathsf{res}(t).$$

[5] Recall that every subset of an admissible set of axioms is also admissible.

For $\mathcal{T}' \subseteq \mathcal{T}$, the *$\mathcal{T}'$-restricted subautomaton of $\mathcal{A}$ w.r.t. $\Delta\mathsf{res}$ and $I\mathsf{res}$* is defined as

$$\mathcal{A}_{|\mathcal{T}'} := (Q, \Delta \cap \Delta\mathsf{res}(\mathcal{T}'), I \cap I\mathsf{res}(\mathcal{T}'), F_1, \ldots, F_n).$$

Definition 12 (axiomatic automaton) Let $\mathcal{A} = (Q, \Delta, I, F_1, \ldots, F_n)$ be a generalized Büchi automaton for arity k, $\Gamma = (\mathcal{I}, \mathcal{T})$ an axiomatized input, and $\Delta\mathsf{res} : \mathcal{T} \to \mathscr{P}(Q^{k+1})$ and $I\mathsf{res} : \mathcal{T} \to \mathscr{P}(Q)$ a transition and an initial restricting function, respectively. Then we call $(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})$ an *axiomatic automaton* for Γ.

Given a c-property $\mathcal{P}$, we say that $(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})$ is *correct* for Γ w.r.t. $\mathcal{P}$ if the following holds for every $\mathcal{T}' \subseteq \mathcal{T}$: $(\mathcal{I}, \mathcal{T}') \in \mathcal{P}$ iff $\mathcal{A}_{|\mathcal{T}'}$ does not have a successful run r with $r(\varepsilon) \in I \cap I\mathsf{res}(\mathcal{T}')$.

Given a correct axiomatic automaton for $\Gamma = (\mathcal{I}, \mathcal{T})$, we can decide $(\mathcal{I}, \mathcal{T}') \in \mathcal{P}$ for $\mathcal{T}' \subseteq \mathcal{T}$ by applying the emptiness test for generalized Büchi automata to $\mathcal{A}_{|\mathcal{T}'}$.

Example 3 Let $\Gamma = (\mathcal{I}, \mathcal{T})$ be an axiomatized input, where $\mathcal{T} = \{\mathrm{ax}_1, \mathrm{ax}_2, \mathrm{ax}_3\}$, and assume that, for all $\mathcal{T}' \subseteq \mathcal{T}$, the c-property $\mathcal{P}$ holds for $(\mathcal{I}, \mathcal{T}')$ iff $\{\mathrm{ax}_1, \mathrm{ax}_2\} \cap \mathcal{T}' \neq \emptyset$. Thus, $\mathsf{MIN}_{\mathcal{P}(\Gamma)} = \{\{\mathrm{ax}_1\}, \{\mathrm{ax}_2\}\}$, and $\mathrm{ax}_1 \vee \mathrm{ax}_2$ is a pinpointing formula.

Consider the axiomatic automaton $(\mathcal{A}^{ex}, \Delta\mathsf{res}, I\mathsf{res})$, where

- $\mathcal{A}^{ex}$ is the Büchi tree automaton introduced in Example 1;
- the transition restricting function is defined as $\Delta\mathsf{res}(\mathrm{ax}_1) = \Delta \setminus \{(q_1, q_1, q_1)\}$, $\Delta\mathsf{res}(\mathrm{ax}_2) = \Delta$, and $\Delta\mathsf{res}(\mathrm{ax}_3) = \Delta \setminus \{(q_2, q_2, q_2)\}$;
- the initial restricting function is defined as $I\mathsf{res}(\mathrm{ax}_1) = I$, $I\mathsf{res}(\mathrm{ax}_2) = \emptyset$, and $I\mathsf{res}(\mathrm{ax}_3) = I$.

It is easy to see that $(\mathcal{A}^{ex}, \Delta\mathsf{res}, I\mathsf{res})$ is correct for Γ w.r.t. $\mathcal{P}$. In fact, recall that the only successful run of $\mathcal{A}^{ex}$ is r_1, which labels the root with q_0 and all non-root nodes with q_1. Now, assume that $\mathcal{T}' \subseteq \mathcal{T}$. If $\mathrm{ax}_1 \in \mathcal{T}'$, then the transition (q_1, q_1, q_1), which is used in the run r_1, is no longer available, and thus r_1 is not a run of $\mathcal{A}_{|\mathcal{T}'}$. If $\mathrm{ax}_2 \in \mathcal{T}'$, then $\mathcal{A}_{|\mathcal{T}'}$ does not have an initial state, and thus r_1 no longer starts with an initial state. Finally, having ax_3 in $\mathcal{T}'$ does not remove the run r_1 since this axiom only removes the transition (q_2, q_2, q_2), which is not used in r_1, and it also does not change the set of initial states. Consequently, we have seen that $\mathcal{A}_{|\mathcal{T}'}$ does not have a run that labels the root with an initial state iff $\{\mathrm{ax}_1, \mathrm{ax}_2\} \cap \mathcal{T}' \neq \emptyset$, and thus iff $\mathcal{P}$ holds for $(\mathcal{I}, \mathcal{T}')$.

Now, we show how to transform a correct axiomatic automaton into a weighted generalized Büchi automaton whose behaviour is a pinpointing formula for the input. This weighted automaton uses the $\mathcal{T}$-Boolean semiring, which is defined as $\mathbb{B}^{\mathcal{T}} := (\hat{\mathbb{B}}(\mathcal{T}), \wedge, \vee, \top, \bot)$, where $\hat{\mathbb{B}}(\mathcal{T})$ is the quotient set of all monotone Boolean formulae over $\mathsf{lab}(\mathcal{T})$ by the propositional equivalence relation, i.e., two propositionally equivalent formulae correspond to the same element of $\hat{\mathbb{B}}(\mathcal{T})$. It is easy to see that this semiring is indeed a distributive lattice, where the partial order is defined as $\phi \leq \psi$ iff $\psi \to \phi$ is valid. Furthermore, as $\mathcal{T}$ is finite, this lattice is also finite.[6] Note that, similar to the case of the Boolean semiring $\mathbb{B}$, conjunction is the semiring addition (i.e., yields the supremum $\oplus$) and disjunction is the semiring multiplication (i.e., yields the infimum $\otimes$). Likewise, $\top$ is the least element $\mathbf{0}$ and $\bot$ is the greatest element $\mathbf{1}$.

[6] More precisely, $\mathbb{B}^{\mathcal{T}}$ is the free distributive lattice over the generators $\mathsf{lab}(\mathcal{T})$.

Definition 13 (pinpointing automaton) Let $(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})$ be an axiomatic automaton for $\Gamma = (\mathcal{I}, \mathcal{T})$, with $\mathcal{A} = (Q, \Delta, I, F_1, \ldots, F_n)$. The *violating functions* $\Delta\mathsf{vio} : Q^{k+1} \to \mathbb{B}^{\mathcal{T}}$ and $I\mathsf{vio} : Q \to \mathbb{B}^{\mathcal{T}}$ are given by

$$\Delta\mathsf{vio}(q_0, q_1, \ldots, q_k) := \bigvee_{\{t \in \mathcal{T} \mid (q_0, q_1, \ldots, q_k) \notin \Delta\mathsf{res}(t)\}} \mathsf{lab}(t);$$

$$I\mathsf{vio}(q) := \bigvee_{\{t \in \mathcal{T} \mid q \notin I\mathsf{res}(t)\}} \mathsf{lab}(t),$$

where the empty disjunction yields $\bot$.

The *pinpointing automaton* induced by $(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})$ w.r.t. $\mathcal{T}$ is the WGBA over $\mathbb{B}^{\mathcal{T}}$ $(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})^{\mathsf{pin}} = (Q, in, wt, F_1, \ldots, F_n)$, where

$$in(q) := \begin{cases} I\mathsf{vio}(q) & \text{if } q \in I, \\ \top & \text{otherwise;} \end{cases}$$

$$wt(q_0, q_1, \ldots, q_k) := \begin{cases} \Delta\mathsf{vio}(q_0, q_1, \ldots, q_k) & \text{if } (q_0, q_1, \ldots, q_k) \in \Delta, \\ \top & \text{otherwise.} \end{cases}$$

It is easy to see that, if $r : K^* \to Q$ is a run of $\mathcal{A}$, then its weight is given by $wt(r) = \bigvee_{u \in K^*} \Delta\mathsf{vio}(\overrightarrow{r(u)})$; otherwise, $wt(r) = \top$. Intuitively, the violating function $\Delta\mathsf{vio}$ expresses which axioms are not "satisfied" by a given transition, and thus the weight of a run accumulates all the axioms violated by any of the transitions appearing as labels in it. Additionally, the function $I\mathsf{vio}$ represents the axioms that are violated by the initial state of this run. Removing all the axioms appearing in these two formulae would yield a subset of axioms which actually allows for this run; and hence, if the run is successful and the root is labelled with an initial state, due to correctness, the property does not hold anymore. Conjoining this information for all possible successful runs leads us to a pinpointing formula.

Before formulating and proving this fact more formally, let us illustrate the construction of the pinpointing automaton on the axiomatic automaton introduced in Example 3.

Example 4 Let $(\mathcal{A}^{ex}, \Delta\mathsf{res}, I\mathsf{res})$ be the axiomatic automaton from Example 3. The corresponding pinpointing automaton has the initial distribution in, where

$$in(q_0) = \mathrm{ax}_2 \quad \text{and} \quad in(q_1) = in(q_2) = in(q_3) = \top,$$

and the weight function wt, where

$$\begin{aligned} &wt(q_1, q_1, q_1) = \mathrm{ax}_1 \quad \text{and} \quad wt(q_2, q_2, q_2) = \mathrm{ax}_3, \\ &wt(q, q', q'') = \bot \quad \text{if} \quad (q, q', q'') \in \Delta \setminus \{(q_1, q_1, q_1), (q_2, q_2, q_2)\}, \\ &wt(q, q', q'') = \top \quad \text{if} \quad (q, q', q'') \notin \Delta. \end{aligned}$$

The behaviour of this WBA is $\|(\mathcal{A}^{ex}, \Delta\mathsf{res}, I\mathsf{res})^{\mathsf{pin}}\| = \bigwedge_{r \in \mathsf{succ}_{\mathcal{A}^{ex}}} in(r(\varepsilon)) \vee wt(r)$. Obviously, only successful runs that label the root with q_0 can contribute a conjunct different from $\top$ to this conjunction. There is a single successful run of $\mathcal{A}^{ex}$ that satisfies this restriction: the run r_1, which labels the root with q_0 and all other nodes with q_1. The weight of this run is $wt(r_1) = wt(q_0, q_1, q_1) \vee wt(q_1, q_1, q_1) = \bot \vee \mathrm{ax}_1 = \mathrm{ax}_1$. Since $in(q_0) = \mathrm{ax}_2$, this shows that $\|(\mathcal{A}^{ex}, \Delta\mathsf{res}, I\mathsf{res})^{\mathsf{pin}}\| = \mathrm{ax}_2 \vee \mathrm{ax}_1$, which is a pinpointing formula for Γ w.r.t. $\mathcal{P}$ (see Example 3).

Theorem 1 *Let $\mathcal{P}$ be a c-property, and $\Gamma = (\mathcal{I}, \mathcal{T})$ an axiomatized input. If the axiomatic automaton $(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})$ is correct for Γ w.r.t. $\mathcal{P}$, then $\|(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})^{\mathsf{pin}}\|$ is a pinpointing formula for Γ w.r.t. $\mathcal{P}$.*

Proof We need to show that, for every valuation $\mathcal{V} \subseteq \mathsf{lab}(\mathcal{T})$, it holds that $\mathcal{V}$ satisfies $\|(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})^{\mathsf{pin}}\|$ iff $(\mathcal{I}, \mathcal{T}_\mathcal{V}) \in \mathcal{P}$. Let $\mathcal{V} \subseteq \mathsf{lab}(\mathcal{T})$. Suppose first that $(\mathcal{I}, \mathcal{T}_\mathcal{V}) \notin \mathcal{P}$. Since $(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})$ is correct for Γ w.r.t. $\mathcal{P}$, there must be a successful run r of $\mathcal{A}_{|\mathcal{T}_\mathcal{V}}$ with $r(\varepsilon) \in I \cap I\mathsf{res}(\mathcal{T}_\mathcal{V})$. Consequently, $\overrightarrow{r(u)} \in \Delta\mathsf{res}(\mathcal{T}_\mathcal{V})$ holds for every $u \in K^*$, and thus $\mathcal{V}$ cannot satisfy $\Delta\mathsf{vio}(\overrightarrow{r(u)})$, for any $u \in K^*$. Since r is a successful run of $\mathcal{A}_{|\mathcal{T}_\mathcal{V}}$, it is also a successful run of $\mathcal{A}$, which implies $wt(r) = \bigvee_{u \in K^*} \Delta\mathsf{vio}(\overrightarrow{r(u)})$. Thus, $\mathcal{V}$ does not satisfy $wt(r)$. Since $r(\varepsilon) \in I$, we know that $in(r(\varepsilon)) = I\mathsf{vio}(r(\varepsilon))$; additionally, $r(\varepsilon) \in I\mathsf{res}(\mathcal{T}_\mathcal{V})$ implies that $\mathcal{V}$ does not satisfy $I\mathsf{vio}(r(\varepsilon))$. Thus, $\mathcal{V}$ does not satisfy $in(r(\varepsilon)) \vee wt(r)$. But then $\mathcal{V}$ also cannot satisfy $\bigwedge_{r \in \mathsf{succ}_\mathcal{A}} in(r(\varepsilon)) \vee wt(r) = \|(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})^{\mathsf{pin}}\|$.

