Journal of Automated Reasoning manuscript No.
(will be inserted by the editor)

A PSpace Algorithm for Acyclic Epistemic DL $\mathcal{ALCS}5_m$

Jia Tao

Received: date / Accepted: date

Abstract We study the description language $\mathcal{ALCS}5_m$, a variant of $\mathcal{ALCK}_m$ and $\mathcal{ALCS}4_m$. It augments $\mathcal{ALC}$ by allowing multi-modal epistemic operators over concept and role expressions. The epistemic operators are interpreted in modal logic $\mathbf{S}5_m$. By examining design issues of the tableau algorithm specific for $\mathcal{ALCS}5_m$, different from those for $\mathcal{ALCK}_m$ and $\mathcal{ALCS}4_m$, we provide a sound and complete tableau algorithm for deciding the satisfiability of an $\mathcal{ALCS}5_m$ knowledge base with an acyclic TBox, and further show how it can be implemented in PSpace.

Keywords Epistemic Logic · Description Logic · $\mathcal{ALC}$ · Tableau Algorithm · PSpace

1 Introduction

Epistemic logic $\mathbf{S}5_m$ has been viewed as most appropriate for many epistemic applications [1] and Description Logics (DLs) [2] offer a powerful formalism for representing and reasoning with knowledge in a broad range of applications. Combining these two kinds of logics often yields more expressive logics for complicated applications. For example, by extending DL $\mathcal{ALCQI}$ with a single $\mathbf{S}5$ modality in front of both concepts and roles, the resulting logic $\mathbf{S}5_{\mathcal{ALCQI}}$ is able to reason about change by allowing to express the change of concepts and roles over time [3]. Due to the expressivity of the language, the satisfiability of $\mathbf{S}5_{\mathcal{ALCQI}}$ concepts with respect to general TBoxes is 2-ExpTime-complete.

In this article, we study the logic $\mathcal{ALCS}5_m$ that extends the description logic $\mathcal{ALC}$ by adding modal operators of the logic $\mathbf{S}5_m$. Given a knowledge base Σ, one may study the following problems:

Jia Tao
Department of Computer Science, Lafayette College, Easton, PA, USA
E-mail: taoj@lafayette.edu

1. Knowledge base satisfiability: Σ is satisfiable if it has a model.
2. Concept satisfiability: a concept C is satisfiable with respect to Σ if there is a model of Σ in which the interpretation of C is not empty.
3. Concept subsumption: a concept C is subsumed by a concept D with respect to Σ if for every model of Σ the interpretation of C is a subset of the interpretation of D.
4. Instance checking: an individual a is an instance of a concept C if the assertion $C(a)$ is satisfied in every model of Σ.

Since problems 2-4 can be reduced to the knowledge base satisfiability problem in linear time [4], we focus on the knowledge base satisfiability problem for $\mathcal{ALCS}5_m$, an extension of modalized description logics $\mathcal{ALCK}_m$ and $\mathcal{ALCS}4_m$. Both $\mathcal{ALCK}_m$ and $\mathcal{ALCS}4_m$ are sublanguages of $\mathbf{K}_{\mathcal{ALC}}$ [5], a language obtained by extending $\mathcal{ALC}$ with modal operators being used both inside concept expressions and in front of assertions and terminological axioms but not in front of roles. Whereas the satisfiability problem for $\mathbf{K}_{\mathcal{ALC}}$ is NExpTime-complete, by considering only acyclic TBoxes and restricting modal operators to concept expressions, we were able to provide a PSpace algorithm for $\mathcal{ALCK}_m$ instance checking problem. The PSpace result was further extended to the case of $\mathcal{ALCS}4_m$, a variant of $\mathcal{ALCK}_m$ that requires the multi-modal operators to satisfy the Truth and Positive Introspection axioms [6].

We continue our study, extend the syntax of $\mathcal{ALCK}_m$ and $\mathcal{ALCS}4_m$ by adding modalized roles, and interpret the syntax with the modal logic $\mathbf{S}5_m$. The resulting language is termed $\mathcal{ALCS}5_m$ and is intended to express epistemic statements. For instance, an $\mathcal{ALCS}5_m$ concept $\Box_i Student$ represents all the individuals known to be students by agent i, assertion $\Box_i Student(a)$ should be thought of as individual a belonging to the concept $\Box_i Student$, i.e., agent i knows that a is a student, and assertion $\Box_i lives(Aria, PA)$ means that agent i knows that Aria lives in Pennsylvania. We can also express statements that are not tied to specific individuals. As an example, $Student \sqsubseteq \Box_i \exists HasFriend.Student$ says that if an individual is a student, then agent i knows that the individual has a friend who is a student. However, statements that express knowledge of general rules such as "agent i knows that students at Bryn Mawr College are female" which can be formulated as $\Box_i(Student \sqcap BMC \sqsubseteq Female)$ is not expressible in $\mathcal{ALCS}5_m$ and will not be dealt with in this article.

By modifying the sound and complete algorithms for deciding the satisfiability of an $\mathcal{ALCK}_m$ or $\mathcal{ALCS}4_m$ knowledge base [6], we provide a sound and complete algorithm for deciding the satisfiability of an $\mathcal{ALCS}5_m$ knowledge base with an acyclic TBox using a set of tableau expansion rules. Since accessibility relations in $\mathbf{S}5_m$-models are equivalence relations, expansion rules [7,8, 5] have to be designed to syntactically enforce the symmetry property present in $\mathbf{S}5_m$-models but not necessarily in $\mathbf{S}4_m$- or $\mathbf{K}_m$-models. Moreover, due to the symmetry of $\mathbf{S}5_m$-models, which requires reasoning in both directions over the accessibility relations, the PSpace implementation for $\mathcal{ALCS}5_m$ is much more involved and requires backtracking, which is completely unnecessary in the PSpace implementations for both $\mathcal{ALCK}_m$ and $\mathcal{ALCS}4_m$ tableau algorithms (see [6]). For the tableau algorithm, backtracking means that assertions need to be pushed not only down the constraint tree, but also up. The inevitability of backtracking constitutes the main difficulty in a PSpace implementation of the tableau algorithm for $\mathcal{ALCS}5_m$ satisfiability problem and requires special treatment.

This article is structured as follows. Section 2 introduces the syntax and the semantics of $\mathcal{ALCS}5_m$. Section 3 presents issues in designing a tableau algorithm specific to $\mathcal{ALCS}5_m$, motivates the solution, and provides a sound and complete tableau algorithm for the satisfiability problem of $\mathcal{ALCS}5_m$. With the guidance of the tableau algorithm, in Section 4, a PSpace implementation that backtracks using a restart technique is given. Finally, Section 5 concludes the paper.

2 Syntax and Semantics

Let $N_{\mathcal{C}}$, $N_{\mathcal{R}}$, $N_{\mathcal{O}}$ and $N_{\mathcal{E}} = \{1, \ldots, m\}$ denote the pairwise disjoint sets of concept names, role names, individual names and agents, respectively, where $N_{\mathcal{C}}$, $N_{\mathcal{R}}$ and $N_{\mathcal{O}}$ are countably infinite. The set of *concepts* $\mathcal{C}$ and the set of *roles* $\mathcal{R}$ are recursively defined as follows:

$$C, D \longrightarrow A \mid \top \mid \bot \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \forall R.C \mid \exists R.C \mid \Diamond_i C \mid \Box_i C$$
$$R \longrightarrow P \mid \Diamond_i R \mid \Box_i R$$

where $A \in N_{\mathcal{C}}$, $\top$ is the *top symbol*, $\bot$ is the *bottom symbol*, $C, D \in \mathcal{C}$, $P \in N_{\mathcal{R}}$, $R \in \mathcal{R}$ and $i \in N_{\mathcal{E}}$. A concept is said to be of *negation normal form* (NNF) if negation occurs only in front of concept names. It is well-known that any concept can be rewritten into an equivalent negation normal form in linear time [9].

An *assertion* is of the form $C(a)$ or $R(a, b)$ and a *definition* is of the form $A \doteq C$ where $a, b \in N_{\mathcal{O}}$, $C \in \mathcal{C}$, $R \in \mathcal{R}$ and $A \in N_{\mathcal{C}}$. An *ABox* is a finite set of assertions whose concepts and roles belong to the language $\mathcal{ALCS}5_m$. A *TBox* contains a finite set of definitions. Moreover, we only consider *acyclic TBoxes* where no defined concept has more than one definition and no concept name refers to itself via a sequence of TBox definitions. TBoxes containing such definitions are usually referred to as terminologies [2]. An ABox $\mathcal{A}$ and a TBox $\mathcal{T}$ together form an $\mathcal{ALCS}5_m$ knowledge base $\Sigma = \langle \mathcal{A}, \mathcal{T} \rangle$. Given a knowledge base Σ, a concept $A \in N_{\mathcal{C}}$ is said to be *primitive* if it is not defined.

The semantics of $\mathcal{ALCS}5_m$ language is defined by using Kripke structures [10]. A *Kripke structure* for m agents is a tuple $\mathbb{M} = \langle S, \pi, \mathcal{E}_1, ..., \mathcal{E}_m \rangle$ where S is a set of *states*, $\mathcal{E}_i \subseteq S \times S$ is the *accessibility* relation for agent i, and, for each state $s \in S$, the function $\pi(s)$ corresponds to an interpretation of the description logic $\mathcal{ALC}$ that interprets the symbols in $N_{\mathcal{C}}, N_{\mathcal{R}}$, and $N_{\mathcal{O}}$ in a *common non-empty (state-independent) domain* Δ [5,6] as follows: for each

$a \in N_{\mathcal{O}}$, each $A \in N_{\mathcal{C}}$, each $P \in N_{\mathcal{R}}$, $C, D \in \mathcal{C}$, and $R \in \mathcal{R}$,

$$\begin{array}{rlrl}
\top^{\pi(s)} = \Delta, & & (C \sqcup D)^{\pi(s)} & = C^{\pi(s)} \cup D^{\pi(s)}, \\
\bot^{\pi(s)} = \varnothing, & & (C \sqcap D)^{\pi(s)} & = C^{\pi(s)} \cap D^{\pi(s)}, \\
a^{\pi(s)} \in \Delta, & & (\Box_i C)^{\pi(s)} & = \bigcap_{t \in \mathcal{E}_i(s)} C^{\pi(t)}, \\
A^{\pi(s)} \subseteq \Delta, & & (\Diamond_i C)^{\pi(s)} & = \bigcup_{t \in \mathcal{E}_i(s)} C^{\pi(t)}, \\
P^{\pi(s)} \subseteq \Delta \times \Delta, & & (\Box_i R)^{\pi(s)} & = \bigcap_{t \in \mathcal{E}_i(s)} R^{\pi(t)}, \\
(\neg C)^{\pi(s)} = \Delta \setminus C^{\pi(s)}, & & (\Diamond_i R)^{\pi(s)} & = \bigcup_{t \in \mathcal{E}_i(s)} R^{\pi(t)},
\end{array}$$

$$(\forall R.C)^{\pi(s)} = \{a \in \Delta \mid \text{if } (a,b) \in R^{\pi(s)} \text{ for each } b \in \Delta, \text{ then } b \in C^{\pi(s)}\},$$
$$(\exists R.C)^{\pi(s)} = \{a \in \Delta \mid \text{there is } b \in \Delta \text{ such that } (a,b) \in R^{\pi(s)} \text{ and } b \in C^{\pi(s)}\},$$

where $\mathcal{E}_i(s) = \{t \mid (s,t) \in \mathcal{E}_i\}$.

