$\mathcal{SI}$! Automata Can Show PSpace Results for Description Logics

Franz Baader[1], Jan Hladik[1], and Rafael Peñaloza[2]*

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

Abstract. In Description Logics (DLs), both tableau-based and automata-based algorithms are frequently used to show decidability and complexity results for basic inference problems such as satisfiability of concepts. Whereas tableau-based algorithms usually yield worst-case optimal algorithms in the case of PSpace-complete logics, it is often very hard to design optimal tableau-based algorithms for ExpTime-complete DLs. In contrast, the automata-based approach is usually well-suited to prove ExpTime upper-bounds, but its direct application will usually also yield an ExpTime-algorithm for a PSpace-complete logic since the (tree) automaton constructed for a given concept is usually exponentially large. In the present paper, we formulate conditions under which an on-the-fly construction of such an exponentially large automaton can be used to obtain a PSpace-algorithm. We illustrate the usefulness of this approach by proving a new PSpace upper-bound for satisfiability of concepts w.r.t. acyclic terminologies in the DL $\mathcal{SI}$, which extends the basic DL $\mathcal{ALC}$ with transitive and inverse roles.

1 Introduction

Description Logics (DLs) [2] are a successful family of logic-based knowledge representation formalisms, which can be used to represent the conceptual knowledge of an application domain in a structured and formally well-understood way. DL systems provide their users with inference services that deduce implicit knowledge from the explicitly represented knowledge. For these inference services to be feasible, the underlying inference problems must at least be decidable, and preferably of low complexity. For this reason, investigating the computational complexity of reasoning in DLs of differing expressive power has been one of the most important research topics in the field for the last 20 years. Since Description Logics are closely related to Modal Logics (MLs) [17], results and techniques can be transferred between the two areas.

Two of the most prominent methods for showing decidability and complexity results for DLs and MLs are the tableau-based [9, 5] and the automata-based

* Funded by the DFG, Graduiertenkolleg Wissensrepräsentation, Uni Leipzig.

[19, 8] approach. Both approaches basically depend on the tree-model property of the DL/ML under consideration: if a concept/formula is satisfiable, then it is also satisfiable in a tree-shaped model. They differ in how they test for the existence of a tree-shaped model. Tableau-based algorithms try to generate such a model in a top-down non-deterministic manner, starting with the root of the tree. Automata-based algorithms construct a tree automaton that accepts exactly the tree-shaped models of the concept/formula, and then test the language accepted by this automaton for emptiness. The usual emptiness test for tree automata is deterministic and works in a bottom-up manner. This difference between the approaches also leads to different behaviour regarding elegance, complexity, and practicability.

If the logic has the *finite* tree model property, then termination of tableau-based algorithms is usually easy to achieve. If, in addition, the tree models these algorithms are trying to construct are of polynomial depth (as is the case for the PSpace-complete problem of satisfiability in the basic DL $\mathcal{ALC}$, which corresponds to the multi-modal variant of the ML K), then one can usually modify tableau-based algorithms such that they need only polynomial space: basically, they must only keep one path of the tree in memory [18]. However, the automaton constructed in the automata-based approach is usually exponential, and thus constructing it explicitly before applying the emptiness test requires exponential time and space. In [10], we formulate conditions on the constructed automaton that ensure—in the case of finite tree models of polynomially bounded depth—that an on-the-fly construction of the automaton during a non-deterministic top-down emptiness test yields a PSpace algorithm.

If the logic does *not* have the *finite* tree model property, then applying the tableau-based approach in a straightforward manner leads to a non-terminating procedure. To ensure termination of tableau-based algorithms in this case, one must apply an appropriate cycle-checking technique, called "blocking" in the DL literature [5]. This is, for example, the case for satisfiability in $\mathcal{ALC}$ w.r.t. so-called general concept inclusions (GCIs) [1]. Since blocking usually occurs only after an exponential number of steps and since tableau-based algorithms are non-deterministic, the best complexity upper-bound that can be obtained this way is NExpTime. This is not optimal since satisfiability in $\mathcal{ALC}$ w.r.t. GCIs is "only" ExpTime-complete. The ExpTime upper-bound can easily be shown with the automata-based approach: the constructed automaton is of exponential size, and the (bottom-up) emptiness test for tree automata runs in time polynomial in the size of the automaton. Although the automata-based approach yields a worst-case optimal algorithm in this case, the obtained algorithm is not practical since it is also exponential in the best case: before applying the emptiness test, the exponentially large automaton must be constructed. In contrast, optimised implementations of tableau-based algorithms usually behave quite well in practice [11], in spite of the fact that they are not worst-case optimal. There have been some attempts to overcome this mismatch between practical and worst-case optimal algorithms for ExpTime-complete DLs. In [6] we show that the so-called inverse tableau method [20] can be seen as an on-the-fly implementation of the emptiness test in the automata-based approach, which avoids the a priori construction of the exponentially large automaton. Conversely, we show in [3] that the existence of a sound and complete so-called ExpTime-admissible tableau-based algorithm for a logic always implies the existence of an ExpTime automata-based algorithm. This allows us to construct only the (practical, but not worst-case optimal) tableau-based algorithm, and get the optimal ExpTime upper-bound for free.

In the present paper, we extend the approach from [10] mentioned above such that it can also deal with PSpace-complete logics that do not have the finite tree model property. A well-known example of such a logic is $\mathcal{ALC}$ extended with transitive roles [15]. To illustrate the power of our approach, we use the more expressive DL $\mathcal{SI}$ as an example, which extends $\mathcal{ALC}$ with transitive and inverse roles. In addition, we also allow for acyclic concept definitions. To the best of our knowledge, the result that satisfiability in $\mathcal{SI}$ w.r.t. acyclic concept definitions is in PSpace is new.

For lack of space we must omit most of the proofs of our results. Detailed proofs can be found in [4].

2 The description logic $\mathcal{SI}$

In Description Logics, concepts are built from concept names (unary predicates) and role names (binary predicates) with the help of concept constructors. In addition, one sometimes has additional restrictions on the interpretation of role names. A particular DL is determined by the available constructors and restrictions. The DL $\mathcal{SI}$ has the same concept constructors as the basic DL $\mathcal{ALC}$ [18], but it additionally allows to restrict roles to being transitive and to being inverses of each other.[3] A typical example of a role that should be interpreted as transitive is has-offspring. In addition, has-ancestors should be interpreted as the inverse of has-offspring.

Definition 1 (Syntax and semantics of $\mathcal{SI}$). Let N_C be a set of *concept names* and N_R be a set of *role names*, where $N_T \subseteq N_R$ is the set of *transitive role names*. Then the set of $\mathcal{SI}$ *roles* is defined as $N_R \cup \{r^- \mid r \in N_R\}$, and the set of $\mathcal{SI}$ *concepts* is the smallest set that satisfies the following conditions:

- all concept names are $\mathcal{SI}$ concepts;
- if C and D are $\mathcal{SI}$ concepts, then $\neg C$, $C \sqcup D$ and $C \sqcap D$ are $\mathcal{SI}$ concepts;
- if C is an $\mathcal{SI}$ concept and r an $\mathcal{SI}$ role, then $\exists r.C$ and $\forall r.C$ are $\mathcal{SI}$ concepts.

An *interpretation* $\mathcal{I}$ is a pair $(\Delta^\mathcal{I}, \cdot^\mathcal{I})$, where $\Delta^\mathcal{I}$ is a non-empty set (the *domain* of $\mathcal{I}$) 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 $r^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$ such that $r^\mathcal{I}$ is transitive for all $r \in N_T$. This function is extended to $\mathcal{SI}$ roles and concepts by defining

- $(r^-)^\mathcal{I} := \{(y, x) \mid (x, y) \in r^\mathcal{I}\}$;

[3] $\mathcal{SI}$ thus corresponds to the multi-modal logic $\mathsf{S4_m}$ with converse modalities.

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

The following notation will turn out to be useful later on: for an $\mathcal{SI}$ role s, the *inverse of s (denoted by $\overline{s}$)* is s^- if s is a role name, and r if $s = r^-$. Since a role is interpreted as transitive iff its inverse is interpreted as transitive, we will use the predicate $\mathsf{trans}(r)$ on $\mathcal{SI}$ roles to express that r or $\overline{r}$ belongs to N_T.

Knowledge about the domain of interest is stored in *TBoxes*. TBoxes can contain *concept definitions*, which introduce abbreviations for complex concepts, and *general concept inclusions*, which restrict the possible interpretations.

Definition 2 (Syntax and semantics of TBoxes). A *general concept inclusion (GCI)* has the form $C \sqsubseteq D$, where C and D are $\mathcal{SI}$ concepts, and a *concept definition* has the form $A \doteq C$, where A is a concept name and C is an $\mathcal{SI}$ concept.

An *acyclic TBox* is a finite set of concept definitions such that every concept name occurs at most once as a left-hand side, and there is no cyclic dependency between the definitions, i.e. there is no sequence of concept definitions $A_1 \doteq C_1, \ldots, A_n \doteq C_n$ such that C_i contains A_{i+1} for $1 \leq i < n$ and C_n contains A_1. A *general TBox* is an acyclic TBox extended with a finite set of GCIs.

An interpretation $\mathcal{I}$ is called a *model* of the (general or acyclic) TBox $\mathcal{T}$ if $A^\mathcal{I} = C^\mathcal{I}$ ($C^\mathcal{I} \subseteq D^\mathcal{I}$) holds for for every concept definition $A \doteq C \in \mathcal{T}$ (GCI $C \sqsubseteq D \in \mathcal{T}$).

A concept name is called *defined* if it occurs on the left-hand side of a concept definition, and *primitive* otherwise. The definition of acyclic TBoxes ensures that the concept definitions simply introduce abbreviations (macro definitions), which could in principle be completely expanded by repeatedly replacing defined names by their definitions. Thus, acyclic TBoxes do not increase the expressive power, but they increase succinctness: it is well-known that expansion can lead to an exponential blow-up [14]. Obviously, the concept definition $A \doteq C$ can be expressed by the two GCIs $A \sqsubseteq C$ and $C \sqsubseteq A$. Nevertheless, it makes sense to distinguish between an acyclic set of concept definitions and GCIs within general TBoxes since acyclic concept definitions can be treated in a more efficient way when deciding the satisfiability problem.

Definition 3 (The satisfiability problem). The $\mathcal{SI}$ concept C is *satisfiable* w.r.t. the (general or acyclic) TBox $\mathcal{T}$ if there is a model $\mathcal{I}$ of $\mathcal{T}$ with $C^\mathcal{I} \neq \emptyset$. In this case, we call $\mathcal{I}$ also a *model* of C w.r.t. $\mathcal{T}$.

For the DL $\mathcal{ALC}$ (i.e. $\mathcal{SI}$ without transitive and inverse roles), it is known that the satisfiability problem is PSpace-complete w.r.t. acyclic TBoxes [13] and ExpTime-complete w.r.t. general TBoxes [17]. We will show in this paper that the same is true for $\mathcal{SI}$.

Tree models of satisfiable $\mathcal{SI}$ concepts can be obtained by applying the well-known technique of unravelling [7]. For example, the $\mathcal{SI}$ concept A is satisfiable w.r.t. the general TBox $\{A \sqsubseteq \exists r.A\}$ in a one-element model whose single element belongs to A and is related to itself via r. The corresponding unravelled model consists of a sequence $d_0, d_1, d_2, \ldots$ of elements, all belonging to A, where d_i is related to d_{i+1} via r. Intuitively, Hintikka trees are tree models where every node is labelled with the concepts to which the element represented by the node belongs. These concepts are taken from the set of subconcepts of the concept to be tested for satisfiability and of the concepts occurring in the TBox. In our example, the nodes d_i would be labelled by A and $\exists r.A$ since each d_i belongs to these concepts.

To simplify the formal definitions, we assume in the following that *all* concepts are in *negation normal form (NNF)*, i.e. negation appears only directly in front of concept names. Any $\mathcal{SI}$ concept can be transformed into NNF in linear time using de Morgan's laws, duality of quantifiers, and elimination of double negation. We denote the NNF of a concept C by $\mathsf{nnf}(C)$ and $\mathsf{nnf}(\neg C)$ by $\backsim C$.

Definition 4 (Subconcepts, Hintikka sets). The set of *subconcepts* of an $\mathcal{SI}$ concept C ($\mathsf{sub}(C)$) is the least set S that contains C and has the following properties: if S contains $\neg A$ for a concept name A, then $A \in S$; if S contains $D \sqcup E$ or $D \sqcap E$, then $\{D, E\} \subseteq S$; if S contains $\exists r.D$ or $\forall r.D$, then $D \in S$. For a TBox $\mathcal{T}$, $\mathsf{sub}(C, \mathcal{T})$ is defined as follows:

$$\mathsf{sub}(C) \quad \cup \bigcup_{A \doteq D \in \mathcal{T}} (\{A, \neg A\} \cup \mathsf{sub}(D) \cup \mathsf{sub}(\backsim D)) \quad \cup \bigcup_{D \sqsubseteq E \in \mathcal{T}} \mathsf{sub}(\backsim D \sqcup E)$$

A set $H \subseteq \mathsf{sub}(C, \mathcal{T})$ is called a *Hintikka set for C* if the following three conditions are satisfied: if $D \sqcap E \in H$, then $\{D, E\} \subseteq H$; if $D \sqcup E \in H$, then $\{D, E\} \cap H \neq \emptyset$; and there is no concept name A with $\{A, \neg A\} \subseteq H$. For a TBox $\mathcal{T}$, a Hintikka set H is called *$\mathcal{T}$-expanded* if for every GCI $D \sqsubseteq E \in \mathcal{T}$, it holds that $\backsim D \sqcup E \in H$, and for every concept definition $A \doteq D \in \mathcal{T}$, it holds that, if $A \in H$ then $D \in H$, and if $\neg A \in H$ then $\backsim D \in H$.[4]

Hintikka trees for C and $\mathcal{T}$ are infinite trees of a fixed arity k, which is determined by the number of existential restrictions, i.e. concepts of the form $\exists r.D$, in $\mathsf{sub}(C, \mathcal{T})$. For a positive integer k, we denote the set $\{1, \ldots, k\}$ by K. The nodes of a k-ary tree can be denoted by the elements of K^*, with the empty word ε denoting the root, and ui the ith successor of u. In the case of labelled trees, we will refer to the label of the node u in the tree t by $t(u)$. In the definition of Hintikka trees, we need to know which successor in the tree corresponds to which existential restriction. For this purpose, we fix a linear order on the existential restrictions in $\mathsf{sub}(C, \mathcal{T})$. Let $\varphi : \{\exists r.D \in \mathsf{sub}(C, \mathcal{T})\} \to K$ be the corresponding ordering function, i.e. $\varphi(\exists r.D)$ determines the successor node corresponding to $\exists r.D$. In general, such a successor node need not exist in a tree model. To obtain a full k-ary tree, Hintikka trees contain appropriate dummy

[4] This technique of handling concept definitions is called *lazy unfolding*. Note that, in contrast to GCIs, concept definitions are only applied if A or $\neg A$ is explicitly present in H.

nodes. For technical reasons, which will become clear later on, the nodes of the Hintikka trees defined below are not simply labelled by Hintikka sets, but by quadruples $(\Gamma, \Pi, \Omega, \varrho)$, where ϱ is the role which connects the node with the father node, Ω is the complete Hintikka set for the node, $\Gamma \subseteq \Omega$ consists of the unique concept D contained in Ω because of an existential restriction $\exists\varrho.D$ in the father node, and Π contains only those concepts that are contained in Ω because of universal restrictions $\forall\varrho.E$ in the father node. We will use a special new role name λ for nodes that are not connected to the father node by a role, i.e. the root node and those (dummy) nodes which are labelled with an empty set of concepts.

Definition 5 (Hintikka trees). The tuple $((\Gamma_0, \Pi_0, \Omega_0, \varrho_0), (\Gamma_1, \Pi_1, \Omega_1, \varrho_1), \ldots, (\Gamma_k, \Pi_k, \Omega_k, \varrho_k))$ is called $C, \mathcal{T}$*-compatible* if, for all $i, 0 \leq i \leq k$, $\Gamma_i \cup \Pi_i \subseteq \Omega_i$, Ω_i is a $\mathcal{T}$-expanded Hintikka set, and the following holds for every existential concept $\exists r.D \in \mathsf{sub}(C, \mathcal{T})$:

- if $\exists r.D \in \Omega_0$, then
  1. $\Gamma_{\varphi(\exists r.D)}$ consists of D;
  2. $\Pi_{\varphi(\exists r.D)}$ consists of all concepts E for which there is a universal restriction $\forall r.E \in \Omega_0$, and it additionally contains $\forall r.E$ if $\mathsf{trans}(r)$;
  3. for every concept $\forall \overline{r}.F \in \Omega_{\varphi(\exists r.D)}$, Ω_0 contains F, and additionally $\forall \overline{r}.F$ if $\mathsf{trans}(r)$;
  4. $\varrho_{\varphi(\exists r.D)} = r$;
- if $\exists r.D \notin \Omega_0$, then $\Gamma_{\varphi(\exists r.D)} = \Pi_{\varphi(\exists r.D)} = \Omega_{\varphi(\exists r.D)} = \emptyset$ and $\varrho_{\varphi(\exists r.D)} = \lambda$.

A k-ary tree t is called a *Hintikka tree for* C *and* $\mathcal{T}$ if, for every node $v \in K^*$, the tuple $(t(v), t(v1), \ldots, t(vk))$ is $C, \mathcal{T}$-compatible, and $t(\varepsilon)$ has empty Γ- and Π-components, an Ω-component containing C, and λ as its ϱ-component.

Our definition of a Hintikka tree ensures that the existence of such a tree characterises satisfiability of $\mathcal{SI}$ concepts. It basically combines the technique for handling transitive and inverse roles introduced in [12][5] with the technique for dealing with acyclic TBoxes employed in [10]. A full proof of the next theorem can be found in [4].

Theorem 6. The $\mathcal{SI}$ concept C is satisfiable w.r.t. the general TBox $\mathcal{T}$ iff there exists a Hintikka tree for C and $\mathcal{T}$.

3 Tree automata

The existence of a Hintikka tree can be decided with the help of so-called looping automata, i.e. automata on infinite trees without a special acceptance condition. After introducing these automata, we will first show how they can be used to decide satisfiability in $\mathcal{SI}$ w.r.t. general TBoxes in exponential time. Then we will introduce a restricted class of looping automata and use it to show that satisfiability in $\mathcal{SI}$ w.r.t. acyclic TBoxes can be decided in polynomial space.

[5] there used in the context of tableau-based algorithms.

3.1 Looping automata

The following definition of looping tree automata does not include an alphabet for labelling the nodes of the trees. In fact, when deciding the emptiness problem for such automata, only the *existence* of a tree accepted by the automaton is relevant, and not the labels of its nodes. For our reduction this implies that the automaton we construct for a given input $C, \mathcal{T}$ has as its *successful runs* all Hintikka trees for $C, \mathcal{T}$ rather than actually accepting all Hintikka trees for $C, \mathcal{T}$.

Definition 7 (Automaton, run). A *looping tree automaton* over k-ary trees 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 (unique) unlabelled k-ary tree t is a labelled k-ary tree $r : K^* \to Q$ such that $(r(v), r(v1), \ldots, r(vk)) \in \Delta$ for all $v \in K^*$. The run is *successful* if $r(\varepsilon) \in I$. The *emptiness problem for looping tree automata* is the problem of deciding whether a given looping tree automaton has a successful run or not.

In order to *decide the emptiness problem* in time polynomial in the size of the automaton, one computes the set of all bad states, i.e. states that do not occur in any run, in a *bottom-up* manner [19, 6]: states that do not occur as first component in a transition are bad, and if all transitions that have the state q as first component contain a state already known to be bad, then q is also bad. The automaton has a successful run iff there is an initial state that is not bad.

For an $\mathcal{SI}$ concept C and a general TBox $\mathcal{T}$, we can construct a looping tree automaton whose successful runs are exactly the Hintikka trees for C and $\mathcal{T}$.

Definition 8 (Automaton $\mathcal{A}_{C,\mathcal{T}}$). For an $\mathcal{SI}$ concept C and a TBox $\mathcal{T}$, let k be the number of existential restrictions in $\mathsf{sub}(C, \mathcal{T})$. Then the looping automaton $\mathcal{A}_{C,\mathcal{T}} = (Q, \Delta, I)$ is defined as follows:

- Q consists of all 4-tuples $(\Gamma, \Pi, \Omega, \varrho)$ such that $\Gamma \cup \Pi \subseteq \Omega \subseteq \mathsf{sub}(C, \mathcal{T})$, Γ is a singleton set, Ω is a $\mathcal{T}$-expanded Hintikka set for C, and ϱ is a role that occurs in C or $\mathcal{T}$ or is equal to λ;
- Δ consists of all $C, \mathcal{T}$-compatible tuples $((\Gamma_0, \Pi_0, \Omega_0, \varrho_0), (\Gamma_1, \Pi_1, \Omega_1, \varrho_1), \ldots, (\Gamma_k, \Pi_k, \Omega_k, \varrho_k))$;
- $I := \{(\emptyset, \emptyset, \Omega, \lambda) \in Q \mid C \in \Omega\}$.

Lemma 9. $\mathcal{A}_{C,\mathcal{T}}$ has a successful run iff C is satisfiable w.r.t. $\mathcal{T}$.

Since the cardinality of $\mathsf{sub}(C, \mathcal{T})$ and the size of each of its elements is linear in the size of $C, \mathcal{T}$, the size of the automaton $\mathcal{A}_{C,\mathcal{T}}$ is exponential in the size of $C, \mathcal{T}$. Together with the fact that the emptiness problem for looping tree automata can be decided in polynomial time, this yields:

Theorem 10. Satisfiability in $\mathcal{SI}$ w.r.t. general TBoxes is in ExpTime.

This complexity upper-bound is optimal since ExpTime-hardness follows from the known hardness result for $\mathcal{ALC}$ with general TBoxes [17].

One could also try to solve the emptiness problem by constructing a successful run in a *top-down manner*: label the root with an element q_0 of I, then apply a transition with first component q_0 to label the successor nodes, etc. There are, however, two problems with this approach. Firstly, it yields a *non-deterministic* algorithm since I may contain more than one element, and in each step more than one transition may be applicable. Secondly, one must employ an appropriate cycle-checking technique (similar to blocking in tableau-based algorithms) to obtain a terminating algorithm. Applied to the automaton $\mathcal{A}_{C,\mathcal{T}}$, this approach would at best yield a (non-optimal) NExpTime satisfiability test.

3.2 Blocking-invariant automata

In order to obtain a PSpace result for satisfiability w.r.t. *acyclic* TBoxes, we use the top-down emptiness test sketched above. In fact, in this case non-determinism is unproblematic since NPSpace is equal to PSpace by Savitch's theorem [16]. The advantage of the top-down over the bottom-up emptiness test is that it is not necessary to construct the whole automaton before applying the emptiness test. Instead, the automaton can be constructed on-the-fly. However, we still need to deal with the termination problem. For this purpose, we adapt the blocking technique known from the tableau-based approach. In the following, when we speak about a *path* in a k-ary tree, we mean a sequence of nodes $v_1, \ldots, v_m$ such that v_1 is the root ε and v_{i+1} is a direct successor of v_i.

Definition 11 ($\longleftarrow\!\!\prec$-invariant, m-blocking). Let $\mathcal{A} = (Q, \Delta, I)$ be a looping tree automaton and $\longleftarrow\!\!\prec$ be a binary relation over Q, called the *blocking relation*. If $q \longleftarrow\!\!\prec p$, then we say that q is *blocked* by p. The automaton $\mathcal{A}$ is called $\longleftarrow\!\!\prec$-*invariant* if, for every $q \longleftarrow\!\!\prec p$, and $(q_0, q_1, \ldots, q_{i-1}, q, q_{i+1}, \ldots, q_k) \in \Delta$, it holds that $(q_0, q_1, \ldots, q_{i-1}, p, q_{i+1}, \ldots, q_k) \in \Delta$. A $\longleftarrow\!\!\prec$-invariant automaton $\mathcal{A}$ is called m-*blocking* if, for every successful run r of $\mathcal{A}$ and every path $v_1, \ldots, v_m$ of length m in r, there are $1 \leq i < j \leq m$ such that $r(v_j) \longleftarrow\!\!\prec r(v_i)$.

Obviously, any looping automaton $\mathcal{A} = (Q, \Delta, I)$ is $=$-invariant (i.e. the blocking relation is equality) and m-blocking for every $m > \#Q$ (where "$\#Q$" denotes the cardinality of Q). However, we are interested in automata and blocking relations where blocking occurs earlier than after a linear number of transitions.

To test an m-blocking automaton for emptiness, it is sufficient to construct partial runs of depth m. More formally, we define $K^{\leq n} := \bigcup_{i=0}^{n} K^i$. A *partial run of depth* m is a mapping $r : K^{\leq m-1} \rightarrow Q$ such that $(r(v), r(v1), \ldots, r(vk)) \in \Delta$ for all $v \in K^{\leq m-2}$. It is *successful* if $r(\varepsilon) \in I$.

Lemma 12. An m-blocking automaton $\mathcal{A} = (Q, \Delta, I)$ has a successful run iff it has a successful partial run of depth m.

For $k > 1$, the size of a successful partial run of depth m is still exponential in m. However, when checking for the existence of such a run, one can perform a depth-first traversal of the run while constructing it. To do this, it is basically

```
if I ≠ ∅ then
    guess an initial state q ∈ I
else
    return "empty"
if there is a transition from q then
    guess such a transition (q, q_1, ..., q_k) ∈ Δ
    push(SQ, (q_1, ..., q_k)), push(SN, 0)
else
    return "empty"
while SN is not empty do
    (q_1, ..., q_k) := pop(SQ), n := pop(SN) + 1
    if n ≤ k then
        push(SQ, (q_1, ..., q_k)), push(SN, n)
        if length(SN) < m − 1 then
            if there is a transition from q_n then
                guess a transition (q_n, q'_1, ..., q'_k) ∈ Δ
                push(SQ, (q'_1, ..., q'_k)), push(SN, 0)
            else
                return "empty"
return "not empty"
```

Fig. 1. The non-deterministic top-down emptiness test for m-blocking automata.

enough to have at most one path of length up to m in memory.[6] The algorithm that realizes this idea is shown in Figure 1. It uses two stacks: the stack SQ stores, for every node on the current path, the right-hand side of the transition which led to this node, and the stack SN stores, for every node on the current path, on which component of this right-hand side we are currently working. The entries of SQ and SN are elements of Q^k and $K \cup \{0\}$, respectively, and the number of entries is bounded by m for each stack.

Note that the algorithm does not require the automaton $\mathcal{A}$ to be explicitly given. It can be constructed on-the-fly during the run of the algorithm.

Definition 13. Assume that we have a set of inputs $\mathfrak{I}$ and a construction that yields, for every $\mathfrak{i} \in \mathfrak{I}$, an $m_\mathfrak{i}$-blocking automaton $\mathcal{A}_\mathfrak{i} = (Q_\mathfrak{i}, \Delta_\mathfrak{i}, I_\mathfrak{i})$ working on $k_\mathfrak{i}$-ary trees. We say that this construction is a PSpace *on-the-fly construction* if there is a polynomial P such that, for every input $\mathfrak{i}$ of size n we have

- $m_\mathfrak{i} \leq P(n)$ and $k_\mathfrak{i} \leq P(n)$;
- every element of $Q_\mathfrak{i}$ is of a size bounded by $P(n)$;
- one can non-deterministically guess in time bounded by $P(n)$ an element of $I_\mathfrak{i}$ and, for a state $q \in Q_\mathfrak{i}$, a transition from $\Delta_\mathfrak{i}$ with first component q.

The algorithms guessing an initial state (a transition starting with q) are assumed to yield the answer "no" if there is no initial state (no such transition).

The following theorem is an easy consequence of the correctness of the top-down emptiness test described in Figure 1 and Savitch's theorem [16].

[6] This is similar to the so-called trace technique for tableau-based algorithms [18].

Theorem 14. If the automata $\mathcal{A}_\mathfrak{i}$ are obtained from the inputs $\mathfrak{i} \in \mathfrak{I}$ by a PSPACE on-the-fly construction, then the emptiness problem for $\mathcal{A}_\mathfrak{i}$ can be decided by a deterministic algorithm in space polynomial in the size of $\mathfrak{i}$.

3.3 Satisfiability in $\mathcal{SI}$ w.r.t. acyclic TBoxes

It is easy to see that the construction of the automaton $\mathcal{A}_{C,\mathcal{T}}$ from a given $\mathcal{SI}$ concept C and a general TBox $\mathcal{T}$ satisfies all but one of the conditions of a PSPACE on-the-fly construction. The condition that is violated is the one requiring that blocking must occur after a polynomial number of steps. In the case of general TBoxes, this is not surprising since we know that the satisfiability problem is EXPTIME-hard. Unfortunately, this condition is also violated if $\mathcal{T}$ is an acyclic TBox. The reason is that successor states may contain new concepts that are not really required by the definition of $C,\mathcal{T}$-compatible tuples, but are also not prevented by this definition. In the case of acyclic TBoxes, we can construct a subautomaton that avoids such unnecessary concepts. It has fewer runs than $\mathcal{A}_{C,\mathcal{T}}$, but it does have a successful run whenever $\mathcal{A}_{C,\mathcal{T}}$ has one. The construction of this subautomaton follows the following general pattern.

Definition 15 (Faithful). Let $\mathcal{A} = (Q, \Delta, I)$ be a looping tree automaton on k-ary trees. The family of functions $f_q : Q \to Q^{\mathrm{S}}$ for $q \in Q^{\mathrm{S}}$ is *faithful* w.r.t. $\mathcal{A}$ if $I \subseteq Q^{\mathrm{S}} \subseteq Q$, and the following two conditions are satisfied for every $q \in Q^{\mathrm{S}}$:

1. if $(q, q_1, \ldots, q_k) \in \Delta$, then $(q, f_q(q_1), \ldots, f_q(q_k)) \in \Delta$;
2. if $(q_0, q_1, \ldots, q_k) \in \Delta$, then $(f_q(q_0), f_q(q_1), \ldots, f_q(q_k)) \in \Delta$.[7]

The *subautomaton* $\mathcal{A}^S = (Q^S, \Delta^S, I)$ *of* $\mathcal{A}$ *induced by this family* has the transition relation $\Delta^{\mathrm{S}} := \{(q, f_q(q_1), \ldots, f_q(q_k)) \mid (q, q_1, \ldots, q_k) \in \Delta \text{ and } q \in Q^{\mathrm{S}}\}$.

Lemma 16. Let $\mathcal{A}$ be a looping tree automaton and $\mathcal{A}^{\mathrm{S}}$ its subautomaton induced by the faithful family of functions $f_q : Q \to Q^{\mathrm{S}}$ for $q \in Q^{\mathrm{S}}$. Then $\mathcal{A}$ has a successful run iff $\mathcal{A}^{\mathrm{S}}$ has a successful run.

Intuitively, the range of f_q contains the states that are allowed after state q has been reached. Before we can define an appropriate family of functions for $\mathcal{A}_{C,\mathcal{T}}$, we must introduce some notation. For an $\mathcal{SI}$ concept C and an acyclic TBox $\mathcal{T}$, the *role depth* $\mathsf{rd}_\mathcal{T}(C)$ of C w.r.t. $\mathcal{T}$ is the maximal nesting of (universal and existential) role restrictions in the concept obtained by expanding C w.r.t. $\mathcal{T}$. Obviously, $\mathsf{rd}_\mathcal{T}(C)$ is polynomially bounded by the size of $C, \mathcal{T}$. For a set of $\mathcal{SI}$ concepts S, its role depth $\mathsf{rd}_\mathcal{T}(S)$ w.r.t. $\mathcal{T}$ is the maximal role depth w.r.t. $\mathcal{T}$ of the elements of S. We define $\mathsf{sub}_{\leqslant n}(C, \mathcal{T}) := \{D \mid D \in \mathsf{sub}(C, \mathcal{T}) \text{ and } \mathsf{rd}_\mathcal{T}(D) \leq n\}$, and $S/r := \{D \in S \mid \text{there is an } E \text{ such that } D = \forall r.E\}$.

The main idea underlying the next definition is the following. If $\mathcal{T}$ is acyclic then, since we use lazy unfolding of concept definitions, the definition of $C,\mathcal{T}$-compatibility requires, for a transition $(q, q_1, \ldots, q_k)$ of $\mathcal{A}_{C,\mathcal{T}}$, only the existence

[7] Note that this condition does neither imply nor follow from condition 1, since q_0 need not be equal to q, and it is not required that $f_q(q)$ equals q.

of concepts in $q_i = (\Gamma_i, \Pi_i, \Omega_i, \varrho_i)$ that are of a smaller depth than the maximal depth n of concepts in q if ϱ_i is not transitive. If ϱ_i is transitive, then Π_i may also contain universal restrictions of depth n. We can therefore remove from the states q_i all concepts with a higher depth and still maintain $C, \mathcal{T}$-compatibility.

Definition 17 (Functions f_q). For two states $q = (\Gamma, \Pi, \Omega, \varrho)$ and $q' = (\Gamma', \Pi', \Omega', \varrho')$ of $\mathcal{A}_{C,\mathcal{T}}$ with $\mathsf{rd}_{\mathcal{T}}(\Omega) = n$, we define the function $f_q(q')$ as follows:

- if $\mathsf{rd}_{\mathcal{T}}(\Gamma') \geq \mathsf{rd}_{\mathcal{T}}(\Omega)$, then $f_q(q') := (\emptyset, \emptyset, \emptyset, \lambda)$;
- otherwise, $f_q(q') := (\Gamma', \Pi'', \Omega'', \varrho')$, where
  - $P = \mathsf{sub}_{\leqslant n}(C, \mathcal{T})/\varrho'$, if $\mathsf{trans}(\varrho')$; otherwise $P = \emptyset$;
  - $\Pi'' = \Pi' \cap (\mathsf{sub}_{\leqslant n-1}(C, \mathcal{T}) \cup P)$;
  - $\Omega'' = \Omega' \cap (\mathsf{sub}_{\leqslant n-1}(C, \mathcal{T}) \cup \Pi'')$.

If $\mathcal{T}$ is acyclic, then the set Ω'' defined above is still a $\mathcal{T}$-expanded Hintikka set.

Lemma 18. The family of mappings f_q (for states q of $\mathcal{A}_{C,\mathcal{T}}$) introduced in Definition 17 is faithful w.r.t. $\mathcal{A}_{C,\mathcal{T}}$.

Consequently, $\mathcal{A}_{C,\mathcal{T}}$ has a successful run iff the induced subautomaton $\mathcal{A}^{\mathrm{S}}_{C,\mathcal{T}}$ has a successful run.

Lemma 19. The construction of $\mathcal{A}^{\mathrm{S}}_{C,\mathcal{T}}$ from an input consisting of an $\mathcal{SI}$ concept C and an acyclic TBox $\mathcal{T}$ is a PSpace on-the-fly construction.

The main thing to show in the proof is that blocking always occurs after a polynomial number of steps. To show this, we use the following blocking relation: $(\Gamma_1, \Pi_1, \Omega_1, \varrho_1) \hookleftarrow_{\mathcal{SI}} (\Gamma_2, \Pi_2, \Omega_2, \varrho_2)$ if $\Gamma_1 = \Gamma_2$, $\Pi_1 = \Pi_2$, $\Omega_1/\overline{\varrho}_1 = \Omega_2/\overline{\varrho}_2$, and $\varrho_1 = \varrho_2$. If $m := \#\mathsf{sub}(C, \mathcal{T})$, then $\mathcal{A}^{\mathrm{S}}_{C,\mathcal{T}}$ is m^4-blocking w.r.t. $\hookleftarrow_{\mathcal{SI}}$. The main reasons for this to hold are the following: (i) if a successor node is reached w.r.t. a non-transitive role, then the role depth of the Ω-component decreases, and the same is true if within two steps two different transitive roles are used; (ii) if a successor node is reached w.r.t. a transitive role, then there is an inclusion relationship between the Π-components of the successor node and its father; the same is true (though in the other direction) for the $\Omega/\overline{\varrho}$-components.

Since we know that C is satisfiable w.r.t. $\mathcal{T}$ iff $\mathcal{A}_{C,\mathcal{T}}$ has a successful run iff $\mathcal{A}^{\mathrm{S}}_{C,\mathcal{T}}$ has a successful run, Theorem 14 yields the desired PSpace upper-bound. PSpace-hardness for this problem follows directly from the known PSpace-hardness of satisfiability w.r.t. the empty TBox in $\mathcal{ALC}$ [18].

Theorem 20. Satisfiability in $\mathcal{SI}$ w.r.t. acyclic TBoxes is PSpace-complete.

4 Conclusion

We have developed a framework for automata that adapts the notion of *blocking* from tableau algorithms and makes it possible to show tight complexity bounds for PSpace logics using the automata approach. In order to achieve this result, we replace the deterministic bottom-up emptiness test with a nondeterministic top-down test that can be interleaved with the construction of the automaton and aborted after a "blocked" state is reached. If the number of transitions before this happens is polynomial in the size of the input, emptiness of the automaton can be tested using space polynomial in the size of the input rather than time exponential in the size of the input. This illustrates the close relationship between tableau and automata algorithms.

As an application of this method, we have shown how blocking automata can be used to decide satisfiability of $\mathcal{SI}$ concepts w.r.t. acyclic TBoxes in PSPACE.

References

[1] F. Baader, H.-J. Bürckert, B. Hollunder, W. Nutt, and J.H. Siekmann. Concept logics. In *Computational Logics, Symposium Proceedings*, Springer-Verlag, 1990.

[2] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P.F. Patel-Schneider (eds). *The DL Handbook: Theory, Implementation, and Applications*. CUP, 2003.

[3] F. Baader, J. Hladik, C. Lutz, and F. Wolter. From tableaux to automata for description logics. *Fundamenta Informaticae*, 57(2–4):247–279, 2003.

[4] F. Baader, J. Hladik, and R. Peñaloza. PSPACE Automata with Blocking for Description Logics. LTCS-Report 06-04, Chair for Automata Theory, TU Dresden, 2006. Available at `http://lat.inf.tu-dresden.de/research/reports.html`.

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

[6] F. Baader and S. Tobies. The inverse method implements the automata approach for modal satisfiability. In *Proc. IJCAR 2001*, Sprincer LNCS 2083, 2001.

[7] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. CUP, 2001.

[8] D. Calvanese, G. De Giacomo, and M. Lenzerini. Reasoning in expressive DLs with fixpoints based on automata on infinite trees. In *Proc. IJCAI'99*, 1999.

[9] Melvin Fitting. *Proof Methods for Modal and Intuitionistic Logics*. Reidel, 1983.

[10] J. Hladik and R. Peñaloza. PSPACE automata for DLs. In *Proc. of DL'06*, 2006.

[11] I. Horrocks and P.F. Patel-Schneider. Optimizing description logic subsumption. *J. of Logic and Computation*, 9(3):267–293, 1999.

[12] I. Horrocks, U. Sattler, and S. Tobies. A PSpace-algorithm for deciding $\mathcal{ALCNI}_{R^+}$-satisfiability. LTCS-Report 98-08, LuFg Theoretical Computer Science, RWTH Aachen, Germany, 1998.

[13] C. Lutz. Complexity of terminological reasoning revisited. In *Proc. LPAR'99*, Springer LNAI 1705, 1999.

[14] B. Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249, 1990.

[15] U. Sattler. A concept language extended with different kinds of transitive roles. In *Proc. KI'96*, Springer LNAI 1137, 1996.

[16] W.J. Savitch. Relationship between nondeterministic and deterministic tape complexities. *J. of Computer and System Sciences*, 4:177–192, 1970.

[17] K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. IJCAI'91*, 1991.

[18] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[19] M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *J. of Computer and System Sciences*, 32:183–221, 1986.

[20] A. Voronkov. How to optimize proof-search in MLs: new methods of proving reduncancy criteria for sequent calculi. *ACM Trans. on Comp. Logic*, 2(2), 2001.