Automata-Based Axiom Pinpointing

Franz Baader[1,⋆] and Rafael Peñaloza[2,⋆⋆]

[1] Theoretical Computer Science, TU Dresden, Germany
baader@tcs.inf.tu-dresden.de
[2] Intelligent Systems, Uni. Leipzig, Germany
penaloza@informatik.uni-leipzig.de

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 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.

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 [15] 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 [14, 13, 22] 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 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 (see [9]) among the more than 350,000 terminological axioms of SNOMED without support by an automated reasoning tool is not easy.

⋆ Partially supported by NICTA, Canberra Research Lab.
⋆⋆ Funded by the German Research Foundation (DFG) under grant GRK 446.
[1] see http://www.ihtsdo.org/our-standards/

A. Armando, P. Baumgartner, and G. Dowek (Eds.): IJCAR 2008, LNCS 5195, pp. 226–241, 2008.

Axiom pinpointing [19] 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). Most of the pinpointing algorithms described in the DL literature (e.g., [4, 19, 18, 17, 16]) are obtained as extensions of tableau-based reasoning algorithms [8] 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 [16], 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 [8], 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 [7, 6], 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 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 [10]. 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,[2] whose so-called behaviour yields a pinpointing formula, i.e., a monotone Boolean formula that encodes all the MinAs of Γ. To obtain an actual pinpointing algorithm, we must develop an algorithm for computing the behaviour of weighted tree automata working on infinite trees. We will use the DL $\mathcal{SI}$, which extends the basic DL $\mathcal{ALC}$ [20] with transitive and inverse roles, to illustrate our 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 [8]. On the other hand, the extension of their approach to $\mathcal{SI}$ is mentioned as an open problem in [16].

[2] Although weighted automata working on *finite trees* [21] and weighted automata working on *infinite words* [11] have been considered before, this appears to be the first use of weighted automata working on *infinite trees*.

2 Preliminaries

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

2.1 The Description Logic $\mathcal{SI}$

As mentioned above, $\mathcal{SI}$ extends $\mathcal{ALC}$ 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 "hardcoding" 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 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 a $\mathcal{ALC}$ 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:*

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

In this paper we restrict the 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 tree automaton whose runs are exactly the Hintikka trees for C and $\mathcal{T}$ (see Section 4.2), and then tests this automaton for emptiness.

2.2 Basic Definitions for Pinpointing

Following [5], we define pinpointing not for a specific DL and inference problem, but rather in a more general setting. The type of inference problem that we will consider is deciding a so-called c-property for a given set of axiomatized inputs.

Definition 4 (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 $\mathcal{T}' \supseteq \mathcal{T}$.*

For example, let $\mathfrak{I}$ consists 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: $\mathcal{P} = \{(C, \mathcal{T}) \mid C \text{ is unsatisfiable w.r.t. } \mathcal{T}\}$.

Definition 5. *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 $\mathsf{MIN}_{\mathcal{P}(\Gamma)} = \emptyset$. In our example, consider the axiomatized input $\Gamma = (A \sqcap \forall r.C, \mathcal{T})$ where $\mathcal{T}$ consists of

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

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

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 labeled with a unique propositional variable, $\mathsf{lab}(t)$. Let $\mathsf{lab}(\mathcal{T})$ be the set of all propositional variables labeling 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 $\mathcal{T}_\mathcal{V} = \{t \in \mathcal{T} \mid \mathsf{lab}(t) \in \mathcal{V}\}$.

Definition 6 (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 $\mathcal{V} \subseteq \mathsf{lab}(\mathcal{T})$: $(\mathcal{I}, \mathcal{T}_\mathcal{V}) \in \mathcal{P}$ iff $\mathcal{V}$ satisfies ϕ.*

In our 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 for Γ w.r.t. $\mathcal{P}$.

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 blowup. 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 blowup happens, then the pinpointing formula actually yields a compact representation of all MinAs.

3 Looping Tree Automata

In this section, we introduce both unweighted and weighted looping 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 labeled trees, we will refer to the labeling 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 unlabeled 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^*.

Definition 7 (looping tree automaton). *A* looping tree automaton *for arity k is a tuple (Q, Δ, I), where Q is a finite set of* states*, $\Delta \subseteq Q^{k+1}$ is the* transition relation*, and $I \subseteq Q$ is the set of* initial states*. A* run *of this automaton on the unlabeled tree K^* is a labeled k-ary tree $r : K^* \to Q$ such that $\overrightarrow{r(u)} \in \Delta$ for all $u \in K^*$. This run is* successful *if $r(\varepsilon) \in I$. The* emptiness problem *for looping tree automata for arity k is the problem of deciding whether a given such automaton has a successful run or not.*

The emptiness problem for looping tree automata can be decided using time polynomial in the size of the automaton. The decision procedure computes all the *bad* states, i.e., states that do not occur in any run, in a *bottom-up* fashion [24, 10]: starting with the empty set as initial set of known bad states, the algorithm iteratively adds states to this set as follows: if all the transitions starting from a state q contain a known bad state, then q is also known to be bad. Note that the condition for adding a bad state q also applies to states that have no transition. Obviously, the set of known bad states becomes stable after at most $|Q|$ iterations, and it is easy to show that the set of states obtained this way is indeed the set of all bad states. The automaton has a successful run iff there is an initial state that is *not* bad.

We will later extend automata-based decision procedures into algorithms that compute pinpointing formulae by transforming looping tree automata into weighted looping automata. The weights of such automata come from a complete distributive lattice [12].

Definition 8 (complete distributive lattice). *A* complete distributive lattice *is a partially ordered set $(S, \leq_S)$ such that infima and suprema of arbitrary subsets of S always exist and distribute over each other.*

In the following, we will often simple use the carrier set S to denote the complete distributive lattice $(S, \leq_S)$. The infimum (supremum) of a subset $T \subseteq S$ will be denoted by $\bigotimes_{t \in T} t$ ($\bigoplus_{t \in T} t$). 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 complete distributive lattice is stronger than the one usually encountered in the literature on weighted automata. In fact, for automata working on finite trees, it is sufficient to assume that the weights come from a so-called semiring [21]. 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 [11]. The additional properties imposed by our requirement to have a complete distributive lattice (in particular, the idempotency of product and sum) are used to show that we can actually compute the behaviour of our weighted looping automata (see Section 5).

Definition 9 (weighted looping automaton). *Let S be a complete distributive lattice. A* weighted looping automaton (WLA) *on S for arity k is a tuple $\mathcal{A} = (Q, in, wt)$ where Q is a finite set of* states*, $in : Q \to S$ is the* initial distribution*, and $wt : Q^{k+1} \to S$ assigns* weights *to transitions. A* run *of $\mathcal{A}$ is a labeled tree $r : K^* \to Q$. The* weight *of this run is $wt(r) = in(r(\varepsilon)) \otimes \bigotimes_{u \in K^*} wt(\overrightarrow{r(u)})$. The* behaviour *of the automaton $\mathcal{A}$ is $\|\mathcal{A}\| := \bigoplus_{r:K^* \to Q} wt(r)$.*

For example, the Boolean semiring $\mathbb{B} = (\{0,1\}, \wedge, \vee, 1, 0)$ is a complete 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 the least element $\mathbf{0}$, and 0 is the greatest element $\mathbf{1}$. We can easily transform a given looping tree automaton $\mathcal{A} = (Q, \Delta, I)$ into a WLA $\mathcal{A}_w$ on $\mathbb{B}$ such that the behaviour of $\mathcal{A}_w$ is 0 iff $\mathcal{A}$ has a successful run. In $\mathcal{A}_w$, the initial distribution maps initial states to 0 and all other states to 1; a tuple in Q^{k+1} receives weight 0 if it belongs to Δ, and weight 1 otherwise.

The bottom-up emptiness test for looping tree 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. 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. In the next step of our iteration, q becomes known to be bad, i.e., $\mathsf{bad}_{i+1}(q)$ is set to 1, if every transition starting from q leads to at least one state already know to be bad. Thus, we have

$$\mathsf{bad}_{i+1}(q) = \bigwedge_{(q,q_1,\ldots,q_k) \in Q^{k+1}} \left(wt(q, q_1, \ldots, q_k) \vee \bigvee_{j=1}^{k} \mathsf{bad}_i(q_j) \right). \tag{2}$$

The function bad is the limit of this iteration, which is reached after at most $|Q|$ iterations. It is easy to see that the behaviour of $\mathcal{A}_w$ is then $\bigwedge_{q \in Q} in(q) \vee \mathsf{bad}(q)$. In Section 5, we will see that the behaviour of a WLA can always be computed by such an iteration.

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 $\mathcal{SI}$.

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 looping tree 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 $\mathcal{T}' \subseteq \mathcal{T}$, which can be obtained by appropriately restricting the transitions of $\mathcal{A}_\Gamma$. To be more precise, let $\mathcal{A} = (Q, \Delta, I)$ be a looping tree automaton for arity k and $\Gamma = (\mathcal{I}, \mathcal{T})$ an axiomatized input. A function $\mathsf{res} : \mathcal{T} \to \mathscr{P}(Q^{k+1})$ is called a *restricting function.* The restricting function res can be extended to sets of axioms $\mathcal{T}' \subseteq \mathcal{T}$ as follows: $\mathsf{res}(\mathcal{T}') := \bigcap_{t \in \mathcal{T}'} \mathsf{res}(t)$. For $\mathcal{T}' \subseteq \mathcal{T}$, the *$\mathcal{T}'$-restricted subautomaton of $\mathcal{A}$ w.r.t.* res is defined as $\mathcal{A}_{|\mathcal{T}'} := (Q, \Delta \cap \mathsf{res}(\mathcal{T}'), I)$.

Definition 10 (axiomatic automaton). *Let $\mathcal{A} = (Q, \Delta, I)$ be a looping tree automaton for arity k, $\Gamma = (\mathcal{I}, \mathcal{T})$ an axiomatized input, and $\mathsf{res} : \mathcal{T} \to \mathscr{P}(Q^{k+1})$ a restricting function. Then we call $(\mathcal{A}, \mathsf{res})$ an* axiomatic automaton *for Γ. Given a c-property $\mathcal{P}$, we say that $(\mathcal{A}, \mathsf{res})$ 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.*

Given a correct axiomatic automaton, we show how to transform it into a weighted looping automaton whose behaviour is a pinpointing formula. 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 complete distributive lattice, where the partial order is defined as $\phi \leq \psi$ iff $\psi \to \phi$ is valid. 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 $\mathbf{1}$.

Definition 11 (pinpointing automaton). *Let $(\mathcal{A}, \mathsf{res})$, with $\mathcal{A} = (Q, \Delta, I)$, be an axiomatic automaton for $\Gamma = (\mathcal{I}, \mathcal{T})$. The* violating function $\mathsf{vio} : Q^{k+1} \to \mathbb{B}^{\mathcal{T}}$ *is given by*

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

The pinpointing automaton *induced by $(\mathcal{A}, \mathsf{res})$ w.r.t. $\mathcal{T}$ is the WLA $(\mathcal{A}, \mathsf{res})^{\mathsf{pin}} = (Q, in, wt)$ on $\mathbb{B}^{\mathcal{T}}$, where*

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

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

It is easy to see that, if $r : K^* \xrightarrow{\longrightarrow} Q$ is a successful run of $\mathcal{A}$, then its weight is given by $wt(r) = \bigvee_{u \in K^*} \mathsf{vio}(\overrightarrow{r(u)})$; otherwise, $wt(r) = \top$. Intuitively, the violating function 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. Removing all the axioms appearing in that formula would yield a subset of axioms which actually allows for this run; and hence, due to correctness, the property does not hold anymore. Conjoining this information for all possible runs leads us to a pinpointing formula.

Theorem 1. *Let $\mathcal{P}$ be a c-property, $\Gamma = (\mathcal{I}, \mathcal{T})$ an axiomatized input, and $(\mathcal{A}, \mathsf{res})$ a correct axiomatic automaton for Γ w.r.t. $\mathcal{P}$. Then $\|(\mathcal{A}, \mathsf{res})^{\mathsf{pin}}\|$ 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 $\|(\mathcal{A}, \mathsf{res})^{\mathsf{pin}}\|$ 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 $(\mathcal{A}, \mathsf{res})$ is correct for Γ w.r.t. $\mathcal{P}$, there must be a successful run r of $\mathcal{A}_{|\mathcal{T}_\mathcal{V}}$. Consequently, $\overrightarrow{r(u)} \in \mathsf{res}(\mathcal{T}_\mathcal{V})$ holds for every $u \in K^*$, and thus $\mathcal{V}$ cannot satisfy $\mathsf{vio}(\overrightarrow{r(u)})$, 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 $wt(r) = \bigvee_{u \in K^*} \mathsf{vio}(\overrightarrow{r(u)})$. Thus, $\mathcal{V}$ does not satisfy $wt(r)$. But then $\mathcal{V}$ also cannot satisfy $\bigwedge_{r:K^* \to Q} wt(r) = \|(\mathcal{A}, \mathsf{res})^{\mathsf{pin}}\|$.

Conversely, if $\mathcal{V}$ does not satisfy $\|(\mathcal{A}, \mathsf{res})^{\mathsf{pin}}\| = \bigwedge_{r:K^* \to Q} wt(r)$, then there must exist a run r such that $\mathcal{V}$ does not satisfy $wt(r)$. This implies that $\overrightarrow{r(u)} \in \mathsf{res}(\mathcal{T}_\mathcal{V})$ for all $u \in K^*$. Consequently, r is a successful run of $\mathcal{A}_{|\mathcal{T}_\mathcal{V}}$, which shows $(\mathcal{I}, \mathcal{T}_\mathcal{V}) \notin \mathcal{P}$. □

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 $(\mathcal{A}_{C,\mathcal{T}}, \mathsf{res}_{C,\mathcal{T}})$ that is correct for $(C, \mathcal{T})$ w.r.t. unsatisfiability. For this purpose, we must adapt the known construction of a looping tree automaton for $\mathcal{SI}$ [3] such that it yields an axiomatic automaton.[3]

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 labeling every node with the "relevant" concept descriptions to which it belongs.

As usual, we assume in the following that all concept descriptions are in *negation normal form* (NNF); that is, 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

[3] 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.

$\mathcal{ALC}$ 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.

Definition 12 (Hintikka set). *A set $H \subseteq \mathsf{sub}(C, \mathcal{T}) \cup \mathsf{rol}(C, \mathcal{T})$ 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 labeled 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 13 (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 $\exists r.D \in \mathsf{sub}(C, \mathcal{T})$: (i) 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 $\forall r.E \in H_0$. (ii) 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 14 (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 $(\mathcal{A}_{C,\mathcal{T}}, \mathsf{res}_{C,\mathcal{T}})$ has as its first component the looping tree 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 restricting function $\mathsf{res}_{C,\mathcal{T}}$ *maps each axiom* $t \in \mathcal{T}$ *to the set of all tuples in* Δ *that are compatible with* t.

Correctness of this automaton construction can be shown by an 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* $(\mathcal{A}_{C,\mathcal{T}}, \mathsf{res}_{C,\mathcal{T}})$ *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 $(\mathcal{A}_{C,\mathcal{T}}, \mathsf{res}_{C,\mathcal{T}})^{\mathsf{pin}}$ induced by $(\mathcal{A}_{C,\mathcal{T}}, \mathsf{res}_{C,\mathcal{T}})$ in order to obtain a pinpointing formula for $(C, \mathcal{T})$ w.r.t. unsatisfiability. In the next section, we show how the behaviour of a weighted looping automaton on a complete distributive lattice can be computed.

5 Computing the Behaviour of a WLA

Clearly, the naive approach that directly uses the definition of the behaviour by first computing and then adding up the weights of all runs would not produce a result in finite time since there are infinitely many runs, which are themselves infinite. Instead, we will use a bottom-up method for computing the behaviour, which generalizes the approach sketched in Section 3 for the special case of weighted automata simulating unweighted ones. In the following, we assume that $\mathcal{A} = (Q, in, wt)$ is an arbitrary, but fixed, WLA on the complete distributive lattice S.

We will show that the WLA $\mathcal{A}$ induces a monotone operator $\mathcal{O} : S^Q \to S^Q$, 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 $\leq_S$ on S can be transferred to S^Q in the usual way, by applying it componentwise: if $\sigma, \sigma' \in S^Q$, then $\sigma \leq_{S^Q} \sigma'$ 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 complete distributive lattice S. 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 $\mathbf{0}$ ($\mathbf{1}$).

Following the idea of Equation (2), the operator $\mathcal{O}$ is defined as follows for every $\sigma \in S^Q$:

$$\mathcal{O}(\sigma)(q) = \bigoplus_{(q,q_1,\ldots,q_k) \in Q^{k+1}} wt(q, q_1, \ldots, q_k) \otimes \bigotimes_{j=1}^{k} \sigma(q_j). \tag{3}$$

Lemma 1. *The operator* $\mathcal{O}$ *is continuous, i.e., for any increasing sequence* $\sigma_0 \leq_{S^Q} \sigma_1 \leq_{S^Q} \sigma_2 \leq_{S^Q} \ldots$ *we have* $\bigoplus_{i \geq 0} \mathcal{O}(\sigma_i) = \mathcal{O}(\bigoplus_{i \geq 0} \sigma_i)$.

Proof. For any sequence $\sigma_0, \sigma_1, \sigma_2, \ldots$ of elements of S^Q we have

$$\begin{aligned}
\bigoplus_{i\geq 0} \mathcal{O}(\sigma_i)(q) &= \bigoplus_{i\geq 0} \bigoplus_{(q,q_1,\ldots,q_k)\in Q^{k+1}} wt(q,q_1,\ldots,q_k) \otimes \bigotimes_{j=1}^{k} \sigma_i(q_j) \\
&= \bigoplus_{(q,q_1,\ldots,q_k)\in Q^{k+1}} \bigoplus_{i\geq 0} wt(q,q_1,\ldots,q_k) \otimes \bigotimes_{j=1}^{k} \sigma_i(q_j) \\
&= \bigoplus_{(q,q_1,\ldots,q_k)\in Q^{k+1}} wt(q,q_1,\ldots,q_k) \otimes \bigoplus_{i\geq 0} \bigotimes_{j=1}^{k} \sigma_i(q_j) \qquad (4) \\
&= \bigoplus_{(q,q_1,\ldots,q_k)\in Q^{k+1}} wt(q,q_1,\ldots,q_k) \otimes \bigotimes_{j=1}^{k} \bigoplus_{i\geq 0} \sigma_i(q_j) \qquad (5) \\
&= \mathcal{O}(\bigoplus_{i\geq 0} \sigma_i),
\end{aligned}$$

where Identities (4) and (5) are a consequence of distributivity. □

By Tarski's fixpoint theorem [23], this implies that $\mathcal{O}$ has $\bigotimes_{n\geq 0} \mathcal{O}^n(\widetilde{\mathbf{1}})$ as its greatest fixpoint (gfp). If S happens to be finite, and hence also S^Q is finite, this gfp is obviously reached after finitely many iterations, i.e., there exists a smallest $m, 0 \leq m \leq |S|^{|Q|}$ such that $\mathcal{O}^m(\widetilde{\mathbf{1}}) = \mathcal{O}^{m+1}(\widetilde{\mathbf{1}})$, and for this m we have $\bigotimes_{n\geq 0} \mathcal{O}^n(\widetilde{\mathbf{1}}) = \mathcal{O}^m(\widetilde{\mathbf{1}})$. Note that the complete distributive lattice $\mathbb{B}^{\mathcal{T}}$ that we have used in our pinpointing automaton is actually finite. Nevertheless, we will not make a finiteness assumption on the complete distributive lattice S when showing that the behaviour of any WLA on S can effectively be computed.

Next, we show how the gfp of $\mathcal{O}$ relates to the behaviour of the given WLA $\mathcal{A}$. Let $K_q^* := \{r : K^* \to Q \mid r(\varepsilon) = q\}$ denote the set of all runs whose root is labeled with q. Consider the function $\sigma^{\|} \in S^Q$ where $\sigma^{\|}(q) := \bigoplus_{r\in K_q^*} \bigotimes_{u\in K^*} wt(\overrightarrow{r(u)})$. Given this function, we can obtain the behaviour of the WLA $\mathcal{A}$ as follows:

Lemma 2. $\|\mathcal{A}\| = \bigoplus_{q\in Q} in(q) \otimes \sigma^{\|}(q)$.

It turns out that $\sigma^{\|}$ is in fact the greatest fixpoint of $\mathcal{O}$. We will prove this with the help of so-called partial runs. For $n \geq 0$ define $K^{\leq n} := \bigcup_{i=0}^{n} K^i$, and $K^{\leq -1} := \emptyset$. A *partial run of depth* m is simply a mapping $r : K^{\leq m} \to Q$. The set of partial runs with root label $q \in Q$ is denoted by $K_q^{\leq m}$.

Lemma 3. *For all $n \geq 0$ and $q \in Q$ it holds that*

$$\mathcal{O}^n(\widetilde{\mathbf{1}})(q) = \bigoplus_{r\in K_q^{\leq n}} \bigotimes_{u\in K^{\leq n-1}} wt(\overrightarrow{r(u)}).$$

Proof. The proof is by induction on n. For $n = 0$, we have that, on the one hand, $\mathcal{O}^0(\widetilde{\mathbf{1}})(q) = \widetilde{\mathbf{1}}(q) = \mathbf{1}$. On the other hand, since $K^{\leq -1} = \emptyset$, we have

$\bigotimes_{u \in K^{\leq -1}} wt(\overrightarrow{r(u)}) = \mathbf{1}$. Now, assume that the above identity holds for n.

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

$$= \bigoplus_{(q_1,\ldots,q_k)\in Q^k} wt(q, q_1, \ldots, q_k) \otimes \bigotimes_{j=1}^{k} \bigoplus_{r_j \in K_{q_j}^{\leq n}} \bigotimes_{u \in K^{\leq n-1}} wt(\overrightarrow{r_j(u)}) \tag{7}$$

$$= \bigoplus_{(q_1,\ldots,q_k)\in Q^k} wt(q, q_1, \ldots, q_k) \otimes \bigoplus_{r_1 \in K_{q_1}^{\leq n}, \ldots, r_k \in K_{q_k}^{\leq n}} \bigotimes_{j=1}^{k} \bigotimes_{u \in K^{\leq n-1}} wt(\overrightarrow{r_j(u)}) \tag{8}$$

$$= \bigoplus_{(q_1,\ldots,q_k)\in Q^k} \bigoplus_{r_1 \in K_{q_1}^{\leq n}, \ldots, r_k \in K_{q_k}^{\leq n}} wt(q, q_1, \ldots, q_k) \otimes \bigotimes_{j=1}^{k} \bigotimes_{u \in K^{\leq n-1}} wt(\overrightarrow{r_j(u)}) \tag{9}$$

$$= \bigoplus_{r \in K_q^{\leq n+1}} wt(\overrightarrow{r(\varepsilon)}) \otimes \bigotimes_{j=1}^{k} \bigotimes_{u \in K^{\leq n-1}} wt(\overrightarrow{r(ju)}) \tag{10}$$

$$= \bigoplus_{r \in K_q^{\leq n+1}} wt(\overrightarrow{r(\varepsilon)}) \otimes \bigotimes_{v \in K^{\leq n} \setminus \{\varepsilon\}} wt(\overrightarrow{r(v)})$$

$$= \bigoplus_{r \in K_q^{\leq n+1}} \bigotimes_{u \in K^{\leq n}} wt(\overrightarrow{r(u)})$$

Identity (6) employs the definition of $\mathcal{O}$, and (7) the induction hypothesis. Identity (8) uses the fact that S^Q is a distributive lattice, which allows us to multiply out, and Identity (9) multiplies $wt(q, q_1, \ldots, q_k)$ in. Identity (10) simplifies the two sums by constructing a run of larger depth. Instead of considering the transition $(q, q_1, \ldots, q_k)$ and then runs of depth n starting with $q_1, \ldots, q_k$, we simply take the corresponding run of depth $n+1$ starting at q. This run labels the root with q and the successor node j of the root with q_j. Below j we have the former run starting with q_j. The remaining identities are trivial. This completes the proof of the lemma. □

Theorem 3. *The mapping $\sigma^{\|}$ is the greatest fixpoint of $\mathcal{O}$.*

Proof. By Tarski's fixpoint theorem, we know that the greatest fixpoint of $\mathcal{O}$ is $\bigotimes_{n\geq 0} \mathcal{O}^n(\widetilde{\mathbf{1}})$. From Lemma 3, we then obtain

$$\bigotimes_{n\geq 0} \mathcal{O}^n(\widetilde{\mathbf{1}})(q) = \bigotimes_{n\geq 0} \bigoplus_{r \in K_q^{\leq n}} \bigotimes_{u \in K^{\leq n-1}} wt(\overrightarrow{r(u)}) = \bigotimes_{n\geq 0} \bigoplus_{r \in K_q^{*}} \bigotimes_{u \in K^{\leq n-1}} wt(\overrightarrow{r(u)}) =$$

$$= \bigoplus_{r \in K_q^{*}} \bigotimes_{n\geq 0} \bigotimes_{u \in K^{\leq n-1}} wt(\overrightarrow{r(u)}) = \bigoplus_{r \in K_q^{*}} \bigotimes_{u \in K^{*}} wt(\overrightarrow{r(u)}) = \sigma^{\|}(q) \quad \square$$

This theorem, along with Lemma 2, allows us to compute the behaviour of weighted looping automata in a bottom-up fashion. If S is finite, then it is

obvious how to compute the fixpoint $\sigma^{\|}$: we know that there exists a least $m \leq |S|^{|Q|}$ such that $\mathcal{O}^m(\widetilde{\mathbf{1}}) = \mathcal{O}^{m+1}(\widetilde{\mathbf{1}})$, and for this m we have $\mathcal{O}^m(\widetilde{\mathbf{1}}) = \sigma^{\|}$. If S is infinite, then it is not a priori clear that the fixpoint can be reached after finitely many iterations. We will now show that it is actually sufficient to apply the operator only $|Q|+1$ times. Note that this result is also useful for the finite case since the bound $|Q|+1$ greatly improves on the generic bound $|S|^{|Q|}$.

Definition 15 (m-complete). *A WLA $\mathcal{A}$ is* m-complete *if, for every partial run $r : K^{\leq m} \to Q$ of depth m, there is a run $s_r : K^* \to Q$ such that*

$$\bigotimes_{u \in K^{\leq m-1}} wt(\overrightarrow{r(u)}) \leq_{S^Q} \bigotimes_{u \in K^*} wt(\overrightarrow{s_r(u)}).$$

Using the fact that $\otimes$ is idempotent, it is easy to see that every WLA is m-complete for any m greater than the number of states $|Q|$. The proof is basically the same as the one given in [3] for the fact that a looping tree automaton has a run iff it has a partial run of depth $> |Q|$.

Theorem 4. *If $\mathcal{A}$ is an m-complete WLA, then $\|\mathcal{A}\| = \bigoplus_{q \in Q} in(q) \otimes \mathcal{O}^m(\widetilde{\mathbf{1}})(q)$.*

Proof. Since Lemma 2 yields $\|\mathcal{A}\| = \bigoplus_{q \in Q} in(q) \otimes \sigma^{\|}(q)$, it suffices to prove that $\mathcal{O}^m(\widetilde{\mathbf{1}}) = \sigma^{\|}$. Furthermore, Theorem 3 implies that $\sigma^{\|}$ is the infimum of $\{\mathcal{O}^n(\widetilde{\mathbf{1}}) \mid n \geq 0\}$. Thus, it is enough to show that $\mathcal{O}^m(\widetilde{\mathbf{1}}) \leq_{S^Q} \sigma^{\|}$.

By Lemma 3, $\mathcal{O}^m(\widetilde{\mathbf{1}})(q) = \bigoplus_{r \in K_q^{\leq m}} \bigotimes_{u \in K^{\leq m-1}} wt(\overrightarrow{r(u)})$. Since $\mathcal{A}$ is m-complete, we can replace every partial run r appearing in the previous identity by the corresponding run s_r, obtaining a greater element in the semiring:

$$\begin{aligned} \mathcal{O}^m(\widetilde{\mathbf{1}})(q) &\leq_{S^Q} \bigoplus_{r \in K_q^{\leq m}} \bigotimes_{u \in K^*} wt(\overrightarrow{s_r(u)}) \\ &\leq_{S^Q} \bigoplus_{s \in K_q^*} \bigotimes_{u \in K^*} wt(\overrightarrow{s(u)}) = \sigma^{\|}(q). \end{aligned}$$

This completes the proof of the theorem. □

Let us apply this result to the pinpointing automaton for $\mathcal{SI}$ constructed in the previous section. 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 $\mathcal{O}$. It is also easy to see that the time required by each application of $\mathcal{O}$ is exponential in n.

Corollary 1. *Let C be an $\mathcal{ALC}$ concept description and $\mathcal{T}$ an $\mathcal{SI}$ TBox. The 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.