A *(Kripke) world* is a pair $w = (\mathbb{M}, s)$ where $\mathbb{M}$ is a Kripke structure and s is a state in S. Same as in [6], we do not make the Unique Name Assumption (UNA), i.e., we do not require that different names always refer to different entities in a world.

$\mathbb{M}$ is *reflexive* (resp. *transitive*, *symmetric*) if for every $i \in N_{\mathcal{E}}$, the accessibility relation $\mathcal{E}_i$ is reflexive (resp. transitive, symmetric). A Kripke structure $\mathbb{M}$ is an $\mathbf{S}5_m$*-structure* if each $\mathcal{E}_i$ is reflexive, transitive and symmetric, i.e., an equivalence relation.

Definition 1 For any world $(\mathbb{M}, s)$ where $\mathbb{M}$ is an $\mathbf{S}5_m$-structure and s is a state, the satisfiability relation is defined as follows: for each $a, b \in N_{\mathcal{O}}$, each $A \in N_{\mathcal{C}}$, each $R \in \mathcal{R}$, and $C, D \in \mathcal{C}$,

1. $(\mathbb{M}, s) \vDash C(a)$ if $a^{\pi(s)} \in C^{\pi(s)}$,
2. $(\mathbb{M}, s) \vDash R(a,b)$ if $(a^{\pi(s)}, b^{\pi(s)}) \in R^{\pi(s)}$,
3. $(\mathbb{M}, s) \vDash A \doteq C$ if $A^{\pi(s)} = C^{\pi(s)}$.

A knowledge base $\Sigma = \langle \mathcal{A}, \mathcal{T} \rangle$ is $\mathcal{ALCS}5_m$*-satisfiable* if there is a world $w = (\mathbb{M}, s)$ such that w satisfies all the assertions in $\mathcal{A}$ and all the definitions in $\mathcal{T}$ where $\mathbb{M}$ is an $\mathbf{S}5_m$-structure. A definition $A \doteq C$ is *valid in an* $\mathbf{S}5_m$*-structure* $\mathbb{M} = \langle S, \pi, \mathcal{E}_1, ..., \mathcal{E}_m \rangle$, written as $\mathbb{M} \vDash A \doteq C$, if $(\mathbb{M}, s) \vDash A \doteq C$ for every $s \in S$. A TBox $\mathcal{T}$ is *satisfied in an* $\mathbf{S}5_m$*-structure* $\mathbb{M}$, written as $\mathbb{M} \vDash \mathcal{T}$, if $\mathbb{M} \vDash A \doteq C$ for every $A \doteq C \in \mathcal{T}$.

3 Tableau Algorithm

Given an $\mathcal{ALCS}5_m$ knowledge base $\Sigma = \langle \mathcal{A}, \mathcal{T} \rangle$, our goal is to determine whether or not it is $\mathcal{ALCS}5_m$-satisfiable. A *constraint graph* $\mathbb{G} = \langle \mathbb{V}, \mathbb{E}, \mathbb{L} \rangle$ is used to construct such a model, where $\mathbb{V}$ is a set of nodes, $\mathbb{E}$ is a set of directed edges including self-loops, and $\mathbb{L}$ is a function that labels each node $n \in \mathbb{V}$ with a *constraint system* and each edge $(n, n') \in \mathbb{E}$ with a nonempty subset of $N_{\mathcal{E}}$. More specifically, $\mathbb{L}(n, n')$ is a singleton when $n \neq n'$ and equals to the set $N_{\mathcal{E}}$ when $n = n'$. We build the constraint graph starting from a single node labeled with the constraint system obtained from ABox $\mathcal{A}$ and then exhaustively apply the expansion rules. Each assertion $D(a) \in \mathcal{A}$ is rewritten into a constraint $a : D'$ and each $R(a,b) \in \mathcal{A}$ into a constraint $(a,b) : R$ where D' is the NNF of D.

Given a constraint graph $\mathbb{G}$, a node $n \in \mathbb{V}$ is said to be *closed* if constraint system $\mathbb{L}(n)$ contains a *clash*, i.e., $\{a : \bot\} \subseteq \mathbb{L}(n)$. $\mathbb{G}$ is said to be *closed* if at least one of its nodes is closed. A constraint graph that is not closed is *open*, and it is *complete* if no expansion rule applies. We denote the set of all individual names that occur in $\mathbb{G}$ by $\mathcal{O}_{\mathbb{G}}$ (a subset of $N_{\mathcal{O}}$).

To build a model, in the case of $\mathcal{ALCK}_m$, it suffices for a constraint graph to be a tree which we call a *constraint tree*. In the case of $\mathcal{ALCS}4_m$, since accessibility relations of $\mathbf{S}4_m$-models are reflexive and transitive, a constraint tree is built such that the corresponding constraint graph can be constructed by only adding necessary edges to the tree without modifying any constraint system (see Definition 6 in [6]). In both cases, since corresponding constraint systems in the constraint tree and its constraint graph are the same, PSpace implementations of tableau algorithms are achieved by working on the tree and proceeding in a "depth-first" manner: namely, when creating a new constraint system because of a $\Diamond$-constraint in a constraint system $\mathbb{L}(n)$, $\mathbb{L}(n)$ will be explored before another $\Diamond$-constraint in $\mathbb{L}(n)$ is considered. Within each constraint system, after adding constraints about some new individual because of an $\exists$-constraint, all constraints about this individual will be explored before another $\exists$-constraint is considered.

Such interleaving of the expansion of the constraint systems facilitates a PSpace implementation. Tableau algorithms for $\mathcal{ALCK}_m$ and $\mathcal{ALCS}4_m$ were designed to avoid backtracking. Unfortunately, for $\mathcal{ALCS}5_m$, since $\mathbf{S}5_m$-models are symmetric, backtracking, i.e., adding some constraints from a constraint system back to its predecessors, is unavoidable.

To address this problem, the expansion rule that deals with $\Box$-constraints has to be modified to take predecessors into account in the case of $\mathcal{ALCS}5_m$. With this modification, to obtain a PSpace implementation, we use a "restart" technique [11]: if a constraint $x : C$ is added to a constraint system $\mathbb{L}(n)$ because of an application to a $\Box$-constraint, the whole subtree below the node n is discarded and its construction is restarted. Due to the interleaving expansion between $\Diamond$- and $\exists$-constraints, such a restart means that an individual x in a constraint system $\mathbb{L}(n)$ may be created because of an $\exists$-constraint in a subtree that does not currently exist (since the subtree is discarded). To ensure that the reconstruction of the subtree below the node n creates the same individual x for the same $\exists$-constraint, we keep track of the names of nodes on the tree as well as each individual name associated with an $\exists$-constraint within each node by using two one-to-one functions $f_\Diamond$ and $f_\exists$. Let N_Σ be the set of all the symbols appearing in Σ and $\mathcal{O}_\Sigma$ be the set of individual names in Σ (a subset of $N_{\mathcal{O}}$).

- Function $f_\Diamond$ maps each $\Diamond$-constraint in a node to a node that has never been used before. For example, $f_\Diamond((a,b){:}\,\Diamond_i R,\, n) = n'$ means that n' is the name of the node created because of the constraint $(a, b) : \Diamond_i R \in \mathbb{L}(n)$ on the constraint tree.

- Function $f_\exists$ maps each $\exists$-constraint within each node to a distinct individual in $N_\mathcal{O} \setminus \mathcal{O}_\Sigma$, i.e., for each constraint of the form $a : \exists R.C$ in a constraint system $\mathbb{L}(n)$, there is a unique individual $x \in N_\mathcal{O} \setminus \mathcal{O}_\Sigma$ assigned to it. For example, if the $\exists$-rule is applicable to $a : \exists R.\exists R.C \in \mathbb{L}(n)$ for some n and $x_1 = f_\exists(a : \exists R.\exists R.C, n)$, then both $(a, x_1) : R$ and $x_1 : \exists R.C$ will be added into $\mathbb{L}(n)$. Moreover, if $x_2 = f_\exists(x_1 : \exists R.C, n)$, then $(x_1, x_2) : R$ and $x_2 : C$ will be added into $\mathbb{L}(n)$ too.

In a constraint graph, a node n is said to be an *i-neighbor* of another node n' if $i \in \mathbb{L}(n, n') \cup \mathbb{L}(n', n)$. Expansion rules are listed in Figure 1. There are three kinds of expansion rules: *local expansion rules* (L-rules) generate constraints within a constraint system, *global expansion rules* (G-rules) add constraints to the constraint systems associated with nodes that are accessible from the current node, and *terminological expansion rules* (T-rules) add constraints to a constraint system based on both the constraints within the constraint systems and the TBox $\mathcal{T}$.

Note that T-rules always expand the constraint system from the left-hand side of a definition to the right-hand side. For example, if $a : \neg A \in \mathbb{L}(n), A \doteq \Diamond_1 C \in \mathcal{T}$, and n has a 1-neighbor n' with $a : C \in \mathbb{L}(n')$, following our left-to-right approach, instead of adding $a : A$, we add $a : \Box_1 \neg C$ to $\mathbb{L}(n)$ and the detection of the clash is relegated to the constraint system $\mathbb{L}(n')$. This strategy allows detection of potential clashes without fully expanding constraint systems. Also note that to build a model for the knowledge base containing an assertion of the form $\exists R.C(a)$ which implies that an R-successor of individual a belongs to C without specifying the individual, under the open world assumption (OWA) in the absence of UNA, it is sufficient to use an individual name that has not yet been used in the constraint graph to denote this unknown individual.

We denote by Λ the *tableau algorithm* which nondeterministically applies the L, G, and T-rules until no further application is possible. In general, algorithm Λ produces a constraint graph. Initially, the constraint graph $\mathbb{G}$ consists of a single constraint system that contains only the individual names occurring in Σ, i.e., $\mathcal{O}_\mathbb{G} = \mathcal{O}_\Sigma$. With applications of expansion rules, new constraint systems may be added. The function $f_\Diamond$ in the $\Diamond$-rule maps each $\Diamond$-constraint in a constraint system to a unique node with which a new constraint system is associated. Within each constraint system, new individual names may be added to $\mathcal{O}_\mathbb{G}$. The function $f_\exists$ in the $\exists$-rule maps each $\exists$-constraint to a unique individual in $N_\mathcal{O} \setminus \mathcal{O}_\Sigma$.

In what follows, we show that the tableau algorithm Λ terminates.

Lemma 1 *All executions of Λ on an $\mathcal{ALCS}5_m$ knowledge base terminate.*

Proof It suffices to show that the constraint graph that Λ creates contains finitely many nodes and that each constraint system contains finitely many constraints.