Conversely, if $\mathcal{V}$ does not satisfy $\|(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})^{\mathsf{pin}}\| = \bigwedge_{r \in \mathsf{succ}_\mathcal{A}} in(r(\varepsilon)) \vee wt(r)$, then there must exist a successful run r such that $\mathcal{V}$ does not satisfy $in(r(\varepsilon)) \vee wt(r)$. This implies that $r(\varepsilon) \in I \cap I\mathsf{res}(\mathcal{T}_\mathcal{V})$ and that $\overrightarrow{r(u)} \in \Delta\mathsf{res}(\mathcal{T}_\mathcal{V})$ for all $u \in K^*$. Consequently, r is a successful run of $\mathcal{A}_{|\mathcal{T}_\mathcal{V}}$ with $r(\varepsilon) \in I \cap I\mathsf{res}(\mathcal{T}_\mathcal{V})$, which shows $(\mathcal{I}, \mathcal{T}_\mathcal{V}) \notin \mathcal{P}$, by the correctness of the axiomatic automaton. □

4.2 Constructing Axiomatic Automata for $\mathcal{SI}$

If we want to apply Theorem 1 to obtain an automata-based approach for pinpointing unsatisfiability in $\mathcal{SI}$, we must show how, given an $\mathcal{ALC}$ concept description C and an $\mathcal{SI}$ TBox $\mathcal{T}$, we can construct an axiomatic automaton $(\mathcal{A}_{C,\mathcal{T}}, \Delta\mathsf{res}_{C,\mathcal{T}}, I\mathsf{res}_{C,\mathcal{T}})$ that is correct for $(C, \mathcal{T})$ w.r.t. unsatisfiability. For this purpose, we must adapt the known construction of a looping automaton for $\mathcal{SI}$ from [3] such that it yields an axiomatic automaton.[7]

As mentioned before, the automata-based approach for deciding (un)satisfiability uses the fact that a concept is satisfiable iff it has a so-called Hintikka tree. The automaton to be constructed will have exactly these Hintikka trees as its runs. Intuitively, Hintikka trees are obtained from tree-shaped models by labelling every node with the "relevant" concept descriptions to which it belongs.

Following [3], we assume that all concept descriptions are in *negation normal form* (NNF), i.e., negation appears only directly in front of concept names. Any $\mathcal{ALC}$ concept description can be transformed into NNF in linear time using de Morgan, duality of quantifiers, and elimination of double negations. We denote the NNF of C by $\mathsf{nnf}(C)$ and $\mathsf{nnf}(\neg C)$ by $\backsim C$. Given an $\mathcal{ALC}$ concept description C and an $\mathcal{SI}$ TBox $\mathcal{T}$, the set of relevant concept descriptions is the set of all subdescriptions of C and of the concept descriptions $\backsim D \sqcup E$ for $D \sqsubseteq E \in \mathcal{T}$. We denote this set by $\mathsf{sub}(C, \mathcal{T})$. The set of role names occurring in C or $\mathcal{T}$ is denoted by $\mathsf{rol}(C, \mathcal{T})$. The states of our automaton are so-called Hintikka sets, which in addition to subdescriptions also contain information about which roles are supposed to be transitive.

[7] On the one hand, the construction in [3] is more complex than the one given here since the states of the automata in [3] contain additional information needed for detecting cycles in a run as early as possible, which is not relevant for the present paper. On the other hand, the states of the automata constructed here contain additional information about transitivity needed for defining the restricting function.

Definition 14 (Hintikka set) A set $H \subseteq \mathsf{sub}(C,\mathcal{T}) \cup \mathsf{ro}|(C,\mathcal{T})$ is called a *Hintikka set* for $(C,\mathcal{T})$ if the following three conditions are satisfied:

(i) if $D \sqcap E \in H$, then $\{D, E\} \subseteq H$;
(ii) if $D \sqcup E \in H$, then $\{D, E\} \cap H \neq \emptyset$; and
(iii) there is no concept name A such that $\{A, \neg A\} \subseteq H$.

The Hintikka set H is *compatible with the GCI* $D \sqsubseteq E \in \mathcal{T}$ if it is the empty set or contains $\backsim D \sqcup E$. It is *compatible with the transitivity axiom* $\mathsf{trans}(r) \in \mathcal{T}$ if it is the empty set or contains r. Finally, it is *compatible with the inverse axiom* $\mathsf{inv}(r,s) \in \mathcal{T}$ if $r \in H$ implies $s \in H$ and vice versa.

The arity k of our automaton is determined by the number of existential restrictions, i.e., concept descriptions of the form $\exists r.D$, contained in $\mathsf{sub}(C,\mathcal{T})$. Since we need to know which successor in the tree corresponds to which existential restriction, we fix an arbitrary bijection $\varphi : \{\exists r.D \mid \exists r.D \in \mathsf{sub}(C,\mathcal{T})\} \to K$. To obtain full k-ary trees, we will use nodes labelled with the empty set (which is a Hintikka set) as dummy nodes. The following Hintikka conditions will be used to define the transitions of our automaton.

Definition 15 (Hintikka condition) The tuple of Hintikka sets $(H_0, H_1, \ldots, H_k)$ for $(C,\mathcal{T})$ satisfies the *Hintikka condition* if the following holds for every existential restriction $\exists r.D \in \mathsf{sub}(C,\mathcal{T})$:

- If $\exists r.D \in H_0$, then $H_{\varphi(\exists r.D)}$ contains D as well as every E for which there is a value restriction $\forall r.E \in H_0$; if, in addition, $r \in H_0$, then $\forall r.E$ belongs to $H_{\varphi(\exists r.D)}$ for every value restriction $\forall r.E \in H_0$.
- If $\exists r.D \notin H_0$, then $H_{\varphi(\exists r.D)} = \emptyset$.

This tuple is *compatible with the GCI* $D \sqsubseteq E \in \mathcal{T}$ (*compatible with the transitivity axiom* $\mathsf{trans}(r) \in \mathcal{T}$) if all its components are compatible with $D \sqsubseteq E$ ($\mathsf{trans}(r)$). It is *compatible with the inverse axiom* $\mathsf{inv}(r,s) \in \mathcal{T}$ if all its components are compatible with $\mathsf{inv}(r,s)$, and the following holds for all $t \in \{r,s\}$ and $t^- \in \{r,s\} \setminus \{t\}$: for every $\forall t.F \in H_{\varphi(\exists t^-.D)}$, the set H_0 contains F, and additionally $\forall t^-.F$ if $t \in H_0$.

We are now ready to define the axiomatic automaton for unsatisfiability in $\mathcal{SI}$.

Definition 16 (axiomatic automaton for $\mathcal{SI}$) Let C be an $\mathcal{ALC}$ concept description, $\mathcal{T}$ an $\mathcal{SI}$ TBox, and k the number of existential restrictions in $\mathsf{sub}(C,\mathcal{T})$. The axiomatic automaton $(\mathcal{A}_{C,\mathcal{T}}, \Delta\mathsf{res}_{C,\mathcal{T}}, I\mathsf{res}_{C,\mathcal{T}})$ has as its first component the looping automaton $\mathcal{A}_{C,\mathcal{T}} := (Q, \Delta, I)$, where

- Q consists of all Hintikka sets for $(C,\mathcal{T})$;
- Δ consists of all $(H_0, H_1, \ldots, H_k) \in Q^{k+1}$ that satisfy the Hintikka condition;
- $I := \{H \in Q \mid C \in H\}$.

The transition restricting function $\Delta\mathsf{res}_{C,\mathcal{T}}$ maps each axiom $t \in \mathcal{T}$ to the set of all tuples in Δ that are compatible with t. The initial restricting function $I\mathsf{res}_{C,\mathcal{T}}$ maps each axiom $t \in \mathcal{T}$ to the set Q, i.e., there is effectively no restriction on the initial states imposed by the axioms.

Correctness of this automaton construction can be shown by an easy adaptation of the arguments used in [3].

Theorem 2 *Let C be an $\mathcal{ALC}$ concept description and $\mathcal{T}$ an $\mathcal{SI}$ TBox. The axiomatic automaton $(\mathcal{A}_{C,\mathcal{T}}, \Delta\mathsf{res}_{C,\mathcal{T}}, I\mathsf{res}_{C,\mathcal{T}})$ is correct for $(C,\mathcal{T})$ w.r.t. unsatisfiability.*

Theorem 1 shows that it is enough to compute the behaviour of the pinpointing automaton $(\mathcal{A}_{C,\mathcal{T}}, \Delta\mathsf{res}_{C,\mathcal{T}}, I\mathsf{res}_{C,\mathcal{T}})^{\mathsf{pin}}$ induced by $(\mathcal{A}_{C,\mathcal{T}}, \Delta\mathsf{res}_{C,\mathcal{T}}, I\mathsf{res}_{C,\mathcal{T}})$ in order to obtain a pinpointing formula for $(C,\mathcal{T})$ w.r.t. unsatisfiability. In Section 5, we will show how this behaviour can be computed, but first we present an example of an axiomatic automaton where the use of a Büchi acceptance condition is necessary.

4.3 Constructing Axiomatic Automata for LTL

The axiomatic automaton for LTL a-unsatisfiability will have as states sets of formulae similar to the Hintikka sets introduced for $\mathcal{SI}$, but they will need to satisfy slightly different conditions, due to the fact that we will not assume that the formulae used are in negation normal form.[8] Given an LTL formula ϕ and a set of LTL formulae $\mathcal{R}$, the *closure of* $(\phi,\mathcal{R})$ is the set of all subformulae of ϕ and $\mathcal{R}$, and their negations, where double negations are cancelled. We denote this set as $\mathsf{cl}(\phi,\mathcal{R})$.

Following [42], the states of our automaton are *elementary* sets of formulae, which play the role of the Hintikka sets of the previous subsection. Elementary sets are maximal and consistent sets of subformulae in $\mathsf{cl}(\phi,\mathcal{R})$.

Definition 17 (elementary set) The set $H \subseteq \mathsf{cl}(\phi,\mathcal{R})$ is called an *elementary set* for $(\phi,\mathcal{R})$ if it satisfies the following conditions:

- $\neg\phi \in H$ iff $\phi \notin H$, for all $\neg\phi \in \mathsf{cl}(\phi,\mathcal{R})$;
- $\phi \wedge \psi \in H$ iff $\{\phi,\psi\} \subseteq H$, for all $\phi \wedge \psi \in \mathsf{cl}(\phi,\mathcal{R})$;
- $\psi \in H$ implies $\phi\mathcal{U}\psi \in H$, for all $\phi\mathcal{U}\psi \in \mathsf{cl}(\phi,\mathcal{R})$;
- if $\phi\mathcal{U}\psi \in H$ and $\psi \notin H$, then $\phi \in H$

The automaton constructed from a given input $(\phi,\mathcal{R})$ takes unary trees as input, i.e., its runs are infinite *words* over the set of states. The transition relation is thus binary. It plays the role of the Hintikka condition, ensuring that temporal restrictions are transfered to successor nodes when necessary.

Definition 18 (compatible) A tuple (H,H') of elementary sets is called *compatible* if it satisfies the following conditions:

- for all $\bigcirc\psi \in \mathsf{cl}(\phi,\mathcal{R})$, $\bigcirc\psi \in H$ iff $\psi \in H'$; and
- for all $\theta\mathcal{U}\psi \in \mathsf{cl}(\phi,\mathcal{R})$, $\theta\mathcal{U}\psi \in H$ iff either (i) $\psi \in H$ or (ii) $\theta \in H$ and $\theta\mathcal{U}\psi \in H'$.

The runs of our automaton will be sequences of elementary sets where each two consecutive ones form a compatible tuple. In contrast to the case for $\mathcal{SI}$, the presence of a run of this automaton does not imply the existence of a computation. The reason is that one can delay the satisfaction of an *until* formula indefinitely; that is, every node in the run may contain the formula $\theta\mathcal{U}\psi$ while none contains ψ, violating this way the last condition in the definition of a computation for the input. In order to rule out these kinds of runs and make sure that each until formula is eventually satisfied, we will impose a generalized Büchi condition, which introduces a set of final states for each until formula in $\mathsf{cl}(\phi,\mathcal{R})$.

[8] Although it is possible to transform LTL formulae into negation normal form, we decided not to do this in order to stay as close as possible to the known automaton construction for LTL [42]. This allows us to reuse the proof of correctness of this construction.

Definition 19 (axiomatic automaton for LTL) Let ϕ and $\mathcal{R}$ be an LTL formula and a set of LTL formulae, respectively, and let $\theta_1 \mathcal{U} \psi_1, \ldots, \theta_n \mathcal{U} \psi_n$ be all the until formulae in $\mathsf{cl}(\phi, \mathcal{R})$. The axiomatic automaton $(\mathcal{A}_{\phi,\mathcal{R}}, \Delta\mathsf{res}_{\phi,\mathcal{R}}, I\mathsf{res}_{\phi,\mathcal{R}})$ has as its first component the generalized Büchi automaton $\mathcal{A}_{\phi,\mathcal{R}} := (Q, \Delta, I, F_1, \ldots, F_n)$,[9] where

- Q is the set of all elementary sets for $(\phi, \mathcal{R})$;
- Δ consists of all compatible pairs $(H, H') \in Q \times Q$;
- $I := \{H \in Q \mid \phi \in H\}$;
- $F_i := \{H \in Q \mid \psi_i \in H \text{ or } \theta_i \mathcal{U} \psi_i \notin H\}$.

For every $\psi \in \mathcal{R}$, the transition restricting and initial restricting functions are given by $\Delta\mathsf{res}_{\phi,\mathcal{R}}(\psi) := \Delta$ and $I\mathsf{res}_{\phi,\mathcal{R}}(\psi) := \{H \in Q \mid \psi \in H\}$, respectively.

Correctness of this automaton can be shown by a simple adaptation of the proof in [42].

Theorem 3 *Let ϕ be an LTL formula and $\mathcal{R}$ a set of LTL formulae. The axiomatic automaton $(\mathcal{A}_{\phi,\mathcal{R}}, \Delta\mathsf{res}_{\phi,\mathcal{R}}, I\mathsf{res}_{\phi,\mathcal{R}})$ is correct for $(\phi, \mathcal{R})$ w.r.t. a-unsatisfiability.*

From Theorem 1 we know that it suffices to compute the behaviour of the pinpointing automaton $(\mathcal{A}_{\phi,\mathcal{R}}, \Delta\mathsf{res}_{\phi,\mathcal{R}}, I\mathsf{res}_{\phi,\mathcal{R}})^{\mathsf{pin}}$ induced by $(\mathcal{A}_{\phi,\mathcal{R}}, \Delta\mathsf{res}_{\phi,\mathcal{R}}, I\mathsf{res}_{\phi,\mathcal{R}})$ in order to obtain a pinpointing formula for $(\phi, \mathcal{R})$ w.r.t. a-unsatisfiability. We will show now how this behaviour can be computed.

5 Computing the Behaviour of Weighted Tree Automata

In this section, we first show how the behaviour of a weighted Büchi automaton (WBA) on a finite distributive lattice can be computed by two nested iterations. Then, we describe how this approach can be simplified to a single "bottom-up" iteration for the special case of a weighted looping automaton (WLA). Next, we show that any weighted generalized Büchi automaton (WGBA) can be reduced, in polynomial time, to a WBA that has the same behaviour. This reduction follows the ideas that have previously been used for the case of unweighted automata [41]. Finally, we compare our approach for computing the behaviour of a weighted Büchi automaton with the one independently developed in [15].

5.1 Computing the Behaviour of a WBA

Clearly, the naïve approach that directly uses the definition of the behaviour by first computing and then adding up the weights of all successful runs would not produce a result in finite time since there are potentially infinitely many successful runs, which are themselves infinite. Instead, we will use an iterative method for computing the behaviour, which generalizes the emptiness test for Büchi automata

[9] If $n = 0$, i.e., ϕ and $\mathcal{R}$ do not contain until formulae, then this automaton is actually a looping automaton.

The Emptiness Test for Büchi Automata

The emptiness problem for Büchi automata can be decided in time polynomial in the size of the automaton [30, 41]. The decision procedure constructs the set of all states that cannot occur as labels in any successful run; we will call these states *bad states*. We can try to disprove that a state is bad by trying to construct a finite partial run where every path ends in a final state.[10] Every state for which this construction fails is clearly bad, but there may be bad states for which this construction succeeds. The reason is that some of the final states reached by the finite run may themselves be bad. Thus, in order to compute all bad states we must iterate this process, where in the next iteration the partial run is required to reach final states that are not already known to be bad. Notice, however, that the construction of a finite partial run ending in non-bad final states can itself be realized by an iterative procedure. Hence, the decision procedure for the emptiness problem uses two nested iterations. In the inner loop, we try to construct a finite partial run finishing in (non-bad) final states for every state. In the outer loop, we use the result of the inner iteration to update the set of (known) bad states, and then re-start the inner iteration with this new information. Let us call the states for which there is a finite partial run finishing in non-bad final states *adequate*. First, any state $q \in Q$ for which there is a transition leading to only non-bad final states is clearly adequate. Then, every state for which there is a transition leading only to states that are either (i) final and not bad or (ii) already known to be adequate is also adequate. Obviously, during this iteration, the set of adequate states becomes stable after at most $|Q|$ iterations. The outer loop then adds all the states that were found not to be adequate to the set of bad states. The set of bad states maintained in this outer iteration becomes stable after at most $|Q|$ steps. It can be shown that there is a successful run that starts with an initial state iff not all initial states are contained in the set of bad states computed this way. This yields an emptiness test that runs in time polynomial in the number of states (see [41] for details).

Example 5 Let us illustrate this approach on the Büchi automaton $\mathcal{A}^{ex}$ of Example 1. First, we try to construct, for every state, a finite partial run where every path ends in a final state. This is possible for q_0, q_1, and q_2, but not for q_3. Thus, in this iteration, q_0, q_1, q_2 are the adequate states, and q_3 is not adequate, which means that q_3 is added to the set of bad states. In the next iteration, q_2 turns out to be no longer adequate since it can only reach the bad final state q_3. Thus, it is also put into the set of bad states. After that, the process becomes stable, i.e., the set $\{q_2, q_3\}$ is the set of bad states computed by the algorithm. Since the initial state q_0 does not belong to this set, we know that there is a successful run that starts with this initial state.

Emptiness Test by Behaviour Computation

Before treating the general case of a WBA, we consider the special case of a weighted automaton over the Boolean semiring that simulates an unweighted one. In Example 2, we have defined, for every Büchi tree automaton $\mathcal{A}$ a WBA $\mathcal{A}_w$ such that the behaviour of $\mathcal{A}_w$ is 0 iff $\mathcal{A}$ has a successful run that labels the root with an initial state. In this case, the computation of the behaviour of $\mathcal{A}_w$ basically coincides with the emptiness test applied to $\mathcal{A}$.

[10] See Definition 20 below for a formal definition of this notion.

In fact, the emptiness test for Büchi automata sketched above can be adapted such that it computes the behaviour of $\mathcal{A}_w$ as follows. We construct a function $\mathsf{bad} : Q \to \{0,1\}$ such that $\mathsf{bad}(q) = 1$ iff q is a bad state. The outer iteration of the algorithm will update this function at every step. In the beginning, no state is known to be bad, and thus we start the iteration with $\mathsf{bad}_0(q) = 0$ for all $q \in Q$. Now assume that the function $\mathsf{bad}_i : Q \to \{0,1\}$ for $i \geq 0$ has already been computed. For the next step of the iteration, we call the inner loop to update the set of adequate states. In this loop, we are going to compute the function $\mathsf{adq}^i : Q \to \{0,1\}$. Here, $\mathsf{adq}^i(q) = 1$ means that q is *not* an adequate state, i.e., that it is not possible to construct a run starting with this state where each path reaches at least one non-bad final state. At the beginning we know nothing about the adequate states, so we set $\mathsf{adq}^i_0(q) = 1$ for all $q \in Q$. Assume that we have already computed $\mathsf{adq}^i_n : Q \to \{0,1\}$. To know whether a state should become adequate in the next step, we need to check for each transition starting from this state whether the final states reached by the transition are non-bad and the non-final states are already known to be adequate. Thus, we have

$$\mathsf{adq}^i_{n+1}(q) = \bigwedge_{(q,q_1,\ldots,q_k)\in Q^{k+1}} wt(q,q_1,\ldots,q_k) \vee \bigvee_{q_j \notin F} \mathsf{adq}^i_j(q_j) \vee \bigvee_{q_j \in F} \mathsf{bad}_i(q_j). \quad (3)$$

The function adq^i is the limit of this inner iteration, which is reached after at most $|Q|$ steps. With this function, we define

$$\mathsf{bad}_{i+1}(q) = \mathsf{bad}_i(q) \vee \mathsf{adq}^i(q).$$

The function bad is the limit of this outer iteration, which is also reached after at most $|Q|$ steps. This computation of the function bad by two nested iterations basically simulates the computation of all bad states in the emptiness test for Büchi tree automata sketched above. It is thus easy to show that $\mathsf{bad}(q) = 1$ iff q is a bad state, i.e., cannot occur as a label in a successful run of $\mathcal{A}$.

Given the definition of $\mathcal{A}_w$, it is easy to see that a run $r : K^* \to Q$ of $\mathcal{A}_w$ has weight 0 iff it is a run of $\mathcal{A}$ (see Example 2). Consequently, $\mathcal{A}$ has a successful run that starts with an initial state iff $\|\mathcal{A}_w\| = \bigwedge_{r \in \mathsf{succ}_{\mathcal{A}_w}} in(r(\varepsilon)) \vee wt(r) = 0$. Putting these observations together, we thus have: the behaviour of $\mathcal{A}_w$ is 0 iff $\mathcal{A}$ has a successful run that starts with an initial state iff there is an initial state q (i.e., $in(q) = 0$) that is not bad (i.e., $\mathsf{bad}(q) = 0$). This shows that the behaviour of $\mathcal{A}_w$ is given by $\bigwedge_{q \in Q} in(q) \vee \mathsf{bad}(q)$.

Next, we show that the behaviour of a WBA can always be computed by such a procedure with two nested iterations.

Computing the Behaviour in the General Case of an Arbitrary WBA

In the following, we assume that $\mathcal{A} = (Q, in, wt, F)$ is an arbitrary, but fixed, WBA over the finite distributive lattice $(S, \leq_S)$. We will show that the WBA $\mathcal{A}$ induces a monotone operator $\mathcal{Q} : S^Q \to S^Q$, where S^Q is the set of all mappings from Q to S, and that the behaviour of $\mathcal{A}$ can easily be obtained from the greatest fixpoint of this operator. The partial order $\leq_S$ can be transferred to S^Q in the usual way, by applying it component-wise: for $\sigma, \sigma' \in S^Q$, we define $\sigma \leq_{S^Q} \sigma'$ iff $\sigma(q) \leq_S \sigma'(q)$ for all $q \in Q$. It is easy to see that $(S^Q, \leq_{S^Q})$ is again a finite distributive lattice. We will use $\otimes$ and $\oplus$ also to denote the infimum and supremum in S^Q. The least (greatest) element of S^Q is the function $\widetilde{\mathbf{0}}$ ($\widetilde{\mathbf{1}}$) that maps every $q \in Q$ to $\mathbf{0}$ ($\mathbf{1}$).

The definition of the operator $\mathcal{Q}$ will follow the idea of the iterative procedure we sketched before for solving the emptiness problem. We focus first on the inner loop, which is realized by another monotone operator $\mathcal{O}$. Notice that the internal iteration of the algorithm depends on the set of bad states computed so far. We will assume that this information is given by a function $f \in S^Q$. Thus, we actually define an operator $\mathcal{O}_f$ for each such f. Following the idea of Equation (3), the operator $\mathcal{O}_f$ is defined as follows for every $\sigma \in S^Q$:

$$\mathcal{O}_f(\sigma)(q) = \bigoplus_{(q,q_1,\ldots,q_k)\in Q^{k+1}} wt(q,q_1,\ldots,q_k) \otimes \bigotimes_{j=1}^{k} \mathsf{step}_f(\sigma)(q_j), \tag{4}$$

where

$$\mathsf{step}_f(\sigma)(q) = \begin{cases} f(q) & \text{if } q \in F \\ \sigma(q) & \text{otherwise} \end{cases}$$

Lemma 1 *For every $f \in S^Q$ the operator $\mathcal{O}_f$ is monotone, i.e., $\sigma \leq_{S^Q} \sigma'$ implies $\mathcal{O}_f(\sigma) \leq_{S^Q} \mathcal{O}_f(\sigma')$.*

Proof Let $\sigma, \sigma' \in S^Q$ be such that $\sigma \leq_{S^Q} \sigma'$. This implies also $\mathsf{step}_f(\sigma) \leq_{S^Q} \mathsf{step}_f(\sigma')$. Thus, we have for every $q \in Q$:

$$\begin{aligned} \mathcal{O}_f(\sigma)(q) &= \bigoplus_{(q,q_1,\ldots,q_k)\in Q^{k+1}} wt(q,q_1,\ldots,q_k) \otimes \bigotimes_{j=1}^{k} \mathsf{step}_f(\sigma)(q_j) \\ &\leq_S \bigoplus_{(q,q_1,\ldots,q_k)\in Q^{k+1}} wt(q,q_1,\ldots,q_k) \otimes \bigotimes_{j=1}^{k} \mathsf{step}_f(\sigma')(q_j) = \mathcal{O}_f(\sigma'). \end{aligned}$$

□

Since we know that S^Q is finite, this in particular means that the operator $\mathcal{O}_f$ is continuous. By Tarski's fixpoint theorem [39], this implies that $\mathcal{O}_f$ has $\bigoplus_{n\geq 0} \mathcal{O}_f^n(\widetilde{\mathbf{0}})$ as its least fixpoint (lfp). Finiteness of S^Q yields that this lfp is reached after finitely many iterations: there exists a smallest $m, 0 \leq m \leq |S|^{|Q|}$ such that $\mathcal{O}_f^m(\widetilde{\mathbf{0}}) = \mathcal{O}_f^{m+1}(\widetilde{\mathbf{0}})$, and for this m we have $\bigoplus_{n\geq 0} \mathcal{O}_f^n(\widetilde{\mathbf{0}}) = \mathcal{O}_f^m(\widetilde{\mathbf{0}})$. This yields a bound on the number of iterations that is exponential in the size of the automaton. We will later show (see Theorem 6) that it is possible to improve this bound to a polynomial number of iterations, measured in the number of states.

Recall that the intuition of the internal iteration was to find out from which states it is possible to build a finite partial run that finishes in final states. In the general case, the operators $\mathcal{O}$ will help in computing the weights of all such partial runs. Next, we give a formal definition of the notion of a finite partial run.

Definition 20 (finite run) A *finite tree* is a finite set $t \subseteq K^*$ that is closed under prefixes and such that, if $ui \in t$ for some $u \in K^*$ and $i \in K$, then $uj \in t$ for all $j, 1 \leq j \leq k$. A node $u \in t$ is called a *leaf* if there is no $j, 1 \leq j \leq k$, such that $uj \in t$. The set of all leaf nodes of a finite tree t is denoted by $\mathsf{lnode}(t)$. The *depth* of a finite tree t is the length of the largest word in t.

A *finite run* is a mapping $r : t \to Q$, where t is a finite tree. Given such a run, $\mathsf{leaf}(r)$ denotes the set of all states appearing as labels of a leaf.

We denote by runs_1 the set of all finite runs r of depth at least 1 such that, for every node $u \neq \varepsilon$, $r(u) \in F$ if and only if u is a leaf. Additionally, for every $n \geq 1$, let $\mathsf{runs}_1^{\leq n}$ denote the set of all finite runs in runs_1 having depth at most n. For a state $q \in Q$, $\mathsf{runs}_1(q) = \{r \in \mathsf{runs}_1 \mid r(\varepsilon) = q\}$; analogously, $\mathsf{runs}_1^{\leq n}(q) = \{r \in \mathsf{runs}_1^{\leq n} \mid r(\varepsilon) = q\}$. The *weight* of a finite run $r : t \to Q$ is $wt(r) = \bigotimes_{u \in t \setminus \mathsf{lnode}(t)} wt(r(u), r(u1), \ldots, r(uk))$.

Looking again at the special case of a weighted automaton simulating an unweighted one, we see that during the inner iteration we do not want to compute the weights of all finite runs in runs_1 but only those that finish in states that are not bad. In other words, we multiply the weight of the run, by the function bad computed so far applied to each of its leafs. Given a function $f : Q \to S$, we define the f-weight of a finite run r as $wt_f(r) = wt(r) \otimes \bigotimes_{q \in \mathsf{leaf}(r)} f(q)$. The lfp of the operator $\mathcal{O}_f$ computes the sum of the f-weights of all runs in runs_1.

Lemma 2 *For all $n \geq 0$ and all $q \in Q$, $\mathcal{O}_f^n(\widetilde{\mathbf{0}})(q) = \bigoplus_{r \in \mathsf{runs}_1^{\leq n}(q)} wt_f(r)$.*

Proof The proof is by induction on n. For $n = 0$, the result follows from the fact that $\mathsf{runs}_1^{\leq 0} = \emptyset$, and hence $\bigoplus_{r \in \mathsf{runs}_1^{\leq 0}(q)} wt_f(r) = \mathbf{0} = \widetilde{\mathbf{0}}(q) = \mathcal{O}_f^0(\widetilde{\mathbf{0}})(q)$. Assume now that the identity holds for n. Given a tuple $(q_1, \ldots, q_k) \in Q^k$, let $i_1, \ldots, i_l$ be all the indices such that $q_{i_j} \notin F$ for all $j, 1 \leq j \leq l$, and $i_{l+1}, \ldots, i_k$ those indices such that $q_{i_j} \in F$ for all $j, l+1 \leq j \leq k$. For $1 \leq j \leq l$, we will abbreviate $\mathsf{runs}_1^{\leq n}(q_{i_j})$ as rn_j^n and $\mathsf{leaf}(r_j)$ as lf_j. In addition, F is an abbreviation for the product $\bigotimes_{j=l+1}^{k} f(q_{i_j})$. Then,

$$\mathcal{O}_f^{n+1}(\widetilde{\mathbf{0}})(q) = \bigoplus_{(q_1,\ldots,q_k) \in Q^k} wt(q, q_1, \ldots, q_k) \otimes \bigotimes_{j=1}^{k} \mathsf{step}_f(\mathcal{O}_f^n(\widetilde{\mathbf{0}}))(q_j) \tag{5}$$

$$= \bigoplus_{(q_1,\ldots,q_k) \in Q^k} wt(q, q_1, \ldots, q_k) \otimes \bigotimes_{j=1}^{l} \mathcal{O}_f^n(\widetilde{\mathbf{0}})(q_{i_j}) \otimes \bigotimes_{j=l+1}^{k} f(q_{i_j}) \tag{6}$$

$$= \bigoplus_{(q_1,\ldots,q_k) \in Q^k} wt(q, q_1, \ldots, q_k) \otimes (\bigotimes_{j=1}^{l} \bigoplus_{r_j \in \mathsf{rn}_j^n} wt_f(r_j)) \otimes \mathsf{F} \tag{7}$$

$$= \bigoplus_{(q_1,\ldots,q_k) \in Q^k} wt(q, q_1, \ldots, q_k) \otimes (\bigoplus_{r_1 \in \mathsf{rn}_1^n, \ldots, r_l \in \mathsf{rn}_l^n} \bigotimes_{j=1}^{l} wt_f(r_j)) \otimes \mathsf{F} \tag{8}$$

$$= \bigoplus_{(q_1,\ldots,q_k) \in Q^k} wt(q, q_1, \ldots, q_k) \otimes (\bigoplus_{r_1 \in \mathsf{rn}_1^n, \ldots, r_l \in \mathsf{rn}_l^n} \bigotimes_{j=1}^{l} wt(r_j) \otimes \bigotimes_{p \in \mathsf{lf}_j} f(p)) \otimes \mathsf{F} \tag{9}$$

$$= \bigoplus_{(q_1,\ldots,q_k) \in Q^k} \bigoplus_{r_1 \in \mathsf{rn}_1^n, \ldots, r_l \in \mathsf{rn}_l^n} wt(q, q_1, \ldots, q_k) \otimes \bigotimes_{q_j \notin F} wt(r_j) \otimes \bigotimes_{p \in \mathsf{lf}_j} f(p) \otimes \mathsf{F} \tag{10}$$

$$= \bigoplus_{r \in \mathsf{runs}_1^{\leq n+1}(q)} wt(r) \otimes \bigotimes_{p \in \mathsf{leaf}(r)} f(p) \tag{11}$$

$$= \bigoplus_{r \in \mathsf{runs}_1^{\leq n+1}(q)} wt_f(r).$$

Identities (5) and (6) employ the definition of the operator $\mathcal{O}_f$ and step_f, respectively, and (7) applies the induction hypothesis. Identity (8) uses the fact that S^Q is a distributive lattice, which allows us to move the addition out of the product, while (9)

uses the definition of the f-weight. Identity (10) uses again the distributivity to multiply $wt(q, q_1, \ldots, q_k)$ in. Finally, Identity (11) simplifies the two sums by constructing a run of larger depth. Instead of considering first the transition $(q, q_1, \ldots, q_k)$ and then runs of depth up to n starting with each q_{i_j}, we simply take the corresponding run of depth $n+1$ starting at q. This run labels the root with q and the successor node i with q_i. If q_i is a final state, then it remains as a leaf, otherwise, below the node i we have the former run starting with q_i. Thus, the set of leafs of this larger run is the union of the sets of leafs of the runs r_j and the set of those q_is that are final states. The last identity merely applies the definition of f-weight again. □

Theorem 4 *Let $f \in S^Q$ and assume that σ_0 is the lfp of the operator $\mathcal{O}_f$. Then, for every $q \in Q$, $\sigma_0(q) = \bigoplus_{r \in \mathsf{runs}_1(q)} wt_f(r)$.*

Proof By Lemma 2, we have

$$\bigoplus_{n \geq 0} \mathcal{O}_f^n(\widetilde{\mathbf{0}})(q) = \bigoplus_{n \geq 0} \bigoplus_{r \in \mathsf{runs}_1^{\leq n}(q)} wt_f(r) = \bigoplus_{r \in \mathsf{runs}_1(q)} wt_f(r).$$

Tarski's fixpoint theorem says that the least fixpoint of $\mathcal{O}_f$ is $\bigoplus_{n \geq 0} \mathcal{O}_f^n(\widetilde{\mathbf{0}})$, which completes the proof of the theorem. □

Before turning our attention to the outer iteration of the method for computing the behaviour, we will present a bound on the number of steps that are necessary before reaching the fixpoint of the inner iteration.

Definition 21 A WBA is *m-finalising* if, for every $f \in S^Q$ and every partial run r in $\mathsf{runs}_1(q)$, there is a partial run s_r in $\mathsf{runs}_1^{\leq m}(q)$ such that $wt_f(r) \leq_S wt_f(s_r)$.

We will first show that every WBA is m-finalising for any m greater to the number of states $|Q|$. Afterwards we will show how this property yields a bound on the number of iterations needed to reach the least fixpoint of $\mathcal{O}_f$.

Theorem 5 *Let $\mathcal{A}$ be a WBA with less than m states. Then $\mathcal{A}$ is m-finalising.*

Proof Let $f \in S^Q$ and consider a run $r \in \mathsf{runs}_1(q)$. If $r \in \mathsf{runs}_1^{\leq m}(q)$, then there is nothing to prove. Otherwise, if $r \notin \mathsf{runs}_1^{\leq m}(q)$, then there must be a path in r of length greater than m. Since there are less than m different states, there must be two non-root nodes u, v in this path such that $r(u) = r(v)$. Since these nodes are on the same path, we can assume w.l.o.g. that $v = uv'$ for some $v' \in K^* \setminus \{\varepsilon\}$. We define a new run s as follows: for every node w, if there is no w' for which $w = uw'$, then set $s(w) := r(w)$; otherwise (that is, if $w = uw'$ for some w') set $s(uw') := r(vw')$. This construction defines an injective function g from the nodes of s to the nodes of r such that, for every node w of s, we have $s(w) = r(g(w))$. Notice that this function is not surjective, as there is no w such that $g(w) = u$. Thus, s has less nodes than r. Furthermore, every transition in s is also a transition in r, and for every $w \in \mathsf{leaf}(s)$, $g(w) \in \mathsf{leaf}(r)$. This implies that $wt_f(r) \leq_S wt_f(s)$. If s is still not in $\mathsf{runs}_1^{\leq m}$, then we can repeat the same process to produce a smaller run s' with a smaller f-weight, until we find one that is in $\mathsf{runs}_1^{\leq m}$. □

Theorem 6 *If $\mathcal{A}$ is m-finalising, then $\mathcal{O}_f^m(\widetilde{\mathbf{0}})$ is the lfp of $\mathcal{O}_f$.*

Proof Let σ_0 be the lfp of $\mathcal{O}_f$. We know that σ_0 is the supremum of $\{\mathcal{O}_f^n(\widetilde{\mathbf{0}}) \mid n \geq 0\}$; thus, it is sufficient to show that $\mathcal{O}_f^m(\widetilde{\mathbf{0}})(q) \geq_S \sigma_0(q)$ for all $q \in Q$. By Theorem 4, we know that $\sigma_0(q) = \bigoplus_{r \in \mathsf{runs}_1(q)} wt_f(r)$. Since $\mathcal{A}$ is m-finalising, we can replace every $r \in \mathsf{runs}_1(q)$ by the corresponding $s_r \in \mathsf{runs}_1^{\leq m}(q)$, thus obtaining a greater element in the lattice. Hence,

$$\sigma_0(q) \leq_S \bigoplus_{r \in \mathsf{runs}_1(q)} wt_f(s_r) \leq_S \bigoplus_{s \in \mathsf{runs}_1^{\leq m}(q)} wt_f(s) = \mathcal{O}_f^m(\widetilde{\mathbf{0}})(q),$$

which completes the proof of the theorem. □

This theorem tells us that, in order to construct the lfp of the operator $\mathcal{O}_f$, it is enough to apply this operator $|Q|+1$ times. Since each of the iteration steps also requires only polynomial time, as a function of the number of states Q, we know that the computation of the lfp needs overall polynomial time in the number of states, independently of the lattice used. As mentioned before, this bound greatly improves on the trivial obtained from the finiteness S^Q since the trivial bound is exponential in the number of states of the automaton and depends also on the size of the lattice S.

We focus now on the outer iteration of the algorithm. For the unweighted case, this iteration mainly updates the set of bad states with the information obtained from the internal iteration. To do this, we define the operator $\mathcal{Q} : S^Q \to S^Q$ as follows: for all $\sigma \in S^Q$

$$\mathcal{Q}(\sigma) := \mathsf{lfp}(\mathcal{O}_\sigma),$$

where lfp represents the least fixpoint.

We will show that, again, a repeated application of this operator leads to an appropriate fixpoint, due to the fact that $\mathcal{Q}$ is monotone and S^Q is finite.

Lemma 3 *The operator $\mathcal{Q}$ is monotone.*

Proof Let $\sigma, \sigma' \in S^Q$ such that $\sigma \leq_{S^Q} \sigma'$. Notice first that, for every run $r \in \mathsf{runs}_1$, this implies that $wt_\sigma(r) \leq_S wt_{\sigma'}(r)$. From this we obtain, for every $q \in Q$,

$$\begin{aligned}
\mathcal{Q}(\sigma)(q) &= \mathsf{lfp}(\mathcal{O}_\sigma)(q) \\
&= \bigoplus_{r \in \mathsf{runs}_1(q)} wt_\sigma(r) && (12) \\
&\leq_S \bigoplus_{r \in \mathsf{runs}_1(q)} wt_{\sigma'}(r) \\
&= \mathsf{lfp}(\mathcal{O}_{\sigma'})(q) && (13) \\
&= \mathcal{Q}(\sigma'(q),
\end{aligned}$$

where Identities (12) and (13) follow from Theorem 4 and the inequality is a consequence of the remark at the beginning of this proof. □

Again, finiteness of S^Q implies that the operator $\mathcal{Q}$ is actually continuous, and thus Tarski's fixpoint theorem says that $\mathcal{Q}$ has $\bigotimes_{n \geq 0} \mathcal{Q}^n(\widetilde{\mathbf{1}})$ as its greatest fixpoint (gfp). It remains to show how this gfp can be used to compute the behaviour of a given WBA. Let $\mathsf{succ}_\mathcal{A}(q)$ denote the set of all successful runs of $\mathcal{A}$ whose root is labelled with q. Consider the function $\sigma^{\|} \in S^Q$ where $\sigma^{\|}(q) := \bigoplus_{r \in \mathsf{succ}_\mathcal{A}(q)} wt(r)$. Given this function, we can obtain the behaviour of the WBA $\mathcal{A}$ as follows:

Lemma 4 $\|\mathcal{A}\| = \bigoplus_{q \in Q} in(q) \otimes \sigma^{\|}(q)$.

It turns out that $\sigma^{\|}$ is in fact the greatest fixpoint of $\mathcal{Q}$. Before proving this, we will introduce some additional notation. We will use the expression runs_n for $n \geq 1$ to denote the set of all finite runs such that every path from the root to a leaf has *exactly* n non-root nodes labelled with a final state, the last of which is the leaf.

Given a run $r \in \mathsf{runs}_n$, its *preamble* is the unique finite run $s \in \mathsf{runs}_1$ such that, for every node u, if $s(u)$ is defined, then $s(u) = r(u)$. We will denote the preamble of r by $\mathsf{pre}(r)$. Notice that, if $r \in \mathsf{runs}_n$ for $n \geq 1$, then its preamble always exists, and can be constructed as follows: first set $\mathsf{pre}(r)(\varepsilon) = r(\varepsilon)$ and $\mathsf{pre}(r)(i) = r(i)$ for all $i, 1 \leq i \leq k$. Then, for every node u for which $\mathsf{pre}(r)(u)$ is defined, if $r(u) \in F$, then u is a leaf of $\mathsf{pre}(r)$; otherwise, set $\mathsf{pre}(r)(ui) = r(ui)$ for all $i, 1 \leq i \leq k$. This construction finishes since, in every path, we must find at least one final state, which will be a leaf in $\mathsf{pre}(r)$, and thus also $\mathsf{pre}(r) \in \mathsf{runs}_1$.

Given a (finite) run r and a node u in r, we will denote the *subrun of r starting at u* as $r_{|u}$. More formally, $r_{|u}$ is the run such that, for every $v \in K^*$, if $r(uv)$ is defined, then $r_{|u}(v) = r(uv)$.

The next lemma relates the number n of times the operator $\mathcal{Q}$ has been applied to the greatest element $\widetilde{\mathbf{1}}$ of S^Q to the weights of the runs in runs_n.

Lemma 5 *For all $n > 0$ and $q \in Q$ it holds that*

$$\mathcal{Q}^n(\widetilde{\mathbf{1}})(q) = \bigoplus_{r \in \mathsf{runs}_n(q)} wt(r).$$

Proof We prove this fact also by induction on n. For $n = 1$, the result follows directly from Theorem 4. Assume now that it holds for n.

$$\begin{aligned}
\mathcal{Q}^{n+1}(\widetilde{\mathbf{1}})(q) &= \mathsf{lfp}(\mathcal{O}_{\mathcal{Q}^n(\widetilde{\mathbf{1}})})(q) \\
&= \bigoplus_{r \in \mathsf{runs}_1(q)} wt_{\mathcal{Q}^n(\widetilde{\mathbf{1}})}(r) && (14)\\
&= \bigoplus_{r \in \mathsf{runs}_1(q)} wt(r) \otimes \bigotimes_{p \in \mathsf{leaf}(r)} \mathcal{Q}^n(\widetilde{\mathbf{1}})(p) && (15)\\
&= \bigoplus_{r \in \mathsf{runs}_1(q)} wt(r) \otimes \bigotimes_{p \in \mathsf{leaf}(r)} \bigoplus_{s \in \mathsf{runs}_n(p)} wt(s) && (16)\\
&= \bigoplus_{r \in \mathsf{runs}_1(q)} wt(r) \otimes \bigotimes_{u \in \mathsf{lnode}(r)} \bigoplus_{s \in \mathsf{runs}_n(r(u))} wt(s) && (17)\\
&= \bigoplus_{r \in \mathsf{runs}_1(q)} wt(r) \otimes \bigoplus_{\{t \in \mathsf{runs}_{n+1}(q) \mid \mathsf{pre}(t) = r\}} \bigotimes_{u \in \mathsf{lnode}(r)} wt(t_{|u}) && (18)\\
&= \bigoplus_{r \in \mathsf{runs}_1(q)} \bigoplus_{\{t \in \mathsf{runs}_{n+1}(q) \mid \mathsf{pre}(t) = r\}} wt(r) \otimes \bigotimes_{u \in \mathsf{lnode}(r)} wt(t_{|u}) && (19)\\
&= \bigoplus_{r \in \mathsf{runs}_1(q)} \bigoplus_{\{t \in \mathsf{runs}_{n+1}(q) \mid \mathsf{pre}(t) = r\}} wt(t) && (20)\\
&= \bigoplus_{s \in \mathsf{runs}_{n+1}(q)} wt(s). && (21)
\end{aligned}$$

The first identity employs only the definition of $\mathcal{Q}$. Theorem 4 yields Identity (14). Identities (15) and (16) follow from the definition of f-weights and the induction hypothesis, respectively. Identity (17) changes the indices to run over the set of leaf nodes, rather than by the states that label them; the idempotency of the operators $\oplus$ and $\otimes$ implies that this change does not alter the result. For Identity (18) we use the distributivity of the lattice. The definition of distributivity says that, in order to exchange the operators $\oplus$ and $\otimes$, the now external addition needs to range over all functions mapping nodes $u \in \mathsf{lnode}(r)$ to runs $s \in \mathsf{runs}_n(r(u))$. We notice that each function of this kind, together with the run $r \in \mathsf{runs}_1(q)$, defines exactly one finite run $t \in \mathsf{runs}_{n+1}(q)$. We thus use this t to represent the function. Identity (19) is an easy consequence of distributivity. For Identity (20), we then use the fact that a run in runs_{n+1} can be seen as its preamble (in runs_1) concatenated at each of its leafs with a run in runs_n. Finally, for Identity (21) we notice that the set of all runs in runs_{n+1} can be partitioned by means of their preambles, which means that both sides of the identity range over the same runs. □

As it was the case for the operator $\mathcal{O}$ in the internal iteration, we can bound the number of iterations that $\mathcal{Q}$ needs before reaching a fixpoint by the number of states of the automaton.

Definition 22 (m-complete) A WBA $\mathcal{A}$ is *m-complete* if, for every partial run $r \in \mathsf{runs}_m(q)$, there is a successful run $s_r \in \mathsf{succ}_{\mathcal{A}}(q)$ such that $wt(r) \leq_S wt(s_r)$.

Using the fact that $\otimes$ is idempotent, it is easy to see that every WBA is m-complete for any m greater than the number of final states $|F|$. The proof is similar to the one given in [3] for the fact that a looping automaton has a run iff it has a partial run of depth greater than $|Q|$. However, we now also need to take into account which are the states that are final, and which are not.

Theorem 7 *If $\mathcal{A}$ is a WBA with less than m final states, then $\mathcal{A}$ is m-complete.*

Proof Suppose that we have a partial run $r : t \to Q$ in $\mathsf{runs}_m(q)$. We use r to construct a function $\beta : K^* \to t$ by induction. With this function, we then construct a successful run s_r by setting $s_r(u) := r(\beta(u))$. The intuitive meaning of $\beta(v) = w$ is that, in the run s_r, the node v will have the same label as the node w in r. We define β as follows:

- $\beta(\varepsilon) := \varepsilon$,
- for a node $v \cdot i$, if there is a predecessor w of $\beta(v) \cdot i$ such that (i) $r(\beta(v) \cdot i) = r(w)$, and (ii) $r(w) \in F$, then set $\beta(v \cdot i) := w$; otherwise, set $\beta(v \cdot i) := \beta(v) \cdot i$.

Notice that the function β is well-defined since, for every $v \in K^*$, we have that $\beta(v)$ is not a leaf node of t. In fact, whenever we find a final state several times in the same path, the mapping β always leads to the earliest one. Thus, reaching a leaf would mean that we have a path reaching m final states, where none of them repeats, contradicting the fact that the automaton has less than m final states in total.

We now show that it is possible to construct a successful run s_r from r by defining $s_r(v) = r(\beta(v))$ for all $v \in K^*$, and that $wt(r) \leq_S wt(s_r)$. Our definition of β ensures that, for every $v \in K^*$ and $i \in \mathsf{K}$, it holds that $s_r(v \cdot i) = r(\beta(v) \cdot i)$. Thus, for every $v \in K^*$, we have $(s_r(v), s_r(v1), \ldots, s_r(vk)) = (r(\beta(v)), r(\beta(v) \cdot 1), \ldots, r(\beta(v) \cdot k))$, and hence,

$$wt(s_r(v), s_r(v1), \ldots, s_r(vk)) = wt(r(\beta(v)), r(\beta(v) \cdot 1), \ldots, r(\beta(v) \cdot k)).$$

This implies that every factor in the product $wt(s_r)$ is also a factor in the product $wt(r)$. Since the product computes the infimum, we thus have $wt(r) \leq_S wt(s_r)$.

It remains only to show that s_r is successful. Suppose to the contrary that s_r is not successful. Then, there must exist a path p and a node $v \in \mathsf{p}$ such that all its successors in p are labelled with non-final states. In other words, for every $w \in K^*$, if $v \cdot w \in \mathsf{p}$, then $s_r(v \cdot w) \notin F$. This implies, by our definition of β, that $\beta(v \cdot w) = \beta(v) \cdot w$, for all $v \cdot w \in \mathsf{p}$. Thus, r has an infinite path, which contradicts the assumption that $r \in \mathsf{runs}_m$. □

The following theorem states that it is possible to compute the mapping $\sigma^{\|}$ for an m-complete automaton by applying the $\mathcal{Q}$ operator to the greatest element of S^Q at most m times.

Theorem 8 *If $\mathcal{A}$ is an m-complete WBA, then $\mathcal{Q}^m(\widetilde{\mathbf{1}}) = \sigma^{\|}$.*

Proof Notice first that, by Lemma 5, we know that $\mathcal{Q}^m(\mathbf{1})(q) = \bigoplus_{r \in \mathsf{runs}_m(q)} wt(r)$. Since $\mathcal{A}$ is m-complete, we can replace each of these partial runs by a successful run, which yields

$$\mathcal{Q}^m(\widetilde{\mathbf{1}})(q) \leq_S \bigoplus_{r \in \mathsf{runs}_m(q)} wt(s_r) \leq_S \bigoplus_{s \in \mathsf{succ}(q)} wt(s) = \sigma^{\|}(q).$$

To prove the inequality in the other direction, notice that, given a successful run r, we can truncate it at every path when m final states have been found. The result of this is a finite run since otherwise, as the tree is finitely branching, König's Lemma would imply the existence of an infinite path in this tree. Since we truncate each branch whenever we have found m final states, an infinite path would be one on which less than m final states occur, contradicting the fact that r is a successful run. Thus, the partial run r_m constructed this way belongs to runs_m. Notice that, for every node u of r_m, it holds that $r_m(u) = r(u)$. Hence, we have $wt(r) \leq_S wt(r_m)$. This yields

$$\begin{aligned} \sigma^{\|}(q) = \bigoplus_{r \in \mathsf{succ}(q)} wt(r) &\leq_S \bigoplus_{r \in \mathsf{succ}(q)} wt(r_m) \\ &\leq_S \bigoplus_{s \in \mathsf{runs}_m(q)} wt(s) = \mathcal{Q}^m(\widetilde{\mathbf{1}})(q). \end{aligned}$$

Putting the two inequalities together proves the theorem. □

In particular, this theorem shows that the mapping $\sigma^{\|}$ is indeed the gfp of $\mathcal{Q}$.

Corollary 1 *The mapping $\sigma^{\|}$ is the greatest fixpoint of $\mathcal{Q}$.*

Proof Since S^Q is finite, the gfp of $\mathcal{Q}$ is reached after finitely many iterations; more precisely, if $n_0 > |S|^{|Q|}$, then this gfp is $\bigotimes_{n \geq 0} \mathcal{Q}^n(\widetilde{\mathbf{1}}) = \mathcal{Q}^{n_0}(\widetilde{\mathbf{1}})$. Obviously, we can choose n_0 such that $n_0 > |F|$. Theorem 7 then says that the automaton is n_0-complete. Thus, by Theorem 8, it follows that $\mathcal{Q}^{n_0}(\widetilde{\mathbf{1}}) = \sigma^{\|}$. □

Overall, we have thus shown how to compute the behaviour of a WBA. By Lemma 4, $\|\mathcal{A}\| = \bigoplus_{q \in Q} in(q) \otimes \sigma^{\|}(q)$. The above corollary says that $\sigma^{\|}$ is the greatest fixpoint of $\mathcal{Q}$. Let us illustrate this process by using it to compute the behaviour of the pinpointing automaton of Example 4.

Example 6 To compute the behaviour of the pinpointing automaton introduced in Example 4, we need to find the greatest fixpoint of $\mathcal{Q}$, found after repeated applications of $\mathcal{Q}$ to $\widetilde{\mathbf{1}}$. By definition, $\mathcal{Q}(\widetilde{\mathbf{1}}) = \mathsf{lfp}(\mathcal{O}_{\widetilde{\mathbf{1}}})$; hence, we repeatedly apply $\mathcal{O}_{\widetilde{\mathbf{1}}}$ to $\widetilde{\mathbf{0}}$ to find this least fixpoint. This operator is defined as

$$\mathcal{O}_{\widetilde{\mathbf{1}}}(\sigma)(p) = \bigwedge_{(p,p_1,p_2)\in Q^3} wt(p,p_1,p_2) \vee \mathsf{step}_{\widetilde{\mathbf{1}}}(\sigma)(p_1) \vee \mathsf{step}_{\widetilde{\mathbf{1}}}(\sigma)(p_2),$$

where $\mathsf{step}_{\widetilde{\mathbf{1}}}(\sigma)(p) = \bot$ if $p \in \{q_1, q_3\}$ and $\sigma(p)$ otherwise. The first iteration of the fixpoint computation looks as follows:[11]

$$\begin{aligned}
\mathcal{O}_{\widetilde{\mathbf{1}}}(\widetilde{\mathbf{0}})(q_0) &= (wt(q_0,q_1,q_1) \vee \bot \vee \bot) \wedge (wt(q_0,q_2,q_2) \vee \top \vee \top) \\
&= (\bot \vee \bot \vee \bot) \wedge (\bot \vee \top \vee \top) = \bot, \\
\mathcal{O}_{\widetilde{\mathbf{1}}}(\widetilde{\mathbf{0}})(q_1) &= wt(q_1,q_1,q_1) \vee \bot \vee \bot = \mathrm{ax}_1 \vee \bot \vee \bot = \mathrm{ax}_1, \\
\mathcal{O}_{\widetilde{\mathbf{1}}}(\widetilde{\mathbf{0}})(q_2) &= (wt(q_2,q_2,q_2) \vee \top \vee \top) \wedge (wt(q_2,q_3,q_3) \vee \bot \vee \bot) \\
&= (\mathrm{ax}_3 \vee \top \vee \top) \wedge (\bot \vee \bot \vee \bot) = \bot, \\
\mathcal{O}_{\widetilde{\mathbf{1}}}(\widetilde{\mathbf{0}})(q_3) &= \top.
\end{aligned}$$

Analogously, we can compute $\mathcal{O}^2_{\widetilde{\mathbf{1}}}(\widetilde{\mathbf{0}}) = \mathcal{O}_{\widetilde{\mathbf{1}}}(\widetilde{\mathbf{0}}) = (\bot, \mathrm{ax}_1, \bot, \top)$, which means that we have found the least fixpoint; hence $\mathcal{Q}(\widetilde{\mathbf{1}}) = (\bot, \mathrm{ax}_1, \bot, \top)$.

For the second iteration, we get that $\mathcal{O}^2_{\mathcal{Q}(\widetilde{\mathbf{1}})}(\widetilde{\mathbf{0}}) = \mathcal{O}_{\mathcal{Q}(\widetilde{\mathbf{1}})}(\widetilde{\mathbf{0}}) = (\mathrm{ax}_1, \mathrm{ax}_1, \top, \top)$, and thus $\mathcal{Q}^2(\widetilde{\mathbf{1}}) = (\mathrm{ax}_1, \mathrm{ax}_1, \top, \top)$. A further iteration of this operator yields $\mathcal{Q}^3(\widetilde{\mathbf{1}}) = \mathcal{Q}^2(\widetilde{\mathbf{1}})$ and hence we have found the greatest fixpoint $\sigma^{\|}$ of $\mathcal{Q}$.

Knowing this fixpoint, we can now compute the behaviour of $(\mathcal{A}^{ex}, \Delta\mathsf{res}, I\mathsf{res})^{\mathsf{pin}}$:

$$\begin{aligned}
\|(\mathcal{A}^{ex}, \Delta\mathsf{res}, I\mathsf{res})^{\mathsf{pin}}\| &= \bigwedge_{i=0}^{3} in(q_i) \vee \sigma^{\|}(q_i) \\
&= (\mathrm{ax}_2 \vee \mathrm{ax}_1) \wedge (\top \vee \mathrm{ax}_1) \wedge (\top \vee \top) \wedge (\top \vee \top) \\
&= \mathrm{ax}_2 \vee \mathrm{ax}_1,
\end{aligned}$$

which is identical to the behaviour that we have computed in an ad hoc manner in Example 4.

In general, the fixpoint $\sigma^{\|}$ can be computed in $m_o := |F| + 1$ iteration steps since m_o is larger than the number of final states of the input WBA (Theorems 7 and 8). Each step of this outer iteration consists of computing the least fixpoint of the operator $\mathcal{O}_\sigma$, where σ is the result of the previous step. This fixpoint can be computed in $m_i = |Q|+1$ iteration steps since m_i is larger that the number of states of the input WBA (Theorems 5 and 6). Such an inner iteration step requires a polynomial number of lattice operations (in the cardinality $|Q|$ of Q).

Thus, to analyze the *complexity* of our algorithm for computing the behaviour of a WBA, we need to know the complexity of applying the lattice operations. If we assume that this complexity is constant (i.e., the lattice S is assumed to be fixed), then we end up with an overall polynomial time complexity. However, this is not always a reasonable assumption. In fact, we were able to restrict our attention to finite distributive lattices by taking, for a given WBA, the distributive lattice generated by the weights occurring

[11] For brevity, we consider only those transitions that have a weight different from $\top$.

in it (where these weights may come from an underlying infinite distributive lattice). Thus, the actual finite distributive lattice used may depend on the automaton. Let us assume that the lattice operations can be performed using time polynomial in the size of any generating set. Since the size of this generating set is itself polynomial in the number of states of the input WBA $\mathcal{A}$, this assumption implies that the lattice operations can be performed in time polynomial in the size of the automaton. Thus, under this assumption, we have an overall polynomial bound (measured in the number of states) for the computation of the behaviour of a WBA.

In the case of pinpointing, we use the $\mathcal{T}$-Boolean semiring $\mathbb{B}^{\mathcal{T}}$, which is the free distributive lattice generated by the set $\mathsf{lab}(\mathcal{T})$. The lattice operations are conjunction and disjunction of monotone Boolean formulae. Note that, strictly speaking, the lattice elements are monotone Boolean formulae *modulo equivalence*, i.e., equivalence classes of monotone Boolean formulae. However, since equivalence of monotone Boolean formulae is known to be an NP-complete problem, we do not try to compute unique representatives of the equivalence classes. We just leave the formulae as they are. Nevertheless, if we are not careful, then the computed pinpointing formula may still be exponential in the size of the automaton, though we apply only a polynomial number of conjunction and disjunction operations. The reason is that we may have to create copies of subformulae. However, this problem can easily be avoided by employing structure sharing, i.e., using directed acyclic graphs (DAGs) as data structure for monotone Boolean formulae.

Corollary 2 *Let Γ be an axiomatized input and $(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})$ an axiomatic automaton for Γ w.r.t. the c-property $\mathcal{P}$ such that $\mathcal{A}$ is a WBA. Then a DAG representation of a pinpointing formula for Γ w.r.t. $\mathcal{P}$ can be computed in time polynomial in the size of $\mathcal{A}$.*

We will show in Section 5.3 below that there is a behaviour-preserving polynomial-time reduction of WGBA to WBA. This implies that the above Corollary 2 also holds for the case where $\mathcal{A}$ is a *generalized* WBA. Note, however, that the size of the automata we have constructed for $\mathcal{SI}$ and LTL is already exponential in the size of the input. Thus, the (DAG representation of the) pinpointing formula may still be exponential in the size of the input, and computing it may take exponential time in the size of the input.

We proceed now to show how the method for computing the behaviour of a WBA introduced above can be used for computing the behaviour of the other two kinds of weighted automata we have defined, namely, WLA and WGBA.

5.2 Computing the Behaviour of a WLA

A WLA is a WGBA that has no set of final states. In this case, the condition for a run to be successful—that is, that every path must have infinitely many states labelled with elements of F_i for each set of final states F_i—is trivially satisfied. Thus, every run of a weighted looping automaton is successful. Alternatively, we can view the WLA (Q, in, wt) as the WBA (Q, in, wt, Q) since every state being a final state also means that every run is successful. Thus, WLAs are special kinds of WBAs, which shows that our approach for computing the behaviour of WBAs can directly be applied to WLAs.

However, the fact that every run is successful can be used to simplify the procedure into one that uses only a single iteration.

Notice first that the operator $\mathcal{O}_f$ depends on the set of final states. More precisely, the set of final states is used in the definition of the auxiliary function step_f:

$$\mathsf{step}_f(\sigma)(q) = \begin{cases} f(q) & \text{if } q \in F \\ \sigma(q) & \text{otherwise} \end{cases}$$

If all states are final, then no case analysis is necessary in step_f, and hence $\mathsf{step}_f(\sigma)(q) = f(q)$ for all $\sigma \in S^Q$ and all $q \in Q$. This collapses the definition of the operator $\mathcal{O}_f$ to

$$\mathcal{O}_f(\sigma)(q) = \bigoplus_{(q,q_1,\ldots,q_k)\in Q^{k+1}} wt(q,q_1,\ldots,q_k) \otimes \bigotimes_{j=1}^{k} f(q_j).$$

Notice that in this case $\mathcal{O}_f$ does not depend on the input σ, and hence its only fixpoint is reached after exactly one iteration. This allows us to simplify the definition of the operator $\mathcal{Q}$ in the following way:

$$\begin{aligned} \mathcal{Q}(\sigma)(q) &= \mathsf{lfp}(\mathcal{O}_\sigma)(q) \\ &= \mathcal{O}_\sigma(\widetilde{\mathbf{0}})(q) \\ &= \bigoplus_{(q,q_1,\ldots,q_k)\in Q^{k+1}} wt(q,q_1,\ldots,q_k) \otimes \bigotimes_{j=1}^{k} \sigma(q_j) \end{aligned}$$

The behaviour of a WLA is then the gfp of this operator $\mathcal{Q}$, which can be computed by a single iteration. The inner iteration of the general procedure is replaced by a direct application of the simplified definition of $\mathcal{Q}$. Note that this simplified definition of $\mathcal{Q}$ coincides with the one introduced in [6] specifically for WLAs. Thus, the "nested iteration approach" for WBAs developed in the present paper can be seen as a direct generalization of the "bottom-up approach" introduced in [6] for the case of WLAs.

Let us apply this insight to the pinpointing automaton for $\mathcal{SI}$ constructed in Section 4.2. This automaton has exponentially many states in the size n of the input $(C, \mathcal{T})$. Thus, we need exponentially many applications of the operator $\mathcal{Q}$, when measured on n. It is also easy to see that the time required by each application of $\mathcal{Q}$ is polynomial in the size of the automaton, and thus exponential in n. Hence, this leads to an algorithm with a total running time that is exponential in the size of the input.

Corollary 3 *Let C be an $\mathcal{ALC}$ concept description and $\mathcal{T}$ an $\mathcal{SI}$ TBox. A pinpointing formula for $(C, \mathcal{T})$ w.r.t. unsatisfiability can be computed in time exponential in the size of $(C, \mathcal{T})$.*

Since even deciding satisfiability of $\mathcal{ALC}$ concept descriptions w.r.t. $\mathcal{SI}$ TBoxes is known to be ExpTime-hard, this bound is optimal.

5.3 Computing the Behaviour of a Generalized WBA

We have shown how to compute the behaviour of a WBA in time polynomial in the number of states. We will now give a polynomial reduction in which, for every WGBA, we construct a WBA that has the same behaviour, transferring this way the problem of

computing the behaviour of WGBAs to the special case of WBAs that we have already solved. The idea of the reduction is to make several copies of the set of states and use each copy to test the Büchi condition for a specific set of final states, moving to the next copy once we have found a final state of the set we are currently looking at. This is the same idea as the one used in the unweighted case to reduce the emptiness problem for GBAs to the one for BAs [41].

Let $\mathcal{A} = (Q, in, wt, F_0, \ldots, F_{n-1})$ be a WGBA. We construct the WBA $\mathcal{A}' = (Q', in', wt', F')$ as follows:

- $Q' = \{(q,i) \mid q \in Q, 0 \leq i \leq n-1\}$,
- $in'(q,i) = \begin{cases} in(q) & \text{if } i = 0, \\ \mathbf{0} & \text{otherwise} \end{cases}$
- $wt'((q_0,i),(q_1,j),\ldots,(q_k,j)) = \begin{cases} wt(q_0,q_1,\ldots,q_k) & \text{if } q_0 \in F_i,\ j = i+1 \mod n, \\ wt(q_0,q_1,\ldots,q_k) & \text{if } q_0 \notin F_i,\ i = j \\ \mathbf{0} & \text{otherwise} \end{cases}$
- $F' = \{(q, n-1) \mid q \in F_{n-1}\}$.

Notice that the automaton $\mathcal{A}'$ has $n \cdot |Q|$ states, where n is the number of sets of final states. Since there can potentially be $2^{|Q|}$ sets of final states, this reduction is not polynomial when measured only in the number of states of $\mathcal{A}$, but it is polynomial in the total size of the automaton $\mathcal{A}$.

Theorem 9 *If $\mathcal{A}$ is a WGBA and $\mathcal{A}'$ is constructed as above, then $\|\mathcal{A}\| = \|\mathcal{A}'\|$.*

Proof Recall first that the behaviour of an automaton is the addition of the weights of all successful runs multiplied with the initial distribution of their root labels. If a run r is such that $in(r(\varepsilon)) \otimes wt(r) = \mathbf{0}$, then it will not be of interest, since it will not influence the computation of the behaviour. Given a WGBA or WBA $\mathcal{B}$, let $\mathsf{supp}(\mathcal{B})$ be the set of all runs r such that $in(r(\varepsilon)) \otimes wt(r) \neq \mathbf{0}$. We introduce a bijective function $f : \mathsf{supp}(\mathcal{A}) \to \mathsf{supp}(\mathcal{A}')$ such that, for every run $r \in \mathsf{supp}(\mathcal{A})$, $wt(r) = wt'(f(r))$ and r is successful (w.r.t. $\mathcal{A}$) iff $f(r)$ is successful (w.r.t. $\mathcal{A}'$).

Let r be a run in $\mathsf{supp}(\mathcal{A})$. We define the run $f(r)$ of $\mathcal{A}'$ inductively as follows:

- $f(r)(\varepsilon) = (r(\varepsilon), 0)$;
- let $u \in \mathsf{K}^*$ and $f(r)(u) = (q,i)$. Then, for all $1 \leq j \leq k$,
  $f(r)(u \cdot j) = \begin{cases} (r(u \cdot j), i) & \text{if } q \notin F_i, \\ (r(u \cdot j), i+1 \mod n) & \text{if } q \in F_i. \end{cases}$

Let $u \in \mathsf{K}^*$ and $f(r)(u) = (q,i)$. Then $r(u) = q$. Furthermore, for all $1 \leq j \leq k$, $f(r)(uj) = (r(uj), i+1 \mod n)$ if $q \in F_i$ and $f(r)(uj) = (r(uj), i)$ otherwise. Together with the definition of wt', this implies

$$wt'(f(r)(u), f(r)(u1), \ldots, f(r)(uk)) = wt(r(u), r(u1), \ldots, r(uk)).$$

This yields $wt(r) = wt'(f(r))$. Since we also have $in'(f(r)(\varepsilon)) = in(r(\varepsilon))$, the fact that $in(r(\varepsilon)) \otimes wt(r) \neq \mathbf{0}$ also implies that $in'(f(r)(\varepsilon)) \otimes wt'(f(r)) \neq \mathbf{0}$. Thus, f is indeed a function from $\mathsf{supp}(\mathcal{A})$ to $\mathsf{supp}(\mathcal{A}')$.

It is easy to see that f is injective. We show now that it is also surjective. Let $s \in \mathsf{supp}(\mathcal{A}')$. We construct a run $r \in \mathsf{supp}(\mathcal{A})$ as follows: for every $u \in \mathsf{K}^*$, if $s(u) = (q,i)$, then $r(u) = q$. We show that $s = f(r)$. First, since $in'(s(\varepsilon)) \otimes wt'(s) \neq \mathbf{0}$, it must be

the case that $in'(s(\varepsilon)) \neq \mathbf{0}$, and thus $s(\varepsilon) = (q, 0)$ for some $q \in Q$. Consider now some $u \in \mathsf{K}^*$ and let $s(u) = (q, i)$. Hence, also $r(u) = q$. Since $wt'(s(u), s(u1), \ldots, s(uk)) \neq \mathbf{0}$, it must be the case that, if $q \in F_i$, then for all $1 \leq j \leq k$ it holds that $s(uj) = (q_j, i+1 \bmod n)$ for some $q_j \in Q$, and if $q \notin F_i$, then $s(uj) = (q_j, i)$. Thus, s satisfies the definition of $f(r)$.

It remains only to show that r is successful (w.r.t. $\mathcal{A}$) iff $f(r)$ is successful (w.r.t. $\mathcal{A}'$). Suppose first that $f(r)$ is successful. Then for every path there are infinitely many nodes labelled with elements of $F' = \{(q, n-1) \mid q \in F_{n-1}\}$. But notice that, according to the way f was defined, if $f(r)(u) \in F'$, then $f(r)(uj)$ is of the form $(q_j, 0)$ for all $1 \leq j \leq k$. All the following nodes in the path will have labels of the form $(\cdot, 0)$ until a state from F_0 is found, in which case the next labels are of the form $(\cdot, 1)$, etc. Thus, to get to another node with label $(q', n-1) \in F'$ on the path, one must first have reached nodes with labels $(q_0, 0), (q_1, 1), \ldots, (q_{n-2}, n-2)$ where $q_i \in F_i$ for $i = 0, \ldots, n-2$. This implies that r is successful.

Conversely, assume that $f(r)$ is not successful. Then there is a path in $f(r)$ on which, from some node on, no element of F' occurs as a label on the path. Since the second component of the node labels can only switch back to 0 when an element of F' is reached, this means that there is an $i_0, 0 \leq i_0 \leq n-1$, such that, from some node on, all the labels on the path have i_0 as their second component. This means, however, that from this node on no element of F_{i_0} occurs in the first component. Consequently, r cannot be successful.

As a consequence of the properties of the function f that we have shown so far, we obtain

$$\begin{aligned}
\|\mathcal{A}\| &= \bigoplus_{r \text{ successful run of } \mathcal{A}} in(r(\varepsilon)) \otimes wt(r) \\
&= \bigoplus_{r \text{ successful run of } \mathcal{A}} in(r(\varepsilon)) \otimes wt(f(r)) \\
&= \bigoplus_{f(r) \text{ successful run of } \mathcal{A}'} in(f(r)(\varepsilon)) \otimes wt(f(r)) \\
&= \bigoplus_{r \text{ successful run of } \mathcal{A}'} in(r(\varepsilon)) \otimes wt(r) = \|\mathcal{A}'\|.
\end{aligned}$$

□

Given a WGBA with m states and n sets of final states, this reduction yields a WBA with $n \cdot m$ states. As described before, computing the behaviour of a WBA requires time polynomial in the size of its state set; in this case, polynomial in $n \cdot m$. Thus, our method computes the behaviour of a WGBA in time polynomial in its number of states and sets of final states.

Let us apply this approach for computing the behaviour of a WGBA to the pinpointing automaton for LTL constructed in Section 4.3. This automaton has exponentially many states in the size n of the input $(\phi, \mathcal{R})$ and linearly many sets of final states in n. Thus, the WBA constructed from the WGBA is of size exponential in n. Overall, the two nested iterations perform exponentially many steps, which leads to an algorithm with a total running time that is exponential in the size of the input.

Corollary 4 *Let ϕ be an LTL formula and $\mathcal{R}$ a set of LTL formulae. A pinpointing formula for $(\phi, \mathcal{R})$ w.r.t. a-unsatisfiability can be computed in time exponential in the size of $(\phi, \mathcal{R})$.*

5.4 An Alternative Approach for Computing the Behaviour

Independently from us, a different algorithm for computing the behaviour of WBAs over distributive lattices was developed by Droste *et.al.* [15]. We will first sketch this alternative approach and then compare it to ours, with special attention to the application in the pinpointing scenario.[12] In the following, we will call our method the *iterative method* and the one from [15] the *prime method*.

The prime method is based on the following property of distributive lattices. Let $(S, \leq_S)$ be a distributive lattice. An element $p \in S$ is called *meet prime* if, for every $t_1, t_2 \in S$, $t_1 \otimes t_2 \leq_S p$ implies that either $t_1 \leq_S p$ or $t_2 \leq_S p$. It is known that any element t of S equals the infimum of all the meet prime elements greater than or equal to t [18]. If one could decide, for a given meet prime element p, whether p is greater than or equal to the behaviour of a weighted automaton, then this behaviour could be readily computed from the outputs of such decisions, as we will show next.

The prime method performs this decision as follows. Let $\mathcal{A} = (Q, in, wt, F)$ be the WBA over the distributive lattice $(S, \leq_S)$ for which we want to compute the behaviour, and let $\mathsf{prime}(S)$ denote the set of all meet prime elements of S. For every meet prime element $p \in \mathsf{prime}(S)$, construct the (unweighted) automaton $\mathcal{A}_p = (Q, \Delta, I, F)$ where:

- $\Delta := \{(q, q_1, \ldots, q_k) \in Q^{k+1} \mid wt(q, q_1, \ldots, q_k) \not\leq_S p\}$;
- $I := \{q \in Q \mid in(q) \not\leq_S p\}$.

It is easy to see that $\mathcal{A}_p$ accepts a non-empty language (i.e., there exists a successful run of $\mathcal{A}_p$ that starts with an initial state) iff there is a successful run r of $\mathcal{A}$ such that $in(r(\varepsilon)) \otimes wt(r) \not\leq_S p$. Equivalently, the language accepted by $\mathcal{A}_p$ is empty iff, for every successful run r of $\mathcal{A}$, it holds that $in(r(\varepsilon)) \otimes wt(r) \leq_S p$. But this means that $\|\mathcal{A}\| \leq_S p$. Thus, if we denote by $\mathcal{L}(\mathcal{A}_p)$ the language accepted by the automaton $\mathcal{A}_p$, we have

$$\|\mathcal{A}\| = \bigotimes_{\{p \in \mathsf{prime}(S) \mid \mathcal{L}(\mathcal{A}_p) = \emptyset\}} p.$$

In the pinpointing application, we use the lattice $\mathbb{B}^{\mathcal{T}}$, where the meet prime elements are exactly all conjunctions of propositional variables in $\mathsf{lab}(\mathcal{T})$.[13] There is then a one-to-one correspondence between the meet prime elements of $\mathbb{B}^{\mathcal{T}}$ and all subsets of axioms appearing in the axiomatic input for which the pinpointing formula is being computed. Take an arbitrary meet prime element p and assume that it corresponds to the set of axioms $\mathcal{T}' \subseteq \mathcal{T}$, i.e., $p = \bigwedge_{t \in \mathcal{T}'} \mathsf{lab}(t)$. The automaton $\mathcal{A}_p$ has a transition $(q, q_1, \ldots, q_k)$ iff

$$\Delta\mathsf{vio}(q, q_1, \ldots, q_k) = wt(q, q_1, \ldots, q_k) \not\leq_{\mathbb{B}^{\mathcal{T}}} p = \bigwedge_{t \in \mathcal{T}'} \mathsf{lab}(t).$$

Since $\Delta\mathsf{vio}(q, q_1, \ldots, q_k) = \bigvee_{\{t \in \mathcal{T} \mid (q, q_1, \ldots, q_k) \notin \Delta\mathsf{res}(t)\}} \mathsf{lab}(t)$, this means that for every $t \in \mathcal{T}'$, $(q, q_1, \ldots, q_k) \in \Delta\mathsf{res}(t)$. But this holds iff $(q, q_1, \ldots, q_k)$ is a transition of $\mathcal{A}_{|\mathcal{T}'}$.

[12] We present only a special case of the algorithm in [15], where we allow only unlabelled trees as inputs. Furthermore, we have exchanged the use of *join prime* elements in [15] with the use of their *meet prime* counterparts. This is justified by duality, and allows for an easier understanding of how this method works in the pinpointing application, and makes it easier to compare it with our approach in this setting.

[13] Recall that the lattice $\mathbb{B}^{\mathcal{T}}$ uses disjunction as its infimum operator, and conjunction as the supremum. Thus, conjunctions of variables are the only elements of the lattice that cannot be written as the infimum (disjunction) of other elements.

Analogously, it is easy to see that a state q is an initial state of $\mathcal{A}_p$ iff it is an initial state of $\mathcal{A}_{|\mathcal{T}'}$. Thus, the automaton $\mathcal{A}_p$ is identical to the $\mathcal{T}'$-restricted subautomaton $\mathcal{A}_{|\mathcal{T}'}$. Consequently, testing the automaton $\mathcal{A}_p$ for emptiness is the same as testing $\mathcal{A}_{|\mathcal{T}'}$ for emptiness, which in turn is just an application of the automata-based decision procedure as a black-box procedure for testing the c-property. One could, of course, also use any other decision procedure for the c-property instead. This shows that the prime method actually corresponds to the naïve black-box approach of testing the c-property for all possible subsets of axioms. Unoptimized, this process will thus always need an exponential number of tests for computing the pinpointing formula. However, this process allows the use of all the optimizations applicable to black-box pinpointing algorithms.

Notice that, in the examples we have presented in this paper (i.e., pinpointing unsatisfiability in $\mathcal{SI}$ and LTL), both the iterative and the prime method have an exponential run time. For the iterative method, we have a bound that is polynomial in the number of states of the constructed automata, but this number is itself exponential in the size of the input. The prime method performs exponentially many emptiness tests, each of which requires exponential time (since it is performed on an exponentially large automaton). Although both approaches result in an exponential-time algorithm in these cases, the bound on the iterative method has the advantage of not depending on the number of meet prime elements of the lattice, as opposed to the prime method. In the case of pinpointing, the lattice has always 2^n meet prime elements, where n is the number of input axioms. If the axiomatic automaton deciding the property has a number of states polynomial in the size of the input, then this exponential number of tests will yield a suboptimal procedure, as demonstrated by the following examples.

Example 7 Assume that we have an input $\mathcal{I}$ and a set of axioms $\mathcal{T} = \{t_0, \ldots, t_{n-1}\}$, and that the c-property is defined as follows: $\mathcal{P}_\mathbf{1} := \{(\mathcal{I}, \mathcal{T}') \mid \mathcal{T}' \subseteq \mathcal{T}, |\mathcal{T}'| > 0\}$. Let each axiom t_i be labelled with the propositional variable p_i. Then a pinpointing formula for $\mathcal{P}_\mathbf{1}$ is given by $\bigvee_{0 \leq i < n} p_i$.

We can construct an axiomatic automaton $(\mathcal{A}_n, \Delta\mathsf{res}, I\mathsf{res})$ for the axiomatized input $(\mathcal{I}, \mathcal{T})$ as follows:

- $\mathcal{A}_n$ is the looping automaton $\mathcal{A}_n := (\{q_0, \ldots, q_{n-1}\}, \Delta, \{q_0\})$;
- $\Delta = \{(q_i, q_{(i+1) \bmod n}) \mid 0 \leq i < n\}$;
- for every $0 \leq j \leq n-1$, $\Delta\mathsf{res}(t_j) = \Delta \setminus \{(q_j, q_{(j+1) \bmod n}\}$;
- for every $t \in \mathcal{T}$, $I\mathsf{res}(t) = \{q_0\}$.

It is easy to see that this axiomatic automaton is correct for the property $\mathcal{P}_\mathbf{1}$. Since $\mathcal{A}_n$ has n states and n transitions, the iterative method needs polynomial time to compute the behaviour of the pinpointing automaton induced by $(\mathcal{A}_n, \Delta\mathsf{res}, I\mathsf{res})$, measured in the number of axioms $n := |\mathcal{T}|$. On the other hand, the unoptimized prime method requires 2^n emptiness tests.

In order to illustrate the working of the iterative methods, we show how it computes the pinpointing formula in this example. The axiomatic automaton $(\mathcal{A}_n, \Delta\mathsf{res}, I\mathsf{res})$ induces the pinpointing automaton $(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})^{pin} = (\{q_0, \ldots, q_{n-1}\}, in, wt)$, where

- $in(q_0) = \bot$ and $in(q_i) = \top$ for all $0 < i < n$; and
- $wt(q_i, q_j)$ equals p_i if $j = (i+1) \mod n$, and $\top$ otherwise.

As this is a weighted looping automaton, the iterative method reduces to an iterated application of the simplified operator $\mathcal{Q}$ described in Section 5.2. Notice that, for every state q_i, there is exactly one transition, namely $(q_i, q_{(i+1) \bmod n})$, having a weight

distinct from $\top$. Hence, for every function $\sigma : Q \to \mathbb{B}^{\mathcal{T}}$ we have:

$$\begin{aligned}\mathcal{Q}(\sigma)(q_i) &= \bigwedge_{0 \le j < n} wt(q_i, q_j) \vee \sigma(q_j) \\ &= wt(q_i, q_{(i+1) \bmod n}) \vee \sigma(q_{(i+1) \bmod n}) = p_i \vee \sigma(q_{(i+1) \bmod n}).\end{aligned}$$

The process starts with the function $\widetilde{\mathbf{1}} : Q \to \mathbb{B}^{\mathcal{T}}$ that maps every state to $\bot$; that is, $\widetilde{\mathbf{1}}(q_i) = \bot$ for all $0 \le i < n$. After the first application of the operator $\mathcal{Q}$, we have $\mathcal{Q}(\widetilde{\mathbf{1}})(q_i) = p_i$ for all $0 \le i < n$ since $p_i \vee \bot$ is equivalent to p_i. Analogously, after m iterations we have, for all $0 \le i < n$, that

$$\mathcal{Q}^m(\widetilde{\mathbf{1}})(q_i) = \bigvee_{0 \le j < m} p_{(i+j) \bmod n}.$$

This process reaches a fixpoint when $m = n$, in which case every state q_i is mapped to the formula $\bigvee_{0 \le j < n} p_j$. Thus, the behaviour of $(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})^{pin}$ is

$$\begin{aligned}\|(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})^{pin}\| &= \textstyle\bigwedge_{0 \le i < n} in(q_i) \vee \mathcal{Q}^n(\widetilde{\mathbf{1}})(q_i) \\ &= in(q_0) \vee \mathcal{Q}^n(\widetilde{\mathbf{1}})(q_0) \\ &= \mathcal{Q}^m(\widetilde{\mathbf{1}})(q_0) = \textstyle\bigvee_{0 \le j < n} p_j,\end{aligned}$$

which is a pinpointing formula.

Our second example shows that this difference in the execution times of the two methods occurs also for more elaborate properties whose automata decision procedure uses a Büchi acceptance condition.

Example 8 Let Q be an infinite set of states and let the set of inputs $\mathfrak{I}$ be the set of all generalized Büchi automata using states from Q, and the set of axioms be $\mathfrak{T} := \mathsf{Q}^{k+1}$. That is, we use the transitions in $\mathcal{A}$ as axioms of our property. We define the c-property $\mathcal{P}_{\mathbf{2}}$ as the set of all tuples $(\mathcal{A}, \Theta)$ where $\mathcal{A} = (Q, \Delta, I, F_1, \ldots, F_n)$ is a generalized Büchi automaton in $\mathfrak{I}$, and $\Theta \subseteq \mathfrak{T}$ is such that $(Q, \Delta \setminus \Theta, I, F_1, \ldots, F_n)$ has no successful run r with $r(\varepsilon) \in I$. Intuitively, the axioms tell which transitions are *disallowed* in the input automaton $\mathcal{A}$. The c-property is satisfied whenever we remove enough transitions (by adding them to the axiom set) to avoid any successful run whose root is labelled with an initial state. It is easy to see that the axiomatic automaton $(\mathcal{A}, \Delta\mathsf{res}, I\mathsf{res})$ where $\Delta\mathsf{res}(t) = \Delta \setminus \{t\}$ and $I\mathsf{res}(t) = Q$ for all $t \in \Theta$ is correct for the property $\mathcal{P}$ and the axiomatized input $(\mathcal{A}, \Theta)$. As we have seen, the iterative method requires time polynomial in the number of states $|Q|$ of this axiomatic automaton to compute the pinpointing formula for this property. On the other hand, the prime method needs $2^{|\Theta|}$ emptiness tests, each polynomial on $|Q|$. We thus have an exponential increase in execution time, when compared to the iterative method.

One advantage of the prime method is that it can easily be generalized to more complex automata models. For instance, it is shown in [15] how the same idea works in the presence of a more complex acceptance condition, known as the Muller condition. Also note that the prime method can possibly be optimized using the ideas underlying the known optimizations of black-box pinpointing procedures, not just in the case of applying it to pinpointing, but also in a more general setting.

6 Conclusions

We have introduced a general framework for extending decision procedures based on the construction of generalized Büchi automata to pinpointing algorithms. This framework can elegantly deal with DLs for which tableau-based decision procedures require sophisticated blocking conditions, and to which consequently the general approach for extending tableau-based decision procedures to pinpointing algorithms introduced in [5] does not apply. Our framework assumes that one can describe the influence of axioms in a c-property by restricting the sets of transitions and initial states of the automaton. One could imagine that in some cases the axioms might also have an influence on the final states. While it should not be hard to integrate this into our framework, we have not investigated this since none of the c-properties we have considered required such a modification of the sets of final states.

Our framework is based on the use of weighted automata working on infinite trees, whose study has only recently begun. One of the main contributions of this paper is an approach for computing the behaviour of such automata with a run time that is polynomial in the size of the automaton and independent of the size of the underlying distributive lattice. An interesting topic for future work is to check whether our iterative approach can be adapted such that it also works in cases where the weighted automaton is not explicitly given, but rather computed on-the-fly. Finally, it would also be interesting to know how to adapt our iterative method such that it can compute the behaviour of weighted automata working on infinite trees that use more complex acceptance conditions for runs, such as the Muller or the Rabin condition.

References

1. Franz Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, 1991.
2. 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.
3. Franz Baader, Jan Hladik, and Rafael Peñaloza. Automata can show PSPACE results for description logics. *Information and Computation*, 206(9–10):1045–1056, 2008.
4. Franz Baader and Bernhard Hollunder. Embedding defaults into terminological knowledge representation formalisms. *J. of Automated Reasoning*, 14:149–180, 1995.
5. Franz Baader and Rafael Peñaloza. Axiom pinpointing in general tableaux. In *Proc. of the Int. Conf. on Analytic Tableaux and Related Methods (TABLEAUX 2007)*, volume 4548 of *Lecture Notes in Artificial Intelligence*, pages 11–27. Springer-Verlag, 2007.
6. Franz Baader and Rafael Peñaloza. Automata-based axiom pinpointing. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2008)*, volume 5195 of *Lecture Notes in Artificial Intelligence*, pages 226–241. Springer-Verlag, 2008.
7. Franz Baader and Rafael Peñaloza. Blocking and pinpointing in forest tableaux. LTCS-Report LTCS-08-02, Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, Germany, 2008. See http://lat.inf.tu-dresden.de/research/reports.html.
8. Franz Baader and Rafael Peñaloza. Axiom pinpointing in general tableaux. *Journal of Logic and Computation*, 2009. To appear.
9. Franz Baader and Ulrike Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.
10. Franz Baader and Boontawee Suntisrivaraporn. Debugging SNOMED CT using axiom pinpointing in the description logic $\mathcal{EL}^+$. In *Proc. of the International Conference on*

Representing and Sharing Knowledge Using SNOMED (KR-MED'08), Phoenix, Arizona, 2008.
11. Franz Baader and Stephan Tobies. The inverse method implements the automata approach for modal satisfiability. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 92–106. Springer-Verlag, 2001.
12. Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, Cambridge, Massachusetts, 2008.
13. Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Reasoning in expressive description logics with fixpoints based on automata on infinite trees. In *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI'99)*, pages 84–89, 1999.
14. Diego Calvanese, Giuseppe DeGiacomo, and Maurizio Lenzerini. 2ATAs make DLs easy. In *Proc. of the 2002 Description Logic Workshop (DL 2002)*, pages 107–118. CEUR Electronic Workshop Proceedings, http://ceur-ws.org/Vol-53/, 2002.
15. Manfred Droste, Werner Kuich, and George Rahonis. Multi-valued MSO logics over words and trees. *Fundamenta Informaticae*, 84(3,4):305–327, 2008.
16. Manfred Droste and George Rahonis. Weighted automata and weighted logics on infinite words. In Oscar H. Ibarra and Zhe Dang, editors, *Developments in Language Theory*, volume 4036 of *Lecture Notes in Computer Science*, pages 49–58. Springer-Verlag, 2006.
17. D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proc. of the 7th ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages (POPL'80)*, pages 163–173, 1980.
18. G. Grätzer. *General Lattice Theory*. Birkhäuser, second edition edition, 1998.
19. Volker Haarslev and Ralf Möller. RACER system description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 701–705. Springer-Verlag, 2001.
20. Ian Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647, 1998.
21. Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.
22. Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, and Evren Sirin. Finding all justifications of OWL DL entailments. In *Proc. of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007*, volume 4825 of *Lecture Notes in Computer Science*, pages 267–280, Busan, Korea, 2007. Springer-Verlag.
23. Orna Kupferman and Yoad Lustig. Lattice automata. In Byron Cook and Andreas Podelski, editors, *8th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'07)*, volume 4349 of *Lecture Notes in Computer Science*, pages 199–213. Springer-Verlag, 2007.
24. Orna Kupferman and Moshe Vardi. Safraless decision procedures. In *Proc. of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS05)*, pages 531–542. IEEE Computer Society, 2005.
25. Kevin Lee, Thomas Meyer, and Jeff Z. Pan. Computing maximally satisfiable terminologies for the description logic $\mathcal{ALC}$ with GCIs. In *Proc. of the 2006 Description Logic Workshop (DL 2006)*, volume 189 of *CEUR Electronic Workshop Proceedings*, 2006.
26. Carsten Lutz and Ulrike Sattler. The complexity of reasoning with Boolean modal logic. In Frank Wolter, Heinrich Wansing, Maarten de Rijke and Michael Zakharyaschev, editors, *Advances in Modal Logic, Volume 3*, pages 329–348. CSLI Publications, 2001.
27. Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Debugging OWL ontologies. In Allan Ellis and Tatsuya Hagino, editors, *Proc. of the 14th International Conference on World Wide Web (WWW'05)*, pages 633–640. ACM, 2005.
28. Amir Pnueli. The temporal logic of programs. In *Proc. of the 18th Annual Symp. on the Foundations of Computer Science (FOCS'77)*, pages 46–57, 1977.
29. Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Proc. of the 16th Annual ACM Symp. on Principles of Programming Languages (POPL89)*, pages 319–327. ACM, 1989.
30. M. O. Rabin. Weakly definable relations and special automata. In Y. Bar-Hillel, editor, *Proc. of Symp. on Mathematical Logic and Foundations of Set Theory*, pages 1–23. North-Holland Publ. Co., Amsterdam, 1970.

31. George Rahonis. Weighted Muller tree automata and weighted logics. *Journal of Automata, Languages and Combinatorics*, 12(4):455–483, 2007.
32. R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.
33. Stefan Schlobach. Diagnosing terminologies. In Manuela M. Veloso and Subbarao Kambhampati, editors, *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, pages 670–675. AAAI Press/The MIT Press, 2005.
34. Stefan Schlobach and Ronald Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In Georg Gottlob and Toby Walsh, editors, *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003)*, pages 355–362, Acapulco, Mexico, 2003. Morgan Kaufmann, Los Altos.
35. Stefan Schlobach, Zhisheng Huang, Ronald Cornet, and Frank Harmelen. Debugging incoherent terminologies. *Journal of Automated Reasoning*, 39(3):317–349, 2007.
36. Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.
37. Helmut Seidl. Finite tree automata with cost functions. *Theor. Comput. Sci.*, 126(1):113–142, 1994.
38. Evren Sirin and Bijan Parsia. Pellet: An OWL DL reasoner. In *Proc. of the 2004 Description Logic Workshop (DL 2004)*, pages 212–213, 2004.
39. Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.
40. Moshe Y. Vardi and Pierre Wolper. Automata-theoretic techniques for modal logics of programs. In *Proc. of the 16th ACM SIGACT Symp. on Theory of Computing (STOC'84)*, pages 446–455, 1984.
41. Moshe Y. Vardi and Pierre Wolper. Automata-theoretic techniques for modal logics of programs. *J. of Computer and System Sciences*, 32:183–221, 1986. A preliminary version appeared in *Proc. of the 16th ACM SIGACT Symp. on Theory of Computing (STOC'84)*.
42. Pierre Wolper, Moshe Y. Vardi, and A. Prasad Sistla. Reasoning about infinite computation paths. In *Proc. of the 24th Annual Symposium of Foundations of Computer Science (SFCS'83)*, pages 185–194, Washington, DC, USA, 1983. IEEE Computer Society.