Carsten Lutz
Holger Sturm
Frank Wolter
Michael Zakharyaschev

A Tableau Decision Algorithm for Modalized $\mathcal{ALC}$ with Constant Domains

Abstract. The aim of this paper is to construct a tableau decision algorithm for the modal description logic $\mathbf{K}_{\mathcal{ALC}}$ with constant domains. More precisely, we present a tableau procedure that is capable of deciding, given an $\mathcal{ALC}$-formula φ with extra modal operators (which are applied only to concepts and TBox axioms, but not to roles), whether φ is satisfiable in a model with constant domains and arbitrary accessibility relations. Tableau-based algorithms have been shown to be 'practical' even for logics of rather high complexity. This gives us grounds to believe that, although the satisfiability problem for $\mathbf{K}_{\mathcal{ALC}}$ is known to be NEXPTIME-complete, by providing a tableau decision algorithm we demonstrate that highly expressive description logics with modal operators have a chance to be implementable. The paper gives a solution to an open problem of Baader and Laux [5].

Keywords: modal logics, description logics, tableaux, combining systems, decidability, complexity.

1. Introduction

Description logics are logical formalisms intended for representing knowledge about static application domains in a structured way (see e.g. [11] and references therein). In the basic description logic $\mathcal{ALC}$ [31], for example, we can use the concept

$$\mathsf{Person} \sqcap \exists \mathsf{lives\_in}.\mathsf{Bavaria} \sqcap \forall \mathsf{drinks}.\mathsf{Beer}$$

to describe the persons living in Bavaria and drinking only beer. (Here Person, Bavaria, and Beer are concept names (unary predicates), and lives_in and drinks are role names (binary predicates).) The knowledge that all people living in Bavaria drink only beer and, conversely, that all people drinking only beer live in Bavaria can be represented then by the terminological axiom

$$\mathsf{Person} \sqcap \exists \mathsf{lives\_in}.\mathsf{Bavaria} = \mathsf{Person} \sqcap \forall \mathsf{drinks}.\mathsf{Beer}. \qquad (1)$$

Modal logics are formal systems intended for reasoning with necessity-like and possibility-like operators, in particular, for reasoning about agents' knowledge of their knowledge bases and about the evolution of these knowledge bases in time (see e.g. [12] and references therein). Using the modal operators [*Tom believes*] and [*always*], we can relativize axiom (1) to represent the knowledge of an agent 'Tom' and also add to it some temporal flavor:

$$[\textit{Tom believes}]\,(\mathsf{Person} \sqcap \exists \mathsf{lives\_in.Bavaria} = \mathsf{Person} \sqcap [\textit{always}]\forall \mathsf{drinks.Beer}) \quad (2)$$

i.e., Tom believes that those people who live in Bavaria and nobody else drink only beer at any given moment of time. As just demonstrated, *modalized description logics* provide means for representing distributed knowledge about dynamic application domains. Such logics were constructed e.g. in [10, 30, 5, 6, 35, 37, 38, 39].

Both description logics and modal logics are featured as 'a compromise between expressiveness and effectiveness', which led to the implementation of many of them as *representation and reasoning systems* [15, 20, 28]. Description logics with modal operators are not so well-behaved. The interaction between description logic constructs and modalities can substantially increase the complexity of reasoning tasks and even make them undecidable [6, 37, 27]. The series of papers [35, 37, 38, 39, 36] identified the following syntactical and semantical 'limits' within which interactions between the modal and the description parts of the combined logics usually preserve decidability:

- Modal operators can be applied to both concepts and terminological axioms (as in the example above) but not to roles.
- In the worlds of a Kripke model interpreting the modal operators, each object and concept name of the description logic can be interpreted both globally (i.e., by the same object or, respectively, set of objects in all worlds) and locally (i.e., by arbitrary objects or sets of objects).[1] Role names can be interpreted only locally.
- The domains of the worlds (on which we interpret the description logic component) can be assumed to be *constant*; the expressive power of the combined language makes it possible to simulate the *varying* or *expanding domain assumptions* which presuppose less interaction.

By allowing modalized or/and global roles, one can easily construct undecidable description logics with epistemic and temporal operators [36].

[1] If we represent a person, say Tom, by the object name tom then we most likely assume that this name is global, while the object name president_of_the_USA, representing the current president of the United States, should clearly be local.

The proofs of decidability in [35, 38, 39] are based on a semantical approach. They do not give any 'practical' decision procedures having a potential to be implemented in reasoning systems which are reasonably fast on reasonably large sets of problems.

The aim of this paper is to construct a tableau decision algorithm for the modal description logic $\mathbf{K}_{\mathcal{ALC}}$ with constant domains. More precisely, we present a tableau procedure which is capable of deciding, given a formula φ of the form (2) (in which modal operators are applied only to concepts and TBox axioms, but not to roles), whether φ is satisfiable in a model with constant domains. Tableau-based algorithms have been shown to be 'practical' even for logics of rather high complexity [14, 19, 21]. This gives us grounds to believe that, although the satisfiability problem for $\mathbf{K}_{\mathcal{ALC}}$ is known to be NEXPTIME-complete [27], by providing a tableau decision algorithm we demonstrate that highly expressive description logics with modal operators have a chance to be implementable.