***L*-Rules:**

$\sqcap$-rule If there is a node n with $a : C_1 \sqcap C_2 \in \mathbb{L}(n)$ and $\{a : C_1, a : C_2\} \not\subseteq \mathbb{L}(n)$,
then $\mathbb{L}(n) := \mathbb{L}(n) \cup \{a : C_1, a : C_2\}$;

$\sqcup$-rule If there is a node n with $a : C_1 \sqcup C_2 \in \mathbb{L}(n)$ and $\{a : C_1, a : C_2\} \cap \mathbb{L}(n) = \varnothing$,
then $\mathbb{L}(n) := \mathbb{L}(n) \cup \{a : C_i\}$ with $i = 1$ or $i = 2$;

$\exists$-rule If there is a node n with $a : \exists R.C \in \mathbb{L}(n)$, no L-, T- or $\Box$-rule except $\exists$-rule is applicable, and there is no $b \in \mathcal{O}_{\mathbb{G}}$ such that $\{(a, b) : R, b : C\} \subseteq \mathbb{L}(n)$,
then $\mathbb{L}(n) := \mathbb{L}(n) \cup \{(a, c) : R, c : C\}$ where $c = f_{\exists}(a : \exists R.C, n)$;

$\forall$-rule If there is a node n such that $\{a : \forall R.C, (a, b) : R\} \subseteq \mathbb{L}(n)$ and $b : C \notin \mathbb{L}(n)$,
then $\mathbb{L}(n) := \mathbb{L}(n) \cup \{b : C\}$;

$\bot$-rule If there is a node n such that $\{a : C, a : \neg C\} \subseteq \mathbb{L}(n)$ and $a : \bot \notin \mathbb{L}(n)$,
then $\mathbb{L}(n) := \mathbb{L}(n) \cup \{a : \bot\}$;

***G*-Rules:**

$\Diamond$-rule If there is a node n with
- *(i)* $a : \Diamond_i C \in \mathbb{L}(n)$ and $a : C \notin \mathbb{L}(n)$, no L-, T-, or $\Box$-rule is applicable, and n has no i-neighbor l such that $a : C \in \mathbb{L}(l)$, then add a new i-neighbor n' of n with $\mathbb{L}(n') := \{a : C\}$ where $n' = f_{\Diamond}(a : \Diamond_i C, n)$, and for each i-neighbor n'' of n, let $\mathbb{L}(n'', n') := \{i\}$ and $\mathbb{L}(n', n'') := \{i\}$;
- *(ii)* $(a, b) : \Diamond_i R \in \mathbb{L}(n)$ and $(a, b) : R \notin \mathbb{L}(n)$, no L-, T-, or $\Box$-rule is applicable, and n has no i-neighbor l such that $(a, b) : R \in \mathbb{L}(l)$, then add a new i-neighbor n' of n with $\mathbb{L}(n') := \{(a, b) : R\}$ where $n' = f_{\Diamond}((a, b) : \Diamond_i R, n)$, and for each i-neighbor n'' of n, let $\mathbb{L}(n'', n') := \{i\}$ and $\mathbb{L}(n', n'') := \{i\}$;

$\Box$-rule If there is a node n with
- *(i)* $a : \Box_i C \in \mathbb{L}(n)$, no L- or T-rule is applicable and n has an i-neighbor n' such that $a : C \notin \mathbb{L}(n')$, then $\mathbb{L}(n') := \mathbb{L}(n') \cup \{a : C\}$;
- *(ii)* $(a, b) : \Box_i R \in \mathbb{L}(n)$, no L- or T-rule is applicable and n has an i-neighbor n' such that $(a, b) : R \notin \mathbb{L}(n')$, then $\mathbb{L}(n') := \mathbb{L}(n') \cup \{(a, b) : R\}$;

***T*-Rules:**

T-rule If there is a node n with $a : A \in \mathbb{L}(n), A \doteq D \in \mathcal{T}$, and $a : D \notin \mathbb{L}(n)$,
then $\mathbb{L}(n) := \mathbb{L}(n) \cup \{a : D'\}$ where D' is the NNF of D;

N-rule If there is a node n with $a : \neg A \in \mathbb{L}(n), A \doteq D \in \mathcal{T}$, and $a : D' \notin \mathbb{L}(n)$,
then $\mathbb{L}(n) := \mathbb{L}(n) \cup \{a : D'\}$ where D' is the NNF of $\neg D$.

Fig. 1 The Tableau Expansion Rules

Within each constraint system, local expansion rules only create constraints that are subexpressions of the original constraints or $\bot$. For each constraint, since the TBox is acyclic, an expansion rule can only be applied at most once. Therefore, each constraint system has finitely many constraints. It then follows from the applicability of global expansion rules that the outdegree of each node in the constraint graph is finite.

Since there are finitely many nodes in the constraint graph and each constraint system in the graph is also finite, tableau algorithm Λ terminates. $\Box$

The next definition provides a formal interpretation of a constraint graph $\mathbb{G}$. The idea is that each constraint system is mapped to a state of $\mathbb{M}$ in which all its assertions are satisfied and each labeled edge in $\mathbb{G}$ is mapped to the corresponding accessibility relation.

Definition 2 Let $\mathbb{G} = \langle \mathbb{V}, \mathbb{E}, \mathbb{L} \rangle$ be a constraint graph, $\mathbb{M} = \langle S, \pi, \mathcal{E}_1, ..., \mathcal{E}_m \rangle$ a Kripke structure, and σ a mapping from $\mathbb{V}$ to S. Then $\mathbb{M}$ *satisfies* $\mathbb{G}$ *via* σ

if, for all $n, n' \in \mathbb{V}$,

(1) $i \in \mathbb{L}(n, n') \Longrightarrow (\sigma(n), \sigma(n')) \in \mathcal{E}_i$

(2) $a : C \in \mathbb{L}(n) \Longrightarrow (\mathbb{M}, \sigma(n)) \models C(a)$

(3) $(a, b) : R \in \mathbb{L}(n) \Longrightarrow (\mathbb{M}, \sigma(n)) \models R(a, b)$

$\mathbb{M}$ *satisfies* $\mathbb{G}$, denoted by $\mathbb{M} \Vdash \mathbb{G}$, if there is a mapping σ such that $\mathbb{M}$ satisfies $\mathbb{G}$ via σ. In this case we also say that $\mathbb{M}$ is a *model of* $\mathbb{G}$. Note that if $\mathbb{M} \Vdash \mathbb{G}$, then $\mathbb{G}$ is open.

Given an interpretation π and a finite set of symbols $N \subseteq N_{\mathcal{C}} \cup N_{\mathcal{R}} \cup N_{\mathcal{O}}$, we define π *restricted to* N, denoted by $\pi|_N$, to be the restriction of function π to N. Let $\mathbb{M} = \langle S, \pi, \mathcal{E}_1, ..., \mathcal{E}_m \rangle$ and $\mathbb{M}' = \langle S, \pi', \mathcal{E}_1, ..., \mathcal{E}_m \rangle$ be two Kripke structures, and $N_{\mathcal{C}} \cup N_{\mathcal{R}} \subseteq N_2 \subseteq N_1$ be finite subsets of $N_{\mathcal{C}} \cup N_{\mathcal{R}} \cup N_{\mathcal{O}}$. Then $\mathbb{M}'$ is a *semantic extension* of $\mathbb{M}$ with respect to N_2 if $(\pi'|_{N_1})|_{N_2} = \pi|_{N_2}$. Note that $\mathbb{M}$ is a semantic extension of itself. The following theorem shows the soundness of the expansion rules and its proof is given in Appendix A.

Theorem 1 *Let* $\mathbb{M} = \langle S, \pi, \mathcal{E}_1, ..., \mathcal{E}_m \rangle$ *be an* $\mathbf{S5}_m$*-structure and* $\mathcal{T}$ *an acyclic TBox such that* $\mathbb{M} \models \mathcal{T}$. *Let* $\mathbb{G}$ *be a constraint graph obtained using* $\mathcal{T}$, α *an L-, G-, or T-rule and* $\mathbb{G}_\alpha$ *a constraint graph obtained by applying* α *to* $\mathbb{G}$. *If* $\mathbb{M} \Vdash \mathbb{G}$ *via* σ, *then there exists a semantic extension* $\mathbb{M}_\alpha$ *of* $\mathbb{M}$ *with respect to* $N_\Sigma \cup \mathcal{O}_{\mathbb{G}}$ *such that* $\mathbb{M}_\alpha \Vdash \mathbb{G}_\alpha$ *via* σ' *(extension of* σ*) and* $\mathbb{M}_\alpha \models \mathcal{T}$. *Furthermore,* $\mathbb{M}_\alpha \Vdash \mathbb{G}$.

The *canonical interpretation* defined below is used to build a model for a constraint graph:

Definition 3 Let $\mathbb{G} = \langle \mathbb{V}, \mathbb{E}, \mathbb{L} \rangle$ be a constraint graph, $\mathcal{T}$ a simple acyclic TBox. The *canonical Kripke structure* $\mathbb{M}_{\mathbb{G}} = \langle S, \pi, \mathcal{E}_1, ..., \mathcal{E}_m \rangle$ for $\mathbb{G}$ with respect to $\mathcal{T}$ is defined as follows:

- $S := \mathbb{V}$,
- $\mathcal{E}_i := \{e \in \mathbb{E} \mid i \in \mathbb{L}(e)\}$, for each $1 \leq i \leq m$,
- $\Delta := \mathcal{O}_{\mathbb{G}}$,
- for every $n \in \mathbb{V}$,
  - $a^{\pi(n)} := a$, for every $a \in \mathcal{O}_{\mathbb{G}}$,
  - $P^{\pi(n)} := \{(a, b) \mid (a, b) : P \in \mathbb{L}(n)\}$, if $P \in N_{\mathcal{R}}$,
  - $A^{\pi(n)} := \{a \mid a : A \in \mathbb{L}(n)\}$, if A is primitive,
  - $A^{\pi(n)} := \{a \mid a : A \in \mathbb{L}(n)\} \cup D^{\pi(n)}$, if $A \doteq D \in \mathcal{T}$.

Note that, due to the acyclicity of the TBox $\mathcal{T}$, the interpretation of concept names is well defined in the canonical Kripke structure $\mathbb{M}_{\mathbb{G}}$.

Lemma 2 *Let* $\mathcal{T}$ *be a simple acyclic TBox and* $\mathbb{G}$ *an open complete constraint graph with respect to local, global and terminological expansion rules. Then the canonical Kripke structure* $\mathbb{M}_{\mathbb{G}}$ *for* $\mathbb{G}$ *with respect to* $\mathcal{T}$ *is an* $\mathbf{S5}_m$*-structure.*

Proof It suffices to show that for each $1 \leq i \leq m$, relation $\mathcal{E}_i$ is an equivalence relation. Indeed, $\mathcal{E}_i$ is reflexive by the definition of function $\mathbb{L}$. Since $\mathbb{G}$ is open and complete, it follows from the definition of i-neighbor and the $\Diamond$-rule during the construction of $\mathbb{G}$ that $\mathcal{E}_i$ is transitive and symmetric. $\square$

Based on Definition 3, we can build the canonical Kripke structure $\mathbb{M}_{\mathbb{G}}$ of the constraint graph $\mathbb{G}$. The rest of this section is dedicated to proving the soundness and completeness of tableau algorithm Λ. That is, we show that $\mathbb{G}$ is open and complete if and only if it has the canonical Kripke structure $\mathbb{M}_{\mathbb{G}}$. We first present two auxiliary lemmas that are involved.

Lemma 3 *Let $\mathbb{G}$ be an open complete constraint graph with respect to local, global and terminological expansion rules. Then for every $R \in \mathcal{R}$ and every $a, b \in \Delta$, if $(a,b) : R \in \mathbb{L}(n)$, then $(\mathbb{M}_{\mathbb{G}}, n) \models R(a,b)$.*

Proof We prove the lemma by induction on the structure of R. The base case is when $R \in N_{\mathcal{R}}$. By Definition 3 and Definition 1, if $(a,b) : R \in \mathbb{L}(n)$, then $(\mathbb{M}_{\mathbb{G}}, n) \models R(a,b)$ for each $R \in N_{\mathcal{R}}$. For the induction step, there are two cases:

- R is of the form $\Diamond_i Q$. Since $\mathbb{G}$ is complete, there exists $n' \in \mathbb{V}$ such that $i \in \mathbb{L}(n,n')$ and $(a,b) : Q \in \mathbb{L}(n')$. Since $i \in \mathbb{L}(n,n')$ implies $\mathcal{E}_i(n,n')$ by Definition 3 and $(a,b) : Q \in \mathbb{L}(n')$ implies $(\mathbb{M}_{\mathbb{G}}, n') \models Q(a,b)$ by the induction hypothesis, we have $(\mathbb{M}_{\mathbb{G}}, n) \models \Diamond_i Q(a,b)$.
- R is of the form $\Box_i Q$. Since $\mathbb{G}$ is complete, then for every $n' \in \mathbb{V}$ where $i \in \mathbb{L}(n,n')$, we have $(a,b) : Q \in \mathbb{L}(n')$. Since $i \in \mathbb{L}(n,n')$ implies $\mathcal{E}_i(n,n')$ by Definition 3 and $(a,b) : Q \in \mathbb{L}(n')$ implies $(\mathbb{M}_{\mathbb{G}}, n') \models Q(a,b)$ by the induction hypothesis, we have $(\mathbb{M}_{\mathbb{G}}, n) \models \Box_i Q(a,b)$.

□

The following auxiliary lemma specifically deals with negation. Its proof is given in Appendix B.

Lemma 4 *Let $\mathcal{T}$ be an acyclic TBox and $\mathbb{G}$ an open complete constraint graph with respect to local, global and terminological expansion rules. Then for every $A \in N_{\mathcal{C}}$ and every $a \in \Delta$, if $a : \neg A \in \mathbb{L}(n)$, then $(\mathbb{M}_{\mathbb{G}}, n) \models \neg A(a)$.*

Theorem 2 *Let $\Sigma = \langle \mathcal{A}, \mathcal{T} \rangle$ be an $\mathcal{ALCS}5_m$ knowledge base where $\mathcal{T}$ is a simple acyclic TBox, and let $\mathbb{G}$ be a constraint graph that is initialized as a single node labeled with the constraint system obtained from $\mathcal{A}$ and complete with respect to L-, G-, and T- rules. Then $\mathbb{G}$ is open if and only if $\mathbb{M}_{\mathbb{G}} \Vdash \mathbb{G}$ and $\mathbb{M}_{\mathbb{G}} \models \mathcal{T}$.*

Proof ($\Leftarrow$) If $\mathbb{G}$ is closed, then there does not exist a model that satisfies $\mathbb{G}$. This contradicts the assumption $\mathbb{M}_{\mathbb{G}} \Vdash \mathbb{G}$.

($\Rightarrow$) Suppose that $\mathbb{G}$ is open and complete with respect to L-, G-, and T-rules. By Definition 3, for every $n, n' \in \mathbb{V}$, if $i \in \mathbb{L}(n,n')$, then $\mathcal{E}_i(n,n')$. Thus, by Definition 2 and Lemma 3, it suffices to show that $a : C \in \mathbb{L}(n)$ implies $(\mathbb{M}_{\mathbb{G}}, n) \models C(a)$ for each $C \in \mathcal{C}$.

We prove the statement by induction on the structures of C. The base case is when $C \in N_{\mathcal{C}}$. If C is primitive, then $(\mathbb{M}_{\mathbb{G}}, n) \models C(a)$ by Definition 3 and Definition 1. If C is not primitive, then there is a definition $C \doteq D \in \mathcal{T}$, and by Definition 3, $C^{\pi(n)} = \{b \mid b : C \in \mathbb{L}(n)\} \cup D^{\pi(n)}$. Hence, $(\mathbb{M}_{\mathbb{G}}, n) \models C(a)$ by Definition 1.

With respect to the induction step, the most involved case is that of the negation, which was dealt with in Lemma 4. The remaining cases, namely, $\sqcap, \sqcup, \exists, \forall, \Diamond$, and $\Box$, are proved below.

1. C is of the form $B_1 \sqcap B_2$. Since $\mathbb{G}$ is complete, $\{a : B_1, a : B_2\} \subseteq \mathbb{L}(n)$. By the induction hypothesis, $(\mathbb{M}_{\mathbb{G}}, n) \models B_1(a)$ and $(\mathbb{M}_{\mathbb{G}}, n) \models B_2(a)$. Thus, $(\mathbb{M}_{\mathbb{G}}, n) \models B_1 \sqcap B_2(a)$. Therefore, $(\mathbb{M}_{\mathbb{G}}, n) \models C(a)$.
2. C is of the form $B_1 \sqcup B_2$. Since $\mathbb{G}$ is complete, $\{a : B_1, a : B_2\} \cap \mathbb{L}(n) \neq \varnothing$. By the induction hypothesis, $(\mathbb{M}_{\mathbb{G}}, n) \models B_1(a)$ or $(\mathbb{M}_{\mathbb{G}}, n) \models B_2(a)$. Thus, $(\mathbb{M}_{\mathbb{G}}, n) \models B_1 \sqcup B_2(a)$. Therefore, $(\mathbb{M}_{\mathbb{G}}, n) \models C(a)$.
3. C is of the form $\exists R.B$. Since $\mathbb{G}$ is complete, there exists b such that $\{(a,b) : R, b : B\} \subseteq \mathbb{L}(n)$. It follows from Lemma 3 and $(a,b) : R \in \mathbb{L}(n)$ that $(\mathbb{M}_{\mathbb{G}}, n) \models R(a,b)$. Furthermore, $(\mathbb{M}_{\mathbb{G}}, n) \models B(b)$ by the induction hypothesis. Therefore, $(\mathbb{M}_{\mathbb{G}}, n) \models \exists R.B(a)$.
4. C is of the form $\forall R.B$. Since $\mathbb{G}$ is complete, for every b where $(a,b) : R \in \mathbb{L}(n)$, we have $b : B \in \mathbb{L}(n)$. By Lemma 3, $(a,b) : R \in \mathbb{L}(n)$ implies $(\mathbb{M}_{\mathbb{G}}, n) \models R(a,b)$. By the induction hypothesis, $b : B \in \mathbb{L}(n)$ implies $(\mathbb{M}_{\mathbb{G}}, n) \models B(b)$. Therefore, $(\mathbb{M}_{\mathbb{G}}, n) \models \forall R.B(a)$.
5. C is of the form $\Diamond_i B$. Since $\mathbb{G}$ is complete, there exists $n' \in \mathbb{V}$ such that $i \in \mathbb{L}(n, n')$ and $a : B \in \mathbb{L}(n')$. By the induction hypothesis, $(\mathbb{M}_{\mathbb{G}}, n') \models B(a)$. Since $i \in \mathbb{L}(n, n')$ implies $\mathcal{E}_i(n, n')$, it follows that $(\mathbb{M}_{\mathbb{G}}, n) \models \Diamond_i B(a)$.
6. C is of the form $\Box_i B$. Since $\mathbb{G}$ is complete, then for every $n' \in \mathbb{V}$ where $i \in \mathbb{L}(n, n')$, we have $a : B \in \mathbb{L}(n')$. Since $i \in \mathbb{L}(n, n')$ implies $\mathcal{E}_i(n, n')$ and, by the induction hypothesis, $a : B \in \mathbb{L}(n')$ implies $(\mathbb{M}_{\mathbb{G}}, n') \models B(a)$, we have $(\mathbb{M}_{\mathbb{G}}, n) \models \Box_i B(a)$.

We next show that $\mathcal{T}$ is valid in $\mathbb{M}_{\mathbb{G}}$. Suppose that there is a node n and a definition $A \doteq D \in \mathcal{T}$ such that $(\mathbb{M}_{\mathbb{G}}, n) \not\models A \doteq D$. Since A is not primitive, $A^{\pi(n)} := \{a \mid a : A \in \mathbb{L}(n)\} \cup D^{\pi(n)}$. Hence, $D^{\pi(n)} \subseteq A^{\pi(n)}$. Suppose that $D^{\pi(n)} \neq A^{\pi(n)}$. Then there is $b \in \mathcal{O}_{\mathbb{G}}$ such that $b \in A^{\pi(n)}$ and $b \notin D^{\pi(n)}$. This implies that $b \in \{a \mid a : A \in \mathbb{L}(n)\}$. $\mathbb{G}$ being complete and $b : A \in \mathbb{L}(n)$ imply that $b : D \in \mathbb{L}(n)$. We already showed that $\mathbb{M}_{\mathbb{G}} \Vdash \mathbb{G}$. Thus, $(\mathbb{M}_{\mathbb{G}}, n) \models D(b)$ if and only if $b \in D^{\pi(n)}$, which leads to a contradiction. It follows that for every definition $A \doteq D \in \mathcal{T}$ and for every $n \in \mathbb{V}$, we have $(\mathbb{M}_{\mathbb{G}}, n) \models A \doteq D$. $\Box$

The next lemma helps design a PSPACE implementation of tableau algorithm Λ.

Lemma 5 *Let $\mathbb{G} = \langle \mathbb{V}, \mathbb{E}, \mathbb{L} \rangle$ be an open and complete constraint graph and $\mathbb{M}_{\mathbb{G}}$ the canonical Kripke structure for $\mathbb{G}$. For every $n \in \mathbb{V}$, if $a : \Box_i C \in \mathbb{L}(n)$, then for every i-neighbor $n' \in \mathbb{V}$ of n, we have $(\mathbb{M}_{\mathbb{G}}, n') \models \Box_i C(a)$.*

Proof Suppose that $a : \Box_i C \in \mathbb{L}(n)$. Consider any i-neighbor $n' \in \mathbb{V}$ of n. We need to show that $(\mathbb{M}_{\mathbb{G}}, n') \models \Box_i C(a)$. Indeed, since $\mathbb{G}$ is open and complete, for each i-neighbor $v \in \mathbb{V}$ of n, we have $a : C \in \mathbb{L}(v)$ and that v is also an i-neighbor of n'. Thus, $(\mathbb{M}_{\mathbb{G}}, v) \models C(a)$ by Theorem 2 and Definition 2. Therefore, $(\mathbb{M}_{\mathbb{G}}, n') \models \Box_i C(a)$. $\Box$

4 A PSpace implementation

In Section 3, our tableau algorithm Λ constructs a constraint graph. To have a PSpace implementation of Λ, we need to construct a tree. The idea is to proceed in a "depth-first" manner and only store the path between the root of the tree and the current node (as well as the corresponding constraint systems). By Lemma 5, all i-neighbors have the same set of $\Box_i$-constraints. However, during the construction, some i-neighbors may not be connected by an edge directly. Therefore, we further extend the concept of i-neighbors to *i-ancestors* and *i-descendants.*

In a constraint tree, if $i \in \mathbb{L}(n, n')$, then n' is an *i-successor* of n and n is an *i-predecessor* of n', i.e., n' is directly accessible from node n for agent i. If $i \in \bigcap_{j=1}^{k-1} \mathbb{L}(n_j, n_{j+1})$ where $k > 1$, then we say that n_1 is an *i-ancestor* of n_k and that n_k is an *i-descendant* or n_1. Note that a node n is an i-neighbor of another node n' if n is an i-ancestor or an i-descendant of n'.

As in the case of $\mathcal{ALCK}_m$ and $\mathcal{ALCS}5_m$, the constraint tree is expanded with two kinds of successors created by the expansion rules, successors of individuals (with respect to roles) created because of the $\exists$-rule, and successors of a constraint system created because of the $\Diamond$-rule. Within each constraint system $\mathbb{L}(n)$, without carefully applying the tableau rules, the number of the constraints generated may be exponential in the size of input. The set of constraints $a : C_i$, where $C_i = \exists R.C_{i1} \sqcap \exists R.C_{i2} \sqcap \forall R.C_{i+1}$ and $C_n = \exists R.C_{n1} \sqcap \exists R.C_{n2}$, provides such an example. Therefore, whenever the $\exists$-rule is applied to a constraint $a : \exists R.D \in \mathbb{L}(n)$, an R-successor of a, say x, is created and an auxiliary constraint system $\mathbb{L}^x(n)$ is initialized as $\{(a, x) : R,\, x : D\}$. Subsequently, constraints involving x are put in $\mathbb{L}^x(n)$. When the $\exists$-rule is applied to another $\exists$-constraint, the space used for $\mathbb{L}^x(n)$ will be reused. Note that since $\mathbb{L}^x(n)$ will be interpreted in the same world as $\mathbb{L}(n)$, it is annotated with the same node n. Similarly, for any $\Diamond$-constraint in $\mathbb{L}(n)$, an application of the $\Diamond$-rule creates a new node with a new constraint system. That space will be reused for another constraint system created because of another $\Diamond$-constraint in $\mathbb{L}(n)$. However, since the newly created constraint system will be interpreted in a different world, it is denoted by $\mathbb{L}(n')$ with a different node name n'.

The algorithm $\mathcal{ALCS}5_m$-Sat (Algorithm 1) is an implementation of the tableau algorithm Λ. Given an $\mathcal{ALCS}5_m$ knowledge base $\Sigma = \langle \mathcal{A}, \mathcal{T} \rangle$, the algorithm calls the recursive function Sat(n, $\mathbb{L}(n)$, Aux) to decide whether or not Σ is $\mathbf{S}5_m$-satisfiable. The main idea behind Sat(n, $\mathbb{L}(n)$, Aux) is to restrict the order in which expansion rules are applied so as to maintain a single path $\mathbb{P}$ of the constraint tree at all times during the execution of the algorithm.

As mentioned before, backtracking is unavoidable (see the $\Box$-rule) and so the "restart" technique [11] is employed: if $x : \Box_i C \in \mathbb{L}(n)$ and $x : C \notin \mathbb{L}(n')$ where n' is an i-ancestor of n, then $x : C$ will be added to $\mathbb{L}(n')$, the whole subtree below the node n' is discarded, and the construction is restarted from node n'. However, with such an approach, constraint $x : C$ cannot be easily passed down to the nodes which are both i-descendants of n' and i-ancestors of n. To ensure that constraint $x : C$ is also added to these nodes, by Lemma 5, instead of adding $x : C$ when backtracking, we add constraint $x : \Box_i C$. This is implemented using a set $\mathbb{B}(n)$ that contains all the $\Box$-constraints in the constraint system associated with node n, including the ones in the auxiliary constraint system. That is, $\mathbb{B}_i(n) = \{a : \Box_i C \in \mathbb{L}(n),\ (a, b) : \Box_i R \in \mathbb{L}(n) \mid i \in N_{\mathcal{E}}\}$ and $\mathbb{B}(n) = \bigcup_i \mathbb{B}_i(n)$.

The individual x may have been created because of some $\exists$-constraint in $\mathbb{L}(n')$ where n' is an i-descendant of n. The function $f_\exists$ that maps each $\exists$-constraint in a constraint system to a distinct individual in $N_{\mathcal{O}} \setminus \mathcal{O}_\Sigma$ guarantees that the reconstruction of subtree below the node n uses the same individual x (as a witness) for the same $\exists$-constraint. As an example, suppose that constraint system $\mathbb{L}(n)$ contains a constraint $a : \Box_1 \exists R.(A \sqcap \Diamond_1 \neg A)$. By a sequence of applications of the $\Box$-, $\exists$-, and $\sqcap$-rules (in order), $\{a : \exists R.(A \sqcap \Diamond_1 \neg A),\ (a, x) : R,\ x : A,\ x : \Diamond_1 \neg A\} \subseteq \mathbb{L}(n)$ where x is a new individual created by the $\exists$-rule. Because of the constraint $x : \Diamond_1 \neg A$, an application of the $\Diamond$-rule creates a new node, say n', with which the constraint system $\mathbb{L}(n')$ is associated. By G-rules, $\{a : \exists R.(A \sqcap \Diamond_1 \neg A),\ x : \neg A\} \subseteq \mathbb{L}(n')$. Note that individual x was created because of constraint $a : \exists R.(A \sqcap \Diamond_1 \neg A) \in \mathbb{L}(n)$. If the same individual x were used for the same constraint in $\mathbb{L}(n')$, an undesired clash would occur.

For subroutines of Algorithm 1 (Functions 2 and 3) to interact with constraint systems on path $\mathbb{P}$ easily, $\mathbb{P}$ is maintained as a global variable which will be changed along with the execution. For every node n on $\mathbb{P}$ during the execution, $\mathbb{B}(n)$ is kept globally and used by $\mathbb{L}(n)$ and all the auxiliary constraint systems associated with the node n. If a node n is removed from $\mathbb{P}$, so is $\mathbb{B}(n)$. Note that, however, if an auxiliary constraint system is removed, set $\mathbb{B}(n)$ should be kept for $\mathbb{L}(n)$ (and other potential auxiliary constraint systems) to use.

The function $\textsc{Sat}(n, \mathbb{L}(n), Aux)$ has three parameters. The first parameter n denotes the node on path $\mathbb{P}$ at which the current function is working. The second parameter is a constraint system associated with the current node. It can be an auxiliary constraint system created by an application of the $\exists$-rule. Whether it is an auxiliary constraint system or not is indicated by the third parameter Aux. For any node n, since the set $\mathbb{B}(n)$ is shared by $\mathbb{L}(n)$ and all the auxiliary constraint systems associated with the node n, it can only be removed when a node n is removed from $\mathbb{P}$. The parameter Aux is used to check if $\mathbb{B}(n)$ and the node n need to be removed whenever a constraint system is to be discarded (See function $\textsc{Sat}$ lines 13-15 and 26-28 in Algorithm 1).

There are three kinds of return values of the function $\textsc{Sat}$: “satisfiable”, “not satisfiable” and a node name. A node name is only returned because of backtracking (see Line 16 of Function $\textsc{Sat}$). The outside while loop of function $\textsc{Sat}(n, \mathbb{L}(n), Aux)$ in Algorithm 1 is used to restart the construction from the current constraint system. Line 2 ensures that when restarted, the current constraint system contains all the $\Box$-constraints, including the ones added by some previous descendant that has been discarded.

Algorithm 1 $\mathcal{ALCS}5_m$-Sat(Σ)

Global variables: $\mathbb{P}$, $\mathbb{B}(n)$ for each node n in $\mathbb{P}$

$\mathcal{ALCS}5_m$-Sat(Σ) := Sat($n_0, \mathbb{L}(n_0), False$), where $\Sigma = \langle \mathcal{A}, \mathcal{T} \rangle$, $\mathbb{L}(n_0)$ is obtained from $\mathcal{A}$, and $False$ implies that $\mathbb{L}(n_0)$ is not an auxiliary constraint system.

$\mathbb{P}$ is initialized to be a constraint tree containing only node n_0.
$\mathbb{B}(n_0)$ is initialized to be $\{a : \Box_k C \in \mathbb{L}(n_0), (a,b) : \Box_k R \in \mathbb{L}(n_0) \mid k \in N_{\mathcal{E}}\}$.

Sat($n, \mathbb{L}(n), Aux$):

```
1: while true do
2:    $\mathbb{L}(n) := \mathbb{L}(n) \cup \mathbb{B}(n)$
3:    while an $L$- or $T$-rule, except for the $\exists$-rule, is applicable to $\mathbb{L}(n)$ do
4:       apply the rule (if it is a $\sqcup$-rule, nondeterministically pick one choice), add the new
         constraints to $\mathbb{L}(n)$, and update $\mathbb{B}(n)$ accordingly
5:    end while
6:    if $\mathbb{L}(n)$ contains a clash then
7:       return "not satisfiable"
8:    end if
9:    if there is an $i$-ancestor $n'$ of $n$ in $\mathbb{G}$ s.t. $\mathbb{B}_i(n) \not\subseteq \mathbb{B}_i(n')$ {/* backtrack */} then
10:      let $n''$ be the $i$-ancestor of $n$ on $\mathbb{P}$ that is closest to the root of $\mathbb{P}$ s.t. $\mathbb{B}_i(n) \not\subseteq \mathbb{B}_i(n'')$
11:      $\mathbb{B}(n'') := \mathbb{B}(n'') \cup \mathbb{B}_i(n)$
12:      discard $\mathbb{L}(n)$
13:      if $Aux = False$ then
14:         discard $\mathbb{B}(n)$; remove $n$ and the corresponding edge from $\mathbb{P}$
15:      end if
16:      return $n''$
17:   end if
18:   result := EBranch($n, \mathbb{L}(n)$)
19:   if result = "satisfiable" then
20:      result := DBranch($n, \mathbb{L}(n)$)
21:   end if
22:   if result = $n$ and $Aux = False$ then
23:      continue {/* restart, i.e., go to Line 1 */}
24:   else
25:      discard $\mathbb{L}(n)$
26:      if $Aux = False$ then
27:         discard $\mathbb{B}(n)$; remove $n$ and the corresponding edge from $\mathbb{P}$
28:      end if
29:      return result
30:   end if
31: end while
```

Within each constraint system, before applying the $\exists$-rule or a global rule, the algorithm ensures that all the other local and terminological rules are applied and checks whether there is a clash (Lines 3-8). The algorithm then checks whether the $\Box$-rule is applicable (Line 9). Since at this point the current constraint system has no successor, if the $\Box$-rule is applicable to a constraint $a : \Box_i C \in \mathbb{L}(n)$, it will add $a : C$ to some constraint system associated with some i-ancestor n' of n by adding $a : \Box_i C$ to $\mathbb{L}(n')$ (see Lemma 5), which, in turn, is implemented by adding all $\Box$-constraints in $\mathbb{B}(n)$ to $\mathbb{B}(n')$. We choose the farthest i-ancestor n'' to which such a $\Box_i$-constraint can be added. Then we recursively remove the whole subtree of this ancestor n'' (including the nodes, edges and associated constraint systems). If no backtrack is needed, the

Function 2 EBRANCH$(n, \mathbb{L}(n))$

```
1:  $\mathbf{E}(n) := \{a : \exists R.C \in \mathbb{L}(n) \mid$ there is no $b$ such that $(a,b) : R, b : C \in \mathbb{L}(n)\}$
2:  while $\mathbf{E}(n) \neq \varnothing$ do
3:     pick one $a : \exists R.C \in \mathbf{E}(n)$ and let $\mathbb{L}^x(n) := \{(a,x) : R, x : C\} \cup \{x : D \mid x : D \in \mathbb{B}(n)\}$
       where $x = f_\exists(a : \exists R.C, n)$
4:     while there is $a : \forall R.E \in \mathbb{L}(n) \cup \mathbb{L}^x(n)$ and $x : E \notin \mathbb{L}^x(n)$ do
5:        add $x : E$ to $\mathbb{L}^x(n)$ and update $\mathbb{B}(n)$ accordingly
6:     end while
7:     result := SAT$(n, \mathbb{L}^x(n), True)$
8:     if result $\neq$ "satisfiable" then
9:        return result
10:    else
11:       $\mathbf{E}(n) := \mathbf{E}(n) \setminus \{a : \exists R.C\}$
12:    end if
13: end while
14: return "satisfiable"
```

Function 3 DBRANCH$(n, \mathbb{L}(n))$

```
1:  $\mathbf{D}(n) := \{a{:}\Diamond_i C \in \mathbb{L}(n) \mid a{:}C \notin \mathbb{L}(n)$ and $n$ has no $i$-ancestor $l$ in $\mathbb{P}$ with $a{:}C \in \mathbb{L}(l)\} \cup$
    $\{(a,b){:}\Diamond_i R \in \mathbb{L}(n) \mid (a,b){:}R \notin \mathbb{L}(n)$ and $n$ has no $i$-ancestor $l$ in $\mathbb{P}$ with $(a,b){:}R \in \mathbb{L}(l)\}$
2:  while $\mathbf{D}(n) \neq \varnothing$ do
3:     pick an assertion $\alpha \in \mathbf{D}(n)$, add a new node $n'$ to $\mathbb{P}$ where $n' = f_\Diamond(\alpha, n)$.
4:     Let $\mathbb{L}(n, n') := \{i\}$.
5:     if $\alpha$ is of the form $a : \Diamond_i C$ then
6:        Let $\mathbb{L}(n') := \{a : C\}$.
7:     else if $\alpha$ is of the form $(a,b) : \Diamond_i R$ then
8:        Let $\mathbb{L}(n') := \{(a,b) : R\}$.
9:     end if
10:    $\mathbb{B}(n') := \mathbb{B}(n') \cup \mathbb{B}_i(n)$
11:    result := SAT$(n', \mathbb{L}(n'), False)$
12:    if result $\neq$ "satisfiable" then
13:       return result
14:    else
15:       $\mathbf{D}(n) := \mathbf{D}(n) \setminus \{\alpha\}$
16:    end if
17: end while
18: return "satisfiable"
```

algorithm expands the current node n. First, it deals with all the $\exists$-constraints by calling function EBRANCH (Line 18) and then it deals with all the $\Diamond$-constraints by calling function DBRANCH (Line 20).

The function EBRANCH (Function 2) gathers all the $\exists$-constraints in $\mathbb{L}(n)$ that do not have a witness in the current constraint system. For each such constraint $a : \exists R.C$, a witness, say x, is assigned by the global function $f_\exists$ and an auxiliary constraint system $\mathbb{L}^x(n)$ is created to work exclusively on constraints related to x. Because of backtracking and the restart technique, $\mathbb{L}(n)$ may already contain some constraints involving x. These constraints are also added to $\mathbb{L}^x(n)$ (Function 2, Line 3). The $\forall$-rule is then applied exhaustively (Function 2, Lines 4-6). Next the algorithm works on the auxiliary constraint system $\mathbb{L}^x(n)$. If $\mathbb{L}^x(n)$ is satisfiable (Function 2, Lines 7-12), another $\exists$-constraint will be reusing the same space used by $\mathbb{L}^x(n)$. The function

DBranch (Function 3) works on all the $\Diamond$-constraints in the same way as the function EBranch does.

Recall that among three types of return values, a node name is only returned because of backtracking. After the function call EBranch in line 18, if the return value is "satisfiable", then we need to further evaluate $\Diamond$-constraints by calling function DBranch (Lines 19-21). Thus, the return value of variable "result" may be modified by function DBranch.

At this point (Line 22), if the return value of "result" is the name of the current node which is *not* an auxiliary node, then the construction of the constraint system of the current node is restarted. The program remains in the outer while loop and goes back to Line 2 which is the restarting point. Otherwise, there are following cases:

1) the return value of "result" is a node name that is not the current node n;
2) the return value of "result" is the current node name. But the node is an auxiliary one;
3) the return value of "result" is "unsatisfiable";
4) the return value of "result" is "satisfiable".

In all of these cases, function Sat returns to the caller, and before it returns, the space taken by the current constraint system is released for future use (see Lines 24-30). In cases 1) and 2), the algorithm recursively returns until it reaches a *non-auxiliary* constraint system whose name matches the return value of "result". Clearly, for each node, there is a non-auxiliary constraint system. Since backtracking is checked before functions EBranch and DBranch in the same constraint system, if there is no need for backtracking at all, the result value will be either "satisfiable" or "not satisfiable" and function Sat will certainly return (as opposed to remaining in the outer while loop) and therefore Algorithm 1 terminates.

Next we show that the algorithm $\mathcal{ALCS}5_m$-Sat, an implementation of the tableau algorithm Λ, runs in PSpace. For any execution of $\mathcal{ALCS}5_m$-Sat, a single path $\mathbb{P}$ of the constraint tree is maintained. It suffices to show that on path $\mathbb{P}$, the number of nodes and the totally number of constraints in each constraint system are polynomially bounded by the number of constraints in the initial constraint system at all times during the execution. Indeed, since the TBox is acyclic and function $f_\exists$ uniquely assigns a witness for each $\exists$-constraint in a constraint system, the depth of the (auxiliary) constraint systems created because of the $\exists$-rule or $\Diamond$-rule is linearly bounded by the length of the constraints in the original constraint system. Moreover, in the same constraint system, if the $\exists$-rule is applied to an $\exists$-constraint, it will not be applicable to the same constraint unless the whole constraint system is reconstructed (see Function 2, Line 11). Same situation applies to a $\Diamond$-constraint (see Function 3, Line 9). For each auxiliary constraint system created by the $\exists$-rule or each non-auxiliary constraint system created by the $\Diamond$-rule, before the algorithm Sat returns, the constraint system is discarded so that the same space could be reused (see Algorithm 1, Lines 25-29). In fact, every time before Sat returns, the constraint system on which it was working will be discarded. If the constraint system is not an auxiliary one, the set $\mathbb{B}(n)$ will be discarded and the node n (that the constraint system belongs to) as well as the corresponding edge will be removed from path $\mathbb{P}$. When backtracking is needed, all the (auxiliary/non-auxiliary) constraint systems will be recursively discarded to the point where the backtracking reaches to the appropriate node on the path $\mathbb{P}$ (see Algorithm 1, Lines 12-16 and Lines 24-30). It follows that the next theorem is true.

Theorem 3 *The $\mathcal{ALCS}5_m$ acyclic knowledge base satisfiability problem can be implemented in* PSPACE.

5 Conclusion

In this article, we have studied the epistemic description language $\mathcal{ALCS}5_m$ that extends $\mathcal{ALCK}_m$ and $\mathcal{ALCS}4_m$ with modalized roles, and provided a sound and complete PSPACE-implementable tableau algorithm for deciding the satisfiability of an $\mathcal{ALCS}5_m$ knowledge base with an acyclic TBox. The main difficulty in transitioning from the PSPACE results for $\mathcal{ALCK}_m$ and $\mathcal{ALCS}4_m$ in [6] to the current results involves dealing with the issue of symmetry in accessibility relations. Although the only difference between classical modal logics $\mathbf{S}4_m$ and $\mathbf{S}5_m$ is the Negative Introspection axiom, a naive approach of just adding a tableau expansion rule that corresponds to the Negative Introspection axiom to the tableau algorithm for $\mathcal{ALCS}4_m$ is not sufficient to yield a sound and complete PSPACE-implementable algorithm for $\mathcal{ALCS}5_m$ satisfiability. Because of the symmetry of the (epistemic) accessibility relations, backtracking is needed for $\mathcal{ALCS}5_m$ tableau algorithm but not necessarily the case in $\mathcal{ALCS}4_m$. The PSPACE implementation of the tableau algorithm for $\mathcal{ALCS}5_m$ is also rather involved compared to that of the tableau algorithm for $\mathcal{ALCS}4_m$ due to backtracking.

6 Acknowledgments

I acknowledge with gratitude to anonymous reviewers who helped me greatly in improving the quality of this article.

References

1. Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning about Knowledge.* MIT Press, 1995.
2. Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The description logic handbook.* Cambridge University Press New York, NY, USA, 2007.
3. Alessandro Artale, Carsten Lutz, and David Toman. A description logic of change. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 218–223, 2007.

4. Martin Buchheit, Francesco M. Donini, and Andrea Schaerf. Decidable reasoning in terminological knowledge representation systems. *JAIR*, 1:109–138, 1993.
5. Carsten Lutz, Holger Sturm, Frank Wolter, and Michael Zakharyaschev. A tableau decision algorithm for modalized $\mathcal{ALC}$ with constant domains. *Studia Logica*, 72(2):199–232, 2002.
6. Jia Tao, Giora Slutzki, and Vasant Honavar. PSPACE tableau algorithms for acyclic modalized $\mathcal{ALC}$. *Journal of Automated Reasoning*, 49:551–582, 2012.
7. Franz Baader and Armin Laux. Terminological logics with modal operators. In *IJCAI (1)*, pages 808–815, 1995.
8. Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt, and Andrea Schaerf. An epistemic operator for description logics. *Artif. Intell.*, 100(1-2):225–274, 1998.
9. Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artif. Intell.*, 48(1):1–26, 1991.
10. John-Jules Ch. Meyer and Wiebe Van Der Hoek. *Epistemic Logic for AI and Computer Science*. Cambridge University Press, 2004.
11. Ian Horrocks, Ullrich Hustadt, Ulrike Sattler, and Renate Schmidt. Chapter 4 computational modal logic. *Studies in Logic and Practical Reasoning*, 3:181–245, 2007.

A Proof of Theorem 1

Theorem 1 *Let $\mathbb{M} = \langle S,\ \pi,\ \mathcal{E}_1, ..., \mathcal{E}_m\rangle$ be an $\mathbf{S}5_m$-structure and $\mathcal{T}$ an acyclic TBox such that $\mathbb{M} \vDash \mathcal{T}$. Let $\mathbb{G}$ be a constraint graph obtained using $\mathcal{T}$, α an L-, G-, or T-rule and $\mathbb{G}_\alpha$ a constraint graph obtained by applying α to $\mathbb{G}$. If $\mathbb{M} \Vdash \mathbb{G}$ via σ, then there exists a semantic extension $\mathbb{M}_\alpha$ of $\mathbb{M}$ with respect to $N_\Sigma \cup \mathcal{O}_\mathbb{G}$ such that $\mathbb{M}_\alpha \Vdash \mathbb{G}_\alpha$ via σ' (extension of σ) and $\mathbb{M}_\alpha \vDash \mathcal{T}$. Furthermore, $\mathbb{M}_\alpha \Vdash \mathbb{G}$.*

Proof Assume the hypotheses.

– If α is a $\sqcap$-rule, then there is a constraint $a : C_1 \sqcap C_2 \in \mathbb{L}(n)$ in $\mathbb{G}$ and $\{a : C_1, a : C_2\} \not\subseteq \mathbb{L}(n)$. After applying $\sqcap$-rule, $\mathbb{L}(n) = \mathbb{L}(n) \cup \{a : C_1, a : C_2\}$. Since $\mathbb{M} \Vdash \mathbb{G}$ and $a : C_1 \sqcap C_2 \in \mathbb{L}(n)$, by Definition 2, we have $(\mathbb{M}, \sigma(n)) \vDash C_1 \sqcap C_2(a)$. It follows that $a^{\pi(\sigma(n))} \in (C_1 \sqcap C_2)^{\pi(\sigma(n))}$, which means that $a^{\pi(\sigma(n))} \in C_1^{\pi(\sigma(n))}$ and $a^{\pi(\sigma(n))} \in C_2^{\pi(\sigma(n))}$. Hence, $(\mathbb{M}, \sigma(n)) \vDash C_1(a)$ and $(\mathbb{M}, \sigma(n)) \vDash C_2(a)$. Thus, $\mathbb{G}_\alpha$ obtained by application of $\sqcap$-rule from $\mathbb{G}$ is satisfied by $\mathbb{M}$ via σ.

– If α is a $\sqcup$-rule, then there is a constraint $a : C_1 \sqcup C_2 \in \mathbb{L}(n)$ in $\mathbb{G}$ and $\{a : C_1, a : C_2\} \cap \mathbb{L}(n) = \varnothing$. Since $\mathbb{M} \Vdash \mathbb{G}$ and $a : C_1 \sqcup C_2 \in \mathbb{L}(n)$, by Definition 2, we have $(\mathbb{M}, \sigma(n)) \vDash C_1 \sqcup C_2(a)$ and therefore $a^{\pi(\sigma(n))} \in (C_1 \sqcup C_2)^{\pi(\sigma(n))}$. This means that $a^{\pi(\sigma(n))} \in C_1^{\pi(\sigma(n))}$ or $a^{\pi(\sigma(n))} \in C_2^{\pi(\sigma(n))}$. Hence, $(\mathbb{M}, \sigma(n))$ satisfies $C_1(a)$ or $C_2(a)$ (or both). It follows that $\sqcup$-rule can be applied in a way such that $\mathbb{G}_\alpha$ is satisfied by $\mathbb{M}$ via σ.

– If α is the $\exists$-rule, then there is a node n with $a : \exists R.C \in \mathbb{L}(n)$, no L-, T-, or $\Box$-rule except the $\exists$-rule is applicable, and there is no $b \in \mathcal{O}_\mathbb{G}$ such that $\{(a, b) : R, b : C\} \subseteq \mathbb{L}(n)$. After applying α, $\{(a, c) : R, c : C\} \subseteq \mathbb{L}(n)$ where $c = f_\exists(a : \exists R.C, n)$. Since $\mathbb{M} \Vdash \mathbb{G}$, there must exist an element $d \in \Delta$ such that $(a^{\pi(\sigma(n))}, d) \in R^{\pi(\sigma(n))}$ and $d \in C^{\pi(\sigma(n))}$. Define the interpretation π' as π except for the individual c: $c^{\pi'(\sigma(n))} = d$. Let $N_1 = N_\Sigma \cup \mathcal{O}_\mathbb{G} \cup \{c\}$ and $N_2 = N_\Sigma \cup \mathcal{O}_\mathbb{G}$. Then $(\pi'|_{N_1})|_{N_2} = \pi|_{N_2}$. Therefore, the resulting $\mathbb{G}_\alpha$ is satisfied by $\mathbb{M}_\alpha$ via σ where $\mathbb{M}_\alpha = \langle S,\ \pi',\ \mathcal{E}_1, ..., \mathcal{E}_m\rangle$ is a semantic extension of $\mathbb{M}$ with respect to N_2.

– If α is a $\forall$-rule, then there is a node n with $\{a : \forall R.C, (a, b) : R\} \subseteq \mathbb{L}(n)$ and $b : C \notin \mathbb{L}(n)$. Since $\mathbb{M} \Vdash \mathbb{G}$ and $a : \forall R.C \in \mathbb{L}(n)$, by Definition 2, we have $(\mathbb{M}, \sigma(n)) \vDash \forall R.C(a)$, which means that for all $d \in \Delta$, $(a^{\pi(\sigma(n))}, d) \in R^{\pi(\sigma(n))}$ implies $d \in C^{\pi(\sigma(n))}$. Moreover, $(a, b) : R \in \mathbb{L}(n)$ implies $(\mathbb{M}, \sigma(n)) \vDash R(a, b)$, which means $(a^{\pi(\sigma(n))}, b^{\pi(\sigma(n))}) \in R^{\pi(\sigma(n))}$. After applying the $\forall$-rule, $b : C$ is added to $\mathbb{L}(n)$. The resulting $\mathbb{G}_\alpha$ is satisfied by $\mathbb{M}$ via σ.

- If α is a $\bot$-rule, then there is a node n such that $\{a : C, a : \neg C\} \subseteq \mathbb{L}(n)$ and $a : \bot \notin \mathbb{L}(n)$. By Definition 2, $a : C \in \mathbb{L}(n)$ implies $(\mathbb{M}, \sigma(n)) \vDash C(a)$, which means that $a^{\pi(\sigma(n))} \in C^{\pi(\sigma(n))} = \Delta \setminus (\neg C)^{\pi(\sigma(n))}$. Therefore, $a^{\pi(\sigma(n))} \notin (\neg C)^{\pi(\sigma(n))}$. However, $a : \neg C \in \mathbb{L}(n)$ implies $(\mathbb{M}, \sigma(n)) \vDash \neg C(a)$, which means that $a^{\pi(\sigma(n))} \in (\neg C)^{\pi(\sigma(n))}$. Hence, $a^{\pi(\sigma(n))} \in \varnothing = \bot^{\pi(\sigma(n))}$.
- If α is the $\Diamond$-rule, then there are two cases.
  - There is a node n with $a : \Diamond_i C \in \mathbb{L}(n)$, $a : C \notin \mathbb{L}(n)$, no L-, T-, or $\Box$-rule is applicable, and n has no i-neighbor l with $a : C \in \mathbb{L}(l)$. By Definition 2, $a : \Diamond_i C \in \mathbb{L}(n)$ implies $(\mathbb{M}, \sigma(n)) \vDash \Diamond_i C(a)$. This means that there is a state s with $(\sigma(n), s) \in \mathcal{E}_i$ and $a^{\pi(s)} \in C^{\pi(s)}$. After applying the $\Diamond$-rule, a new node n' is added to $\mathbb{G}$ with $\mathbb{L}(n') = \{a : C\}$ where $n' = f_{\Diamond}(a : \Diamond_i C, n)$ and $\mathbb{L}(n', n'') = \{i\}$ for each i-neighbor n' of n. Extend σ to σ' such that $\sigma'(n') = s$. Since $\mathbb{M}$ is an $\mathbb{S}5$-structure, $(\sigma'(n'), \sigma(n'')) = (s, \sigma(n'')) \in \mathcal{E}_i$ for each i-neighbor n'' of n. Therefore, $\mathbb{M}$ satisfies the resulting $\mathbb{G}_\alpha$ via σ'.
  - There is a node n with $(a, b) : \Diamond_i R \in \mathbb{L}(n)$, $(a, b) : R \notin \mathbb{L}(n)$, no L-, T-, or $\Box$-rule is applicable, and n has no i-neighbor l with $(a, b) : R \in \mathbb{L}(l)$. The proof of this case is similar to that of the previous case and is omitted.
- If α is the $\Box$-rule, then there are two cases.
  - There are two nodes n and n' with $a : \Box_i C \in \mathbb{L}(n)$, n being an i-neighbor of n' and $a : C \notin \mathbb{L}(n')$. After the application, $a : C \in \mathbb{L}(n')$. Since $\mathbb{M} \Vdash \mathbb{G}$ and $a : \Box_i C \in \mathbb{L}(n)$, by Definition 2, we have $(\mathbb{M}, \sigma(n)) \vDash \Box_i C(a)$. Moreover, $i \in \mathbb{L}(n, n')$ implies that $(\sigma(n), \sigma(n')) \in \mathcal{E}_i$. Therefore, $(\mathbb{M}, \sigma(n')) \vDash C(a)$. It follows that $\mathbb{M}$ satisfies the resulting $\mathbb{G}_\alpha$ via σ.
  - There are two nodes n and n' with $(a, b) : \Box_i R \in \mathbb{L}(n)$, n being an i-neighbor of n' and $(a, b) : R \notin \mathbb{L}(n')$. The proof of this case is similar to that of the previous case and is omitted.
- If α is a T-rule, then $a : A \in \mathbb{L}(n)$, $A \doteq D \in \mathcal{T}$ and $a : D' \notin \mathbb{L}(n)$ where D' is the NNF of D. After applying α, we have $a : D' \in \mathbb{L}(n)$. Since $\mathbb{M} \Vdash \mathbb{G}$ and $\mathbb{M} \vDash \mathcal{T}$, $a^{\pi(\sigma(n))} \in A^{\pi(\sigma(n))} = (D')^{\pi(\sigma(n))}$. Therefore, $(\mathbb{M}, \sigma(n)) \vDash D'(a)$ and hence, $\mathbb{M} \Vdash \mathbb{G}_\alpha$ via σ.
- If α is an N-rule, then $a : \neg A \in \mathbb{L}(n)$, $A \doteq D \in \mathcal{T}$ and $a : D' \notin \mathbb{L}(n)$ where D' is the NNF of $\neg D$. After applying the N-rule, $a : D' \in \mathbb{L}(n)$. Since $\mathbb{M} \Vdash \mathbb{G}$ and $\mathbb{M} \vDash \mathcal{T}$, we have $(\mathbb{M}, \sigma(n)) \vDash A \doteq D$. It follows that $a^{\pi(\sigma(n))} \in (\neg A)^{\pi(\sigma(n))} = (\neg D)^{\pi(\sigma(n))} = (D')^{\pi(\sigma(n))}$. Thus, $(\mathbb{M}, \sigma(n)) \vDash D'(a)$. Therefore, $\mathbb{M} \Vdash \mathbb{G}_\alpha$ via σ.

It follows that after the application of every expansion rule, the resulting constraint graph $\mathbb{G}_\alpha$ is satisfied by $\mathbb{M}_\alpha$ which, except after applying an $\exists$-rule, is the same as $\mathbb{M}$. When α is an $\exists$-rule, $\mathbb{M}_\alpha$ differs from $\mathbb{M}$ only in the interpretation of the newly picked individual name. Therefore, $\mathcal{T}$ is valid in $\mathbb{M}_\alpha$. Moreover, since $\mathbb{M}_\alpha$ is a semantic extension of $\mathbb{M}$ restricted to $N_\Sigma \cup \mathcal{O}_\mathbb{G}$, the constraint graph $\mathbb{G}$ is satisfied by $\mathbb{M}_\alpha$. $\square$

B Proof of Lemma 4

Lemma 4 *Let $\mathcal{T}$ be an acyclic TBox and $\mathbb{G}$ an open complete constraint graph with respect to local, global and terminological expansion rules. Then for every $A \in N_\mathcal{C}$ and every $a \in \Delta$, $a : \neg A \in \mathbb{L}(n)$ implies $(\mathbb{M}_\mathbb{G}, n) \vDash \neg A(a)$.*

Proof There are two cases, and for both, since $\mathbb{G}$ is open, $a : A \notin \mathbb{L}(n)$.

(1) When A is primitive, since $\mathbb{G}$ is open, $a : \neg A \in \mathbb{L}(n)$ implies $a : A \notin \mathbb{L}(n)$. Then, $a \notin A^{\pi(n)}$ by Definition 3. Thus, $a \in (\neg A)^{\pi(n)}$. Therefore, $(\mathbb{M}_\mathbb{G}, n) \vDash \neg A(a)$ by Theorem 1 and Definition 2.

(2) If A is not primitive, i.e., there is a definition $A \doteq D \in \mathcal{T}$, we prove by induction on the structure of D. For the base case where the concept names involved in D are primitive, we have the following cases:
   1. D is of the form $\neg B$ where B is primitive. Since $\mathbb{G}$ is complete, $a : B \in \mathbb{L}(n)$. By Definition 3, $a \in B^{\pi(n)}$ if and only if $a \notin (\neg B)^{\pi(n)}$. Since $\mathbb{G}$ is open, $a : \neg A \in \mathbb{L}(n)$ implies $a : A \notin \mathbb{L}(n)$. However, $A^{\pi(n)} = \{b \mid b : A \in \mathbb{L}(n)\} \cup (\neg B)^{\pi(n)}$. This implies $a \notin A^{\pi(n)}$. Thus, $a \in (\neg A)^{\pi(n)}$. Therefore, $(\mathbb{M}_\mathbb{G}, n) \vDash \neg A(a)$.

2. D is of the form $B_1 \sqcap B_2$ where B_1 and B_2 are primitive. Since $\mathbb{G}$ is complete, $a : \neg B_1 \sqcup \neg B_2 \in \mathbb{L}(n)$, and $a : \neg B_1$ or $a : \neg B_2$ is in $\mathbb{L}(n)$. W.l.o.g., suppose $a : \neg B_1 \in \mathbb{L}(n)$. Since $\mathbb{G}$ is open, $a : B_1 \notin \mathbb{L}(n)$. Because B_1 is primitive, by Definition 3, $a \notin B_1^{\pi(n)}$. Thus, $a \in (\neg B_1)^{\pi(n)}$, which implies that $a \notin (B_1 \sqcap B_2)^{\pi(n)}$. However, $A^{\pi(n)} = \{b \mid b : A \in \mathbb{L}(n)\} \cup (B_1 \sqcap B_2)^{\pi(n)}$ and $a : A \notin \mathbb{L}(n)$. Hence, $a \notin A^{\pi(n)}$. Thus, $a \in (\neg A)^{\pi(n)}$. Therefore, $(\mathbb{M}_{\mathbb{G}}, n) \vDash \neg A(a)$.
3. D is of the form $B_1 \sqcup B_2$ where B_1 and B_2 are primitive. Since $\mathbb{G}$ is complete, $a : \neg B_1 \sqcap \neg B_2 \in \mathbb{L}(n)$ and $\{a : \neg B_1, a : \neg B_2\} \subseteq \mathbb{L}(n)$. Since $\mathbb{G}$ is open, $a : B_1 \notin \mathbb{L}(n)$ and $a : B_2 \notin \mathbb{L}(n)$. Because B_1 and B_2 are primitive, by Definition 3, $a \notin B_1^{\pi(n)}$ and $a \notin B_2^{\pi(n)}$. Thus, $a \notin (B_1 \sqcup B_2)^{\pi(n)}$. However, $A^{\pi(n)} = \{b \mid b : A \in \mathbb{L}(n)\} \cup (B_1 \sqcup B_2)^{\pi(n)}$ and $a : A \notin \mathbb{L}(n)$. Hence, $a \notin A^{\pi(n)}$. Thus, $a \in (\neg A)^{\pi(n)}$. Therefore, $(\mathbb{M}_{\mathbb{G}}, n) \vDash \neg A(a)$.
4. D is of the form $\exists R.B$ where B is primitive. Since $\mathbb{G}$ is complete, $a : \forall R.\neg B \in \mathbb{L}(n)$ and for every b, if $(a, b) : R \in \mathbb{L}(n)$, then $b : \neg B \in \mathbb{L}(n)$. Suppose $(a, b) : R \in \mathbb{L}(n)$. Since B is primitive and $\mathbb{G}$ is open, it follows that $b \notin B^{\pi(n)}$. Moreover, $(a, b) \in R^{\pi(n)}$ by Lemma 3 and Definition 1. Hence, for every b, $(a, b) \in R^{\pi(n)}$ implies $b \notin B^{\pi(n)}$. Thus, $a \in (\forall R.\neg B)^{\pi(n)}$ and therefore, $a \notin (\exists R.B)^{\pi(n)}$. However, $A^{\pi(n)} = \{c \mid c : A \in \mathbb{L}(n)\} \cup (\exists R.B)^{\pi(n)}$ and $a : A \notin \mathbb{L}(n)$. Hence, $a \notin A^{\pi(n)}$, which is equivalent to $a \in (\neg A)^{\pi(n)}$. Therefore, $(\mathbb{M}_{\mathbb{G}}, n) \vDash \neg A(a)$.
5. D is of the form $\forall R.B$ where B is primitive. Since $\mathbb{G}$ is complete, $a : \exists R.\neg B \in \mathbb{L}(n)$ and there exists b such that $(a, b) : R \in \mathbb{L}(n)$ and $b : \neg B \in \mathbb{L}(n)$. Since B is primitive and $\mathbb{G}$ is open, we have $b \notin B^{\pi(n)}$. By Lemma 3 and Definition 1, we have $(a, b) \in R^{\pi(n)}$. Therefore, there exists b such that $(a, b) \in R^{\pi(n)} \wedge b \notin B^{\pi(n)}$. Thus, $a \in (\exists R.\neg B)^{\pi(n)}$ and hence, $a \notin (\forall R.B)^{\pi(n)}$. However, $A^{\pi(n)} = \{c \mid c : A \in \mathbb{L}(n)\} \cup (\forall R.B)^{\pi(n)}$ and $a : A \notin \mathbb{L}(n)$. Thus, $a \notin A^{\pi(n)}$, which is equivalent to $a \in (\neg A)^{\pi(n)}$. Therefore, $(\mathbb{M}_{\mathbb{G}}, n) \vDash \neg A(a)$.
6. D is of the form $\Diamond_i B$ where B is primitive. Since $\mathbb{G}$ is complete, $a : \Box_i \neg B \in \mathbb{L}(n)$ and $a : \neg B \in \mathbb{L}(n')$ for each n' with $i \in \mathbb{L}(n, n')$. Note that B is primitive and $\mathbb{G}$ is open. Hence, $a \notin B^{\pi(n')}$ whenever $i \in \mathbb{L}(n, n')$. Thus, $a \in \bigcap_{n' \in \mathcal{E}_i(n)} (\neg B)^{\pi(n')}$. Then, $a \in (\Box_i \neg B)^{\pi(n)}$. Therefore, $a \notin (\Diamond_i B)^{\pi(n)}$. However, $A^{\pi(n)} = \{b \mid b : A \in \mathbb{L}(n)\} \cup (\Diamond_i B)^{\pi(n)}$ and $a : A \notin \mathbb{L}(n)$. Hence, $a \notin A^{\pi(n)}$. It follows that $(\mathbb{M}_{\mathbb{G}}, n) \vDash \neg A(a)$.
7. D is of the form $\Box_i B$ where B is primitive. Since $\mathbb{G}$ is complete, we have $a : \Diamond_i \neg B \in \mathbb{L}(n)$ and there exists n' such that $i \in \mathbb{L}(n, n')$ and $a : \neg B \in \mathbb{L}(n')$. Note that B is primitive and $\mathbb{G}$ is open. Thus, $a \notin B^{\pi(n')}$. Hence, $a \in \bigcup_{n' \in \mathcal{E}_i(n)} (\neg B)^{\pi(n')}$. Then, $a \in (\Diamond_i \neg B)^{\pi(n)}$. Therefore, $a \notin (\Box_i B)^{\pi(n)}$. However, $A^{\pi(n)} = \{a \mid a : A \in \mathbb{L}(n)\} \cup (\Box_i B)^{\pi(n)}$ and $a : A \notin \mathbb{L}(n)$. Hence, $a \notin A^{\pi(n)}$. It follows that $(\mathbb{M}_{\mathbb{G}}, n) \vDash \neg A(a)$.

Note that for the first five cases, the correctness of the implication that if $a : \neg A \in \mathbb{L}(n)$, then $(\mathbb{M}_{\mathbb{G}}, n) \vDash \neg A(a)$ depends on the fact that the constraint graph $\mathbb{G}$ has no applicable local or terminological expansion rules. For the last two cases, the correctness of the implication depends on the fact that $\mathbb{G}$ has no applicable global or terminological expansion rules.
The induction step is similar to the corresponding base case, except that in the general case, in order to show that $a \notin D^{\pi(n)}$, we use the induction hypothesis rather than relying on the fact that a concept name is primitive when the concept name occurring in D is not primitive. $\Box$