6 Conclusions

We have introduced a general framework for extending decision procedures based on the construction of looping tree automata to pinpointing algorithms. This framework can also 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 is based on the use of weighted automata working on infinite trees, which have until now not been studied. One of the main contributions of this paper is an approach for computing the behaviour of such automata. An interesting topic for future work is to check whether this 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 compute the behaviour of weighted automata working on infinite trees that use a non-trivial acceptance condition for runs, such as the Büchi condition. This would allow us to extend our framework such that it can deal with DLs that require such acceptance conditions, such as the DL $\mathcal{ALC}_{\mathsf{trans}}$, which allows for transitive closure of roles [1].

References

[1] Baader, F.: 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 1991) (1991)

[2] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, Cambridge (2003)

[3] Baader, F., Hladik, J., Peñaloza, R.: Automata can show PSPACE results for description logics. Information and Computation (to appear, 2008)

[4] Baader, F., Hollunder, B.: Embedding defaults into terminological knowledge representation formalisms. J. of Automated Reasoning 14, 149–180 (1995)

[5] Baader, F., Peñaloza, R.: Axiom pinpointing in general tableaux. In: Olivetti, N. (ed.) TABLEAUX 2007. LNCS (LNAI), vol. 4548, pp. 11–27. Springer, Heidelberg (2007)

[6] Baader, F., Peñaloza, R.: 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), `http://lat.inf.tu-dresden.de/research/reports.html`

[7] Baader, F., Peñaloza, R.: Pinpointing in terminating forest tableaux. LTCS-Report LTCS-08-03, Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, Germany (2008), `http://lat.inf.tu-dresden.de/research/reports.html`

[8] Baader, F., Sattler, U.: An overview of tableau algorithms for description logics. Studia Logica 69, 5–40 (2001)

[9] Baader, F., Suntisrivaraporn, B.: Debugging SNOMED CT using axiom pinpointing in the description logic $\mathcal{EL}^+$. In: Proceedings of the International Conference on Representing and Sharing Knowledge Using SNOMED (KR-MED 2008), Phoenix, Arizona (2008)

[10] Baader, F., Tobies, S.: The inverse method implements the automata approach for modal satisfiability. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS (LNAI), vol. 2083, pp. 92–106. Springer, Heidelberg (2001)
[11] Droste, M., Rahonis, G.: Weighted automata and weighted logics on infinite words. In: Ibarra, O.H., Dang, Z. (eds.) DLT 2006. LNCS, vol. 4036, pp. 49–58. Springer, Heidelberg (2006)
[12] Grätzer, G.: General Lattice Theory, 2nd edn. Birkhäuser, Basel (1998)
[13] Haarslev, V., Möller, R.: RACER system description. In: Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001) (2001)
[14] Horrocks, I.: Using an expressive description logic: FaCT or fiction. In: Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 1998), pp. 636–647 (1998)
[15] Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From SHIQ and RDF to OWL: The making of a web ontology language. Journal of Web Semantics 1(1), 7–26 (2003)
[16] Lee, K., Meyer, T., Pan, J.Z.: Computing maximally satisfiable terminologies for the description logic $\mathcal{ALC}$ with GCIs. In: Proc. of the 2006 Description Logic Workshop (DL 2006), CEUR Electronic Workshop Proceedings (2006)
[17] Parsia, B., Sirin, E., Kalyanpur, A.: Debugging OWL ontologies. In: Ellis, A., Hagino, T. (eds.) Proc. of the 14th International Conference on World Wide Web (WWW 2005), pp. 633–640. ACM Press, New York (2005)
[18] Schlobach, S.: Diagnosing terminologies. In: Veloso, M.M., Kambhampati, S. (eds.) Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005), pp. 670–675. AAAI Press/The MIT Press (2005)
[19] Schlobach, S., Cornet, R.: Non-standard reasoning services for the debugging of description logic terminologies. In: Gottlob, G., Walsh, T. (eds.) Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003), Acapulco, Mexico, pp. 355–362. Morgan Kaufmann, Los Altos (2003)
[20] Schmidt-Schauß, M., Smolka, G.: Attributive concept descriptions with complements. Artificial Intelligence 48(1), 1–26 (1991)
[21] Seidl, H.: Finite tree automata with cost functions. Theor. Comput. Sci. 126(1), 113–142 (1994)
[22] Sirin, E., Parsia, B.: Pellet: An OWL DL reasoner. In: Proc. of the 2004 Description Logic Workshop (DL 2004), pp. 212–213 (2004)
[23] Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. Pacific Journal of Mathematics 5, 285–309 (1955)
[24] Vardi, M.Y., Wolper, P.: 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 1984)