The logic $\mathbf{K}_{\mathcal{ALC}}$ *with expanding domains* was first considered by Baader and Laux [5] who proved its decidability by designing a tableau satisfiability-checking algorithm. However, they failed to construct such an algorithm working under the *constant domain assumption*, having left this as an open problem and explained some principle difficulties. Somewhat simplified, the idea of constructing tableaux for $\mathbf{K}_{\mathcal{ALC}}$ with expanding domains is as follows. Given a formula φ, we first apply to it the rules of a tableau system for $\mathcal{ALC}$, thus constructing an $\mathcal{ALC}$-model for the non-modal part of φ in the initial world w_0. Then we apply the rules of a tableau system for $\mathbf{K}$ to the modalized concepts and formulas in this model and construct a number of new worlds w_i 'populated' by *the same objects* as in w_0. After that we use the $\mathcal{ALC}$-rules in the worlds w_i and possibly *extend* their domains. And so forth. This straightforward approach does not work in the case of models with constant domains. Having expanded the domain of w_i with a new object a, we have to add a to the domain of w_0 which, in turn, may force us to expand this domain, and so the domains of the w_i as well, with some new objects, and so on (for more details consult [5] and Section 3 below).

The main technical contribution of the paper is that it shows how to design a machinery for constructing tableaux with constant domains. The fundamental difference from the approach of [5] is that our tableau algorithm constructs not a model itself but its representation in the form of a *quasimodel*, the worlds in which are 'populated' by *types* of objects rather than real objects. Quasimodels were first introduced in [35] and used in [32] for designing a tableau procedure for temporal description logics with expanding domains. In fact, the transference of the notion of quasimodels from model-theory in [35] to proof-theory in [32] and the present paper provides a nice example of how an 'abstract' decidability proof can serve as an important basis for an implementable decidability proof.

We have chosen the basic modal description logic $\mathbf{K}_{\mathcal{ALC}}$ for treating in this paper only in order to make the ideas of the tableaux and the employed techniques as clear as possible. The developed methods can be extended to more sophisticated logics, say, **S4** or temporal logics based on $\mathcal{ALC}$. Moreover, the approach developed in this paper can be generalized to the monodic fragments of *first-order* modal and temporal logics [16, 40] by combining tableau procedures for their first-order and modal components in a *modular* way; for details visit `http://www.dcs.kcl.ac.uk/staff/mz`.

It is worth also noting that $\mathbf{K}_{\mathcal{ALC}}$ is closely related to the *Cartesian product* $\mathbf{K} \times \mathbf{S5}$ (cf. [13, 34]). Thus we obtain for free a tableau-based decision procedure for $\mathbf{K} \times \mathbf{S5}$ as well.

The paper is organized in the following way. In the next section we introduce the syntax and the semantics of $\mathbf{K}_{\mathcal{ALC}}$ with constant domains. Section 3 discusses difficulties in designing tableau procedures for modalized description logics under the constant domain assumption and gives an overview of the tableau algorithm developed in this paper. In Section 4 we define constraint systems for $\mathbf{K}_{\mathcal{ALC}}$-formulas, and Section 5 shows how to encode $\mathbf{K}_{\mathcal{ALC}}$-models in the form of quasimodels. The tableau decision algorithm is presented in Section 6 and its termination and correctness is proved in Section 7. We conclude the paper with a discussion of obtained results and related problems.

2. Preliminaries

In this section, we define the syntax and the semantics of the modal description logic $\mathbf{K}_{\mathcal{ALC}}$.

DEFINITION 1 (syntax). Let N_C, N_R, and N_O be countably infinite sets of *concept names*, *role names*, and *object names*, respectively. The set of $\mathbf{K}_{\mathcal{ALC}}$-*concepts* is defined inductively as follows:

1. All concept names as well as the logical constant $\top$ are concepts.
2. If C and D are concepts, R is a role name, and $i < \omega$, then the following expressions are concepts:

$$\neg C, \quad C \sqcap D, \quad C \sqcup D, \quad \exists R.C, \quad \forall R.C, \quad \Diamond_i C, \quad \Box_i C.$$

$\mathbf{K}_{\mathcal{ALC}}$*-formulas* are also defined inductively:

3. If C and D are concepts and a is an object name, then $C = D$ and $a : C$ are (atomic) formulas.
4. If φ and ψ are formulas and $i < \omega$, then the following expressions are formulas:
$$\neg\varphi, \quad \varphi \wedge \psi, \quad \varphi \vee \psi, \quad \Diamond_i\varphi, \quad \Box_i\varphi.$$

Throughout the paper, we denote concept names by A, role names by R, and object names by a. Arbitrary concepts are denoted by C, D, and E, and formulas by φ, ψ, and ϑ.

The semantics of $\mathbf{K}_{\mathcal{ALC}}$ is a natural hybrid of the possible world semantics for $\mathbf{K}$ and the standard set-theoretic semantics for $\mathcal{ALