Automata Can Show PSpace Results for Description Logics

Franz Baader[a], Jan Hladik[a], Rafael Peñaloza[b,1]

[a] *Theoretical Computer Science, TU Dresden, Germany*

[b] *Intelligent Systems, University of Leipzig, Germany*

Abstract

In the area of Description Logic (DL), 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 with respect to acyclic terminologies in the DL $\mathcal{SI}$, which extends the basic DL $\mathcal{ALC}$ with transitive and inverse roles.

Key words: Automata Theory, Description Logics, Complexity, PSPACE

1 Introduction

Description Logics (DLs) [1] are a successful family of logic-based knowledge representation formalisms, which can be used to represent the concep-

Email addresses: baader@tcs.inf.tu-dresden.de (Franz Baader), hladik@tcs.inf.tu-dresden.de (Jan Hladik), penaloza@informatik.uni-leipzig.de (Rafael Peñaloza).

[1] Funded by the DFG, Graduiertenkolleg Wissensrepräsentation, University of Leipzig

tual 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. Consequently, 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) [2], results and techniques can often 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 [3,4] and the automata-based [5,6] 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 such a model: tableau-based algorithms try to generate a model in a top-down non-deterministic manner, starting with the root of the tree. In contrast, 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. [2] 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 [7]. However, the automaton constructed in the automata-based approach is usually of exponential size, and thus constructing it explicitly before applying the emptiness test requires exponential time and space. In [8], 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

[2] For an example, see Lemma 2.23 and the subsequent discussion in [1].

in this case, one must apply an appropriate cycle-checking technique, called "blocking" in the DL literature [4]. This is, for example, the case for satisfiability in $\mathcal{ALC}$ with respect to so-called general concept inclusions (GCIs) [9]. 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 $\mathcal{ALC}$ with respect to 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 [10], 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 [11] we show that the so-called inverse tableau method [12] 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 [13] 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 [8] 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 [14]. 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 $\mathcal{SI}$ with respect to acyclic concept definitions is in PSpace is new. It should be noted, however, that we do not view this PSpace-result as the main result of this article, and we do not claim that it could not have been obtained using a different technique. The main contribution of the paper is the general framework for showing PSpace-results using the automata-based approach. The result for $\mathcal{SI}$ just illustrates how this framework can be used.

In order to improve readability of this paper, the more technical proofs have been moved to an appendix.

2 The description logic $\mathcal{SI}$

In Description Logics, concepts are built from concept names (unary predicates) and role names (binary predicates) using 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 $\mathcal{ALC}$ [7], but one can additionally restrict roles to being transitive and use the inverses of roles.[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

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

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.

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

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 $A_1 \doteq C_1$,. . .,$A_n \doteq C_n$ 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 [15].

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* with respect to 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 with respect to $\mathcal{T}$.

For the DL $\mathcal{ALC}$ (i.e. $\mathcal{SI}$ without transitive and inverse roles), it is known that the satisfiability problem is PSpace-complete with respect to acyclic TBoxes [16] and ExpTime-complete with respect to general TBoxes [2]. 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 [17]. For example, the $\mathcal{SI}$ concept A is satisfiable with respect to 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 $\dot{\neg} C$.

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:

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

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 $\dot{\neg} D \sqcup E \in H$ and, for every concept definition $A \doteq D \in \mathcal{T}$, it holds that $A \in H$ implies $D \in H$ and that $\neg A \in H$ implies $\dot{\neg} D \in H$.[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 $\varphi : \{\exists r.D \in \mathsf{sub}(C, \mathcal{T})\} \rightarrow$

[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.

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 therefore contain appropriate dummy 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 $\exists r.D \in \mathsf{sub}(C, \mathcal{T})$ with $\varphi(\exists r.D) = i$:

- if $\exists r.D \in \Omega_0$, then

(1) Γ_i consists of D;
(2) Π_i 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 $\mathsf{trans}(r)$ holds;
(3) for every concept $\forall\overline{r}.F \in \Omega_i$, Ω_0 contains F, and additionally $\forall\overline{r}.F$ if $\mathsf{trans}(r)$ holds;
(4) $\varrho_i = r$;

- if $\exists r.D \notin \Omega_0$, then $\Gamma_i = \Pi_i = \Omega_i = \emptyset$ and $\varrho_i = \lambda$.

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 [18] [5] with the technique for dealing with acyclic TBoxes employed in [8]. A full proof of the next theorem can be found in the appendix.

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

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

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 $\mathcal{SI}$ with respect to general TBoxes in exponential time. Then we will introduce a restricted class of looping automata and use it to show that satisfiability in $\mathcal{SI}$ with respect to acyclic TBoxes can be decided in polynomial space.

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 and $\mathcal{T}$.

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 $(r(v), r(v1), \ldots, r(vk)) \in \Delta$ holds 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 [5,11]: 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 with respect to $\mathcal{T}$.

Proof. This follows from Theorem 6 by a simple induction because the possible labels of the root node of a Hintikka tree t correspond directly to the initial states of the automaton, and the transition relation Δ of $\mathcal{A}_{C,\mathcal{T}}$ consists of all $C, \mathcal{T}$-compatible tuples of state labels. □

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 observation immediately yields:

Theorem 10 Satisfiability in $\mathcal{SI}$ with respect to 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 [2].

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 with respect to *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 [19]. 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 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.

[6] This is similar to the so-called trace technique for tableau-based algorithms [7].

```
if $I \neq \emptyset$ then
    guess an initial state $q \in I$
else
    return "empty"
end if
if there is a transition from $q$ then
    guess such a transition $(q, q_1, \ldots, q_k) \in \Delta$
    $\mathsf{push}(\mathsf{SQ}, (q_1, \ldots, q_k)), \mathsf{push}(\mathsf{SN}, 0)$
else
    return "empty"
end if
while $\mathsf{SN}$ is not empty do
    $(q_1, \ldots, q_k) := \mathsf{pop}(\mathsf{SQ}), n := \mathsf{pop}(\mathsf{SN}) + 1$
    if $n \leq k$ then
        $\mathsf{push}(\mathsf{SQ}, (q_1, \ldots, q_k)), \mathsf{push}(\mathsf{SN}, n)$
        if $\mathsf{length}(\mathsf{SN}) < m - 1$ then
            if there is a transition from $q_n$ then
                guess a transition $(q_n, q'_1, \ldots, q'_k) \in \Delta$
                $\mathsf{push}(\mathsf{SQ}, (q'_1, \ldots, q'_k)), \mathsf{push}(\mathsf{SN}, 0)$
            else
                return "empty"
            end if
        end if
    end if
end while
return "not empty"
```

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

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 [19].

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 $\mathcal{SI}$ with respect to acyclic TBoxes

We will now show how Theorem 14 can be used to prove that $\mathcal{SI}$ concept satisfiability with respect to acyclic TBoxes is in PSPACE, which illustrates how such results can be elegantly achieved using blocking automata. [7] 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* with respect to $\mathcal{A}$ 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 $(f_q(q_0), f_q(q_1), \ldots, f_q(q_k)) \in \Delta$. [8]

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

[7] As already mentioned in the introduction, we do not claim that this result could not be obtained using other techniques. For example, a PSPACE tableau algorithm similar to the one in [18] could probably also be developed. A PSPACE-result for satisfiability in $\mathsf{S4_m}$ with converse modalities, the modal logic corresponding to $\mathcal{SI}$, may also be available somewhere in the extensive modal logic literature, but acyclic TBoxes are not considered in modal logics.
[8] 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.

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 with respect to $\mathcal{T}$ is the maximal nesting of (universal and existential) role restrictions in the concept obtained by expanding C with respect to $\mathcal{T}$. 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 $\mathsf{rd}_{\mathcal{T}}(S)$ with respect to $\mathcal{T}$ is the maximal role depth with respect to $\mathcal{T}$ 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 $(q, q_1, \dots, q_k)$ of $\mathcal{A}_{C,\mathcal{T}}$, only the existence 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 with respect to $\mathcal{A}_{C,\mathcal{T}}$.

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 with respect to $\hookleftarrow_{\mathcal{SI}}$. The main reasons for this to hold are the following (details can be found in the appendix):

- if a successor node is reached with respect to 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;
- if a successor node is reached with respect to a transitive role, then there is an inclusion relationship between the Π-components of the successor node and its father; and
- the same is true (though in the other direction) for the $\Omega/\overline{\varrho}$-components.

Since we know that C is satisfiable with respect to $\mathcal{T}$ 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 with respect to the empty TBox in $\mathcal{ALC}$ [7].

Theorem 20 Satisfiability in $\mathcal{SI}$ with respect to 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 non-deterministic top-down test that can be interleaved with the construction of the automaton and stopped 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 with respect to acyclic TBoxes in PSpace.

References

[1] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. F. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation, and Applications, 2nd Edition, Cambridge University Press, 2007.

[2] K. Schild, A correspondence theory for terminological logics: Preliminary report, in: J. Mylopoulos, R. Reiter (Eds.), Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI-91), Morgan Kaufmann, Los Altos, 1991.

[3] M. Fitting, Proof Methods for Modal and Intuitionistic Logics, Reidel, 1983.

[4] F. Baader, U. Sattler, An overview of tableau algorithms for description logics, Studia Logica 69 (2001) 5–40, an abridged version appeared in *Tableaux 2000*, volume 1847 of LNAI, 2000. Springer-Verlag.

[5] M. Y. Vardi, P. Wolper, Automata-theoretic techniques for modal logics of programs, Journal of Computer and System Science 32 (1986) 183–221.

[6] D. Calvanese, G. De Giacomo, M. Lenzerini, Reasoning in expressive description logics with fixpoints based on automata on infinite trees, in: T. Dean (Ed.), Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI-99), Morgan Kaufmann, Los Altos, 1999.

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

[8] J. Hladik, R. Peñaloza, PSPACE automata for description logics, in: B. Parsia, U. Sattler, D. Toman (Eds.), Proc. of the 2006 Description Logic Workshop (DL 2006), Vol. 189 of CEUR Proceedings, 2006, available from `ceur-ws.org`.

[9] F. Baader, H.-J. Bürckert, B. Hollunder, W. Nutt, J. H. Siekmann, Concept logics, in: J. W. Lloyd (Ed.), Computational Logics, Symposium Proceedings, Springer-Verlag, 1990, pp. 177–201.

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

[11] F. Baader, S. Tobies, The inverse method implements the automata approach for modal satisfiability, in: R. Goré, A. Leitsch, T. Nipkow (Eds.), Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR-01), Vol. 2083 of Lecture Notes in Artificial Intelligence, Springer-Verlag, 2001.

[12] A. Voronkov, How to optimize proof-search in modal logics: new methods of proving reduncancy criteria for sequent calculi, ACM Transactions on Computational Logic 2 (2).

[13] F. Baader, J. Hladik, C. Lutz, F. Wolter, From tableaux to automata for description logics, Fundamenta Informaticae 57 (2003) 1–33.

[14] U. Sattler, A concept language extended with different kinds of transitive roles, in: G. Görz, S. Hölldobler (Eds.), Proc. of the 20th German Annual Conf. on Artificial Intelligence (KI'96), No. 1137 in Lecture Notes in Artificial Intelligence, Springer-Verlag, 1996.

[15] B. Nebel, Terminological reasoning is inherently intractable, Artificial Intelligence 43 (1990) 235–249.

[16] C. Lutz, Complexity of terminological reasoning revisited, in: Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99), Vol. 1705 of Lecture Notes in Artificial Intelligence, Springer-Verlag, 1999, pp. 181–200.

[17] P. Blackburn, M. de Rijke, Y. Venema, Modal Logic, Vol. 53 of Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 2001.

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

[19] W. J. Savitch, Relationship between nondeterministic and deterministic tape complexities, Journal of Computer and System Science 4 (1970) 177–192.

Appendix

In this appendix we present the proofs for some of the more technical lemmas and theorems. We begin with the theorem stating that the existence of Hintikka trees (see Definition 5) characterises satisfiability of $\mathcal{SI}$ concepts. Please recall that the nodes in our Hintikka trees are labelled with quadruples $(\Gamma, \Pi, \Omega, \varrho)$, where Ω contains the complete Hintikka set for the node, ϱ denotes the role by which the node is connected with its father, and the remaining elements Γ and Π consist of the subsets of Ω that are required in order to satisfy existential (Γ) and universal (Π) concepts in the father node.

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

Proof. For a node v with $t(v) = (\Gamma, \Pi, \Omega, \varrho)$, we will refer to the components as $\Gamma(v)$, $\Pi(v)$ etc.

For the "if" direction, we will show how to construct a model $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ from a Hintikka tree t. Let $\Delta^{\mathcal{I}} = \{v \in K^* \mid t(v) \neq (\emptyset, \emptyset, \emptyset, \lambda)\}$. For a role name $r \in N_R \setminus N_T$, we define $r^{\mathcal{I}} = \{(v, w) \mid w \text{ is an } r\text{-neighbour of } v\}$. If $r \in N_T$, we define $r^{\mathcal{I}}$ as the transitive closure of this relation.

For a primitive concept name A, we define $A^{\mathcal{I}} = \{v \in \Delta^{\mathcal{I}} \mid A \in \Omega(v)\}$. In order to show that this interpretation can be extended to defined concept names and that it interprets complex concepts correctly, we define a weight function $o(C)$ for concept terms C as follows:

- $o(A) = 0$ for a primitive concept name A;
- $o(B) = o(C) + 1$ for a defined concept name $B \doteq C$.
- $o(\neg A) = o(A) + 1$ for the negation of a (primitive or defined) concept name;
- $o(C \sqcap D) = o(C \sqcup D) = \max\{o(C), o(D)\} + 1$;
- $o(\exists r.C) = o(\forall r.C) = o(C) + 1$.

Note that o is defined differently from the role depth for the Boolean operators and defined concept names in order to ensure that subconcepts or definitions of a concept have a lower weight than the concept itself. However, o is also well-founded if $\mathcal{T}$ is acyclic. We can now show by induction over the weight of the appearing concepts that if $D \in \Omega(v)$, then $v \in D^{\mathcal{I}}$.

- If $A \in \Omega(v)$ for a primitive concept name A then $v \in A^{\mathcal{I}}$ holds by definition.
- If $B \in \Omega(v)$ for a defined concept name $B \doteq C$, we know that $C \in \Omega(v)$ because $\Omega(v)$ is $\mathcal{T}$-expanded. Since $o(C) < o(B)$, it follows by induction that $v \in C^{\mathcal{I}}$ holds. Thus we can define $B^{\mathcal{I}} = C^{\mathcal{I}}$ and obtain $v \in B^{\mathcal{I}}$.
- If $\neg A \in \Omega(v)$ for a negated concept name then $A \notin \Omega(v)$ holds because $\Omega(v)$ is a Hintikka set. If A is primitive, this implies that $v \notin A^{\mathcal{I}}$ holds and we are done. If A is a defined concept name and $A \doteq E$ then, as in the previous case, $\dot\neg E \in \Omega(v)$ holds because $\Omega(v)$ is $\mathcal{T}$-expanded. Again, $o(\dot\neg E) < o(\neg A)$ implies $v \in (\dot\neg E)^{\mathcal{I}}$ by induction and, since $(\dot\neg E)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus E^{\mathcal{I}}$ and $A^{\mathcal{I}} = E^{\mathcal{I}}$, it follows that $v \notin A^{\mathcal{I}}$ holds.
- If $E \sqcap F \in \Omega(v)$ then, since $\Omega(v)$ is a Hintikka set, it contains E and F, and by induction $v \in E^{\mathcal{I}} \cap F^{\mathcal{I}}$ holds.
- If $E \sqcup F \in \Omega(v)$ then $v \in E^{\mathcal{I}} \cup F^{\mathcal{I}}$ follows from an analogous argument.
- If $\exists r.E \in \Omega(v)$ for a role name r then, since t is a Hintikka tree, $(v, v \cdot \varphi(\exists r.E)) \in r^{\mathcal{I}}$ and $E \in \Omega(v \cdot \varphi(\exists r.E))$ (inverse roles can be treated analogously), thus by induction $v \in (\exists r.E)^{\mathcal{I}}$ holds.
- If $\forall r.E \in \Omega(v)$ for a role r and $(v, w) \in r^{\mathcal{I}}$, then $(v, w) \in r^{\mathcal{I}}$ holds either because w is an r-neighbour of v in the Hintikka tree, in which case $E \in \Omega(w)$ holds by definition of $C, \mathcal{T}$-compatible, or r is a transitive role and (v, w) is in the transitive closure of the relation defined above. In this case, there exists a sequence of tree nodes $v = v_0, v_1, \ldots, v_{f-1}, v_f = w$ such that for every $i < f$, v_{i+1} is an r-neighbour of v_i. Since $\mathsf{trans}(r)$ holds, every node label $t(v_i)$ for $1 \leq i \leq t$ contains $\forall r.E$ and E because of the definition of $C, \mathcal{T}$-compatible, thus it follows by induction that $w \in E^{\mathcal{I}}$ and $v \in (\forall r.E)^{\mathcal{I}}$.

For a GCI $E \sqsubseteq F$, $\Omega(v)$ contains $\dot\neg E \sqcup F$ for every node v. As $\Omega(v)$ is a Hintikka set, it contains F or $\dot\neg E$. If it contains F then, as we have just shown, v belongs to $F^{\mathcal{I}}$. Otherwise, $\Omega(v)$ contains $\dot\neg E$, which implies $v \notin E^{\mathcal{I}}$ as in the case of negated concept names above. Consequently, every node $v \in E^{\mathcal{I}}$ is also contained in $F^{\mathcal{I}}$.

For the "only-if" direction, we show how a model $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ for C with respect to $\mathcal{T}$ can be used to define a $C, \mathcal{T}$-compatible Hintikka tree t with $C \in \Omega(\varepsilon)$. Let k be the number of existential concepts in $\mathsf{sub}(C, \mathcal{T})$ and φ be a function as in Definition 5. We inductively define a function $\vartheta : K^* \to \Delta^{\mathcal{I}} \cup \{\psi\}$ for a new individual ψ such that $\vartheta(v)$ satisfies all concepts in $\Omega(v)$.

Since $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a model, there exists an element $d_0 \in \Delta^{\mathcal{I}}$ with $d_0 \in C^{\mathcal{I}}$. So we define $\vartheta(\varepsilon) = d_0$ and set $\Gamma(\varepsilon) = \Pi(\varepsilon) = \emptyset$, $\Omega(\varepsilon) = \{E \in \mathsf{sub}(C, \mathcal{T}) \mid d_0 \in E^{\mathcal{I}}\}$, and $\varrho(\varepsilon) = \lambda$. Then we inductively define, for every node v for which ϑ is already defined, the labels of $v \cdot i, 1 \leq i \leq k$, as follows: if $\Omega(v)$ contains the existential concept $\exists r.E$ with $i = \varphi(\exists r.E)$ then, since $\vartheta(v)$ satisfies $\exists r.E$, there exists a $d \in \Delta^{\mathcal{I}}$ with $(\vartheta(v), d) \in r^{\mathcal{I}}$ and $d \in E^{\mathcal{I}}$, and thus we set $\vartheta(v \cdot i) = d$, $\Omega(v \cdot i) = \{F \in \mathsf{sub}(C, \mathcal{T}) \mid d \in F^{\mathcal{I}}\}$, $\varrho(v \cdot i) = r$, $\Gamma(v \cdot i) = \{\mathrm{E}\}$, and $\Pi(v \cdot i)$ contains every F with $\forall r.F \in \Omega(v)$ and, if r is transitive, additionally $\forall r.F$. If $\vartheta(v)$ does not belong to $(\exists r.E)^{\mathcal{I}}$, we define $\vartheta(v \cdot i) = \psi$ and $(\Gamma(v \cdot i), \Pi(v \cdot i), \Omega(v \cdot i), \varrho(v \cdot i)) = (\emptyset, \emptyset, \emptyset, \lambda)$.

It follows by construction that $\Gamma(v \cdot i)$ and $\Pi(v \cdot i)$ are subsets of $\Omega(v \cdot i)$ and that the tuple $((\Gamma(v), \Pi(v), \Omega(v), \varrho(v)), (\Gamma(v \cdot 1), \Pi(v \cdot i), \Omega(v \cdot 1), \varrho(v \cdot 1)), \ldots,$ $(\Gamma(v \cdot k), \Pi(v \cdot k), \Omega(v \cdot k), \varrho(v \cdot k)))$ is $C, \mathcal{T}$-compatible. Note that for every $v \in K^*$, $\Omega(v)$ is a Hintikka set since it follows from the fact that $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a model that $d \in (E \sqcup [\sqcap] F)^{\mathcal{I}}$ implies $d \in E^{\mathcal{I}} \cup [\cap] F^{\mathcal{I}}$, and that $d \in E^{\mathcal{I}}$ holds iff $d \notin (\neg E)^{\mathcal{I}}$ holds. □

After establishing that the automaton $\mathcal{A}_{C,\mathcal{T}}$ has an accepting run iff there exists a Hintikka tree for C and $\mathcal{T}$ (Lemma 9), and thus that we can use the emptiness test for $\mathcal{A}_{C,\mathcal{T}}$ in order to decide satisfiability of C with respect to $\mathcal{T}$, we want to show that we can restrict our attention to partial runs of depth m in the case that $\mathcal{A}_{C,\mathcal{T}}$ is m-blocking. For this purpose, we show how a partial run can be unravelled into a complete run.

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.

Proof. The "only if" direction is trivial, so only the "if" direction will be proved. For this purpose, we will show how to construct a complete successful run from a partial one by replacing, for every blocked node $v \hookleftarrow w$, the subtree starting at v with the subtree starting at w.

Suppose there is a successful partial run r of depth m. This run will be used to construct a function $\beta : K^* \to K^{\leq m}$ inductively as defined below. The

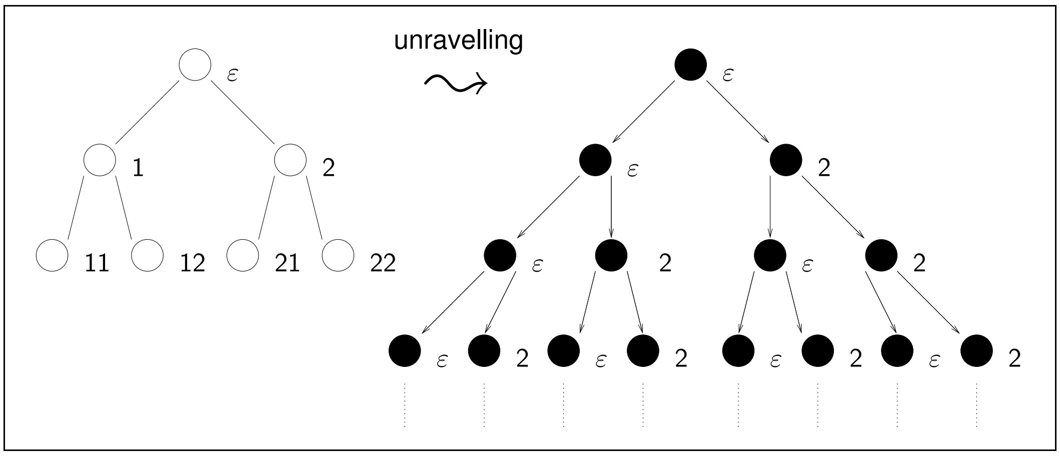

Fig. 2. Unravelling of a partial run.

intuitive meaning of $\beta(v) = w$ is "w stands for v", i.e., we will use the labels of w and w's successors in the partial run also for v and v's successors in the complete run.

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

Figure 2 shows an example for a partial run of a 3-blocking automaton on a binary tree on the left and its unravelling on the right, where the nodes in the unravelled tree are labelled with their respective beta values. We assume that the nodes 1 and 21 are blocked by ε and that node 22 is blocked by node 2. As an example, we consider the values of β for the successors of node 21, where $\beta(21) = \varepsilon$. To determine the values of the successors, we have to test if the successors of ε are blocked. For node 211, since $\varepsilon \cdot 1$ is blocked by ε, it turns out that $\beta(211)$ equals ε. On the other hand, for node 212 the corresponding node $\varepsilon \cdot 2$ is not blocked, thus $\beta(212)$ equals 2.

In the following, we will refer to (direct or indirect) successors of blocked nodes as *indirectly blocked*. Notice that the range of β does not contain any blocked or indirectly blocked nodes, since we start with an non-blocked node and, whenever we encounter a blocked node, we replace it and its successors with the blocking one and its successors. (In the example of the unravelled tree, only the labels ε and 2 appear, which are the only unblocked nodes in the partial run.) Moreover, for every node v with $\beta(v) \neq v$, the depth of v, $|v|$, is larger than $|\beta(v)|$, because β maps a blocked node to a predecessor and the child of a blocked node to a child of the predecessor etc.

We will now show by induction over $|v|$ that the function β is well-defined, more precisely that $|\beta(v)| < m$ for all $v \in K^*$, and that we can use β to construct a successful run s from the successful partial run r by setting, for every node v, $s(v) := r(\beta(v))$. For the root, $s(\varepsilon) = r(\varepsilon)$ holds, thus both s

and r start with the same label. If, for any node v, the successors of v are not blocked, then the transition $(s(v), s(v\cdot 1), \ldots, s(v\cdot k))$ is contained in Δ because $(r(\beta(v)), r(\beta(v)\cdot 1), \ldots, r(\beta(v)\cdot k))$ is a transition in the run r. In this case, since $\beta(v)$ is not blocked or indirectly blocked, $|\beta(v)\cdot i| < m$ for all $1 \leq i \leq k$, because otherwise the path to $\beta(v)\cdot i$ would have length at least m without containing a blocked node, in contradiction with the induction hypothesis that the part of s constructed so far is part of a successful run and that neither $\beta(v)$ nor any of its predecessors is blocked.

If any successors of v are blocked, i.e. $r(v\cdot i) \leftarrowtail r(w)$ then $(r(\beta(v)), r(\beta(v)\cdot 1), \ldots, r(\beta(v)\cdot i), \ldots, r(\beta(v)\cdot k)) \in \Delta$ implies $(r(\beta(v)), r(\beta(v)\cdot 1), \ldots, r(\beta(w)), \ldots, r(\beta(v)\cdot k)) \in \Delta$ because of the definition of $\leftarrowtail$-invariance. Hence, $(s(v), s(v\cdot 1), \ldots, s(v\cdot k)) \in \Delta$, and s is a successful run of $\mathcal{A}$. In this case, since w is a predecessor of $\beta(v)\cdot i$ and $|\beta(v)| < m$, it holds that $|w| < m$, and thus $|\beta(v\cdot i)| < m$. Observe that w cannot be blocked itself because $\beta(v)$ is a successor of w or equal to w and the range of β does not contain blocked or indirectly blocked nodes, thus the range of β only contains non-blocked nodes. $\square$

The fact that we only have to consider partial runs for m-blocking automata is the key to proving the main result of this paper: the conditions for a PSpace *on-the-fly construction* (see Definition 13; in short: the arity, the blocking distance, and the size of every state must be polynomial; and guessing an initial state and a transition may only require polynomial space) ensure that the corresponding problem is in PSpace.

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 $\mathfrak{i}$.

Proof. We will first show by induction that if the algorithm described in Figure 1 answers "not empty", then the we can define a successful partial run r from the q_i values used by the algorithm. Since the algorithm answers "not empty", there is an initial transition $(q, q_1, \ldots, q_k)$. Then set $r(\varepsilon) = q$ and $r(i) = q_i$ for all $1 \leq i \leq k$. Suppose now that the algorithm visits a node $v = a_0 \cdot \ldots \cdot a_\ell \in K^*$. Then, by induction hypothesis, r is defined for the previously visited nodes. If $\mathsf{length}(SP) < k$, then the algorithm guesses a transition, and $r(v\cdot i) = q'_i$ defines a transition in the run. Otherwise, the algorithm has reached depth m, so we have reached the maximum depth of the partial run.

Conversely, if there is a successful partial run r, then it is possible to guess the initial state, and initial transition $(r(\varepsilon), r(1), \ldots, r(k))$. By Definition 13, the space required for guessing the initial state $r(\varepsilon)$ and the transition from

$r(\varepsilon)$ is bounded by $P(n)$. When the algorithm visits one of these initial nodes, they have the same labels as in r. Now suppose the algorithm visits a node v with $r(v) = q$. If the length of v is smaller than m, then there is a transition on $r, (r(v), r(v \cdot 1), \ldots, r(v \cdot k))$ which the algorithm can guess (using space bounded by $P(n)$) and so it will not return "empty". At any time, the stack SQ contains at most $m_\mathfrak{i}$ tuples of $k_\mathfrak{i}$ states and SN contains at most $m_\mathfrak{i}$ numbers between 0 and $k_\mathfrak{i}$. Since $m_\mathfrak{i}$, $k_\mathfrak{i}$ and the size of each state are bounded by $P(m)$, the space used by these stacks is polynomial in the size of $\mathfrak{i}$.

It follows from Lemma 12 that this emptiness test is sound and complete. From Savitch's theorem [19] we obtain the deterministic complexity class. $\square$

In order to apply this theorem to our automata algorithm for $\mathcal{SI}$, we require the construction of *faithful* (see Definition 15) subautomata. The following theorem shows the general result that testing a faithful subautomaton for emptiness is sufficient to decide emptiness of the original automaton.

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

Proof. Since every successful run of $\mathcal{A}^{\mathrm{S}}$ is also a successful run of $\mathcal{A}$, the "if" direction is obvious. For the "only if" direction, we will show how to transform a successful run r of $\mathcal{A}$ into a successful run s of $\mathcal{A}^{\mathrm{S}}$. To do this, we traverse r breadth-first, creating an intermediate run $\hat{r}$, which initially is equal to r. At every node $v \in K^*$, we replace the labels of the direct and indirect successors of v with their respective $f_{\hat{r}(v)}$ values (see Definition 15). More formally, at node v, we replace $\hat{r}(w)$ with $f_{\hat{r}(v)}(\hat{r}(w))$ for all $w \in \{v \cdot u \mid u \in K^+\}$. By Definition 15, $\hat{r}$ is still a successful run after the replacement (note that condition 2 is necessary to ensure transitions from the successors of v). We define s as the value of $\hat{r}$ "in the limit", i.e., for every node v, $s(v)$ has the value of $\hat{r}(v)$ after v has been processed. $\square$

After the general result, we show that in our special case, the automaton $\mathcal{A}^{\mathrm{S}}_{C,\mathcal{T}}$, which uses the functions f_q in order to avoid transitions to states with a larger role depth, is a faithful subautomaton of $\mathcal{A}_{C,\mathcal{T}}$.

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

Proof. We have to show that both conditions of Definition 15 are satisfied.

Condition 1. The case that a successor is replaced by $(\emptyset, \emptyset, \emptyset, \lambda)$ cannot occur because in every successor q_i of q, the role depth of Γ_i is strictly smaller than the maximum depth of Ω. Assume that $(q, q_1, \ldots, q_k) \in \Delta$. To prove that $(q, f_q(q_1), \ldots, f_q(q_k))$ is also contained in Δ, we have to show that this transition satisfies the conditions for $C, \mathcal{T}$-compatibility in Definition 5. Number 1 and 4 are obvious. Number 3 holds because we do not remove anything from Ω. Finally, we do not remove any concepts from the Π_i sets, because these concepts have a maximum depth of $\mathsf{rd}_{\mathcal{T}}(\Omega)$, if ϱ_i is transitive, or $\mathsf{rd}_{\mathcal{T}}(\Omega) - 1$, otherwise. Thus, we only remove concepts from Ω_i, and none of the removed concepts is required.

Condition 2. Let $(q_0, q_1, \ldots, q_k) \in \Delta$. If for some $i > 0$ with $\varphi(\exists r.D) = i$, q_i is replaced by $(\emptyset, \emptyset, \emptyset, \lambda)$, this means that for the concept $D \in \Gamma_i$, $\mathsf{rd}_{\mathcal{T}}(D) \geq \mathsf{rd}_{\mathcal{T}}(\Omega)$. This implies that the corresponding existential concept $\exists r.D$ in Ω_0 has a depth which is strictly larger than $\mathsf{rd}_{\mathcal{T}}(\Omega)$, and therefore will be removed from $f_q(q_0)$. Otherwise, we again have to show the four conditions from Definition 5. Number 1 and 4 are again obvious. For number 3, observe that if $\forall \overline{r}.F \in f_q(\Omega_i)$ with $\varrho_i = r$, then $\mathsf{rd}_{\mathcal{T}}(\forall \overline{r}.F) < n$ because $\overline{r} \neq \varrho_i$, and thus neither F nor $\forall \overline{r}.F$ will be removed from Ω_0. For number 2, if $\forall r.E \in f_q(\Omega_0)$, then it holds either that $\mathsf{rd}_{\mathcal{T}}(\forall r.E) < n$ or $\mathsf{rd}_{\mathcal{T}}(\forall r.E) = n$ and $\mathsf{trans}(r)$. In the former case, neither E nor $\forall r.E$ will be removed from Π_i. In the latter case, $\forall r.E$ will not be removed because $\varrho_i = r$ and $\mathsf{trans}(r)$ holds. □

Finally, we prove that we can apply our framework of PSpace on-the-fly constructions to the faithful subautomaton $\mathcal{A}^{\mathrm{S}}_{C,\mathcal{T}}$ by defining an appropriate blocking condition and showing that $\mathcal{A}^{\mathrm{S}}_{C,\mathcal{T}}$ is n^4-blocking, where n is the size of the input. From this lemma and Theorem 14, it follows directly that $\mathcal{SI}$ satisfiability with respect to acyclic TBoxes is in PSpace.

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.

Proof. Let $\mathfrak{i} = (C, \mathcal{T})$ be an input, i.e. an $\mathcal{SI}$ concept and TBox, and let $|\mathfrak{i}|$ be the length of $\mathfrak{i}$. The blocking relation $\hookleftarrow_{\mathcal{SI}}$ is defined as follows: $(\Gamma_1, \Pi_1, \Omega_1, \varrho_1) \hookleftarrow_{\mathcal{SI}} (\Gamma_2, \Pi_2, \Omega_2, \varrho_2)$ if $\Gamma_1 = \Gamma_2$, $\Pi_1 = \Pi_2$, $\Omega_1/\overline{\varrho}_1 = \Omega_2/\overline{\varrho}_2$, and $\varrho_1 = \varrho_2$. We have to show that there is a polynomial $P(n)$ satisfying the conditions in Definition 13.

Every element of $Q_\mathfrak{i}$ is of a size bounded by $P(n)$. Every state label is a subset of $\mathsf{sub}(C, \mathcal{T})$ and therefore bounded by the size of $\mathsf{sub}(C, \mathcal{T})$, which is linear in $|\mathfrak{i}|$. The size of each of these elements, in turn, is bounded by $|\mathfrak{i}|$. Thus, the size of each node label is at most quadratic in the size of the input.

There is a $P(n)$-space bounded non-deterministic algorithm for guessing an initial state or successor states for a given state. This is obvious, since the size of every state is bounded by $|\mathfrak{i}|^2$ and all necessary information for the successor states can be obtained from the current state.

The automaton $\mathcal{A}^S_{C,\mathcal{T}}$ is operating on $k_\mathfrak{i}$-ary trees and $m_\mathfrak{i}$-blocking, with $m_\mathfrak{i} \leq P(n)$ and $k_\mathfrak{i} \leq P(n)$. The tree width $k_\mathfrak{i}$ is bounded by the number of existential subconcepts of $\mathfrak{i}$ and therefore by $|\mathfrak{i}|$. In order to show a polynomial bound for $m_\mathfrak{i}$, we first have to show that $\mathcal{A}^{\mathrm{S}}_{C,\mathcal{T}}$ is $\hookleftarrow_{\mathcal{SI}}$-invariant. For states $\{q, q_i\} \subseteq Q^{\mathrm{S}}$ with $q = (\Gamma, \Pi, \Omega, \varrho)$ and $q_i = (\Gamma_i, \Pi_i, \Omega_i, \varrho_i)$ let $(q_0, \ldots, q_j, \ldots, q_k)$ be a transition and $q_j \hookleftarrow_{\mathcal{SI}} q_i$. Then the tuple $(q_0, \ldots, q_i, \ldots, q_k)$ is also $C, \mathcal{T}$-compatible since $\Gamma_j = \Gamma_i$, $\Pi_j = \Pi_i$, $\varrho_j = \varrho_i$ and Ω_j contains the same universal concepts involving $\overline{\varrho}_j$ as Ω_i.

What is the maximum depth of a blocked node in a successful run? Firstly, observe that transitions $(q, q_1, \ldots, q', \ldots, q_k)$ with $q = (\Gamma, \Pi, \Omega, \varrho)$ and $q' = (\Gamma', \Pi', \Omega', \varrho')$ where ϱ' is different from ϱ or not transitive decrease the maximum depth of concepts contained in the state: if ϱ' is not transitive, then $\mathsf{rd}_\mathcal{T}(\Omega')$ is smaller than $\mathsf{rd}_\mathcal{T}(\Omega)$ by definition. If ϱ' is transitive, but different from ϱ, then Ω' can only have concepts of depth $\mathsf{rd}_\mathcal{T}(\Omega)$ if these start with $\forall\varrho'$. Similarly, Ω can only contain concepts of the same depth as its predecessor state if they begin with $\forall\varrho$, which implies that the role depth decreases after two transitions. (This is the key to obtaining a polynomial bound, and it does not hold for general TBoxes, where the GCIs maintain the same role depth in every node.) This depth is bounded by the maximum depth in $\mathsf{sub}(C,\mathcal{T})$ and therefore by $|\mathfrak{i}|$, thus there is a linear bound for the number of such steps before depth 0 is reached. After this point, the path will contain a blocked node, since all further nodes are labelled with $(\emptyset, \emptyset, \emptyset, \lambda)$.

So the role depth can only remain the same along a subpath (a subpath is a path which does not need to begin at ε) where every transition involves the same transitive role r. From the definition of Δ, it follows for any subpath with labels $(\Gamma_0, \Pi_0, \Omega_0, r)$, $(\Gamma_1, \Pi_1, \Omega_1, r)$, $\ldots$, $(\Gamma_\ell, \Pi_\ell, \Omega_\ell, r)$ that $\Pi_i \subseteq \Pi_{i+1}$, for all $1 \leq i \leq \ell - 1$, so the number of different sets Π_i is bounded by $|\mathfrak{i}|$. By the same argument, it also holds on this subpath that $\Omega_{i+1}/\overline{r} \subseteq \Omega_i/\overline{r}, 1 \leq i \leq \ell - 1$. Once again, it is only possible to have a subpath of length $|\mathfrak{i}|$ with different sets. Finally, since Γ_i contains only one concept, there is also only a linear number of possibilities for this set. In total, every r-subpath of length larger than $|\mathfrak{i}|^3$ must have $i < j$ such that $\Gamma_j = \Gamma_i$, $\Pi_j = \Pi_i$ and $\Omega_j/\overline{r} = \Omega_i/\overline{r}$, and hence $(\Gamma_j, \Pi_j, \Omega_j, r) \hookleftarrow (\Gamma_i, \Pi_j, \Omega_i, r)$. Thus, an r-subpath for a transitive role r either contains a blocked node or is shorter than $|\mathfrak{i}|^3$ and therefore followed by a transition with a role different from r, which decreases the maximum depth of concepts contained in Ω. Altogether, we obtain that every path which is longer than $|\mathfrak{i}|^4$ contains a blocked node. This concludes the proof that the construction of $\mathcal{A}^{\mathrm{S}}_{C,\mathcal{T}}$ is a PSPACE on-the-fly construction with $P(n) = n^4$. $\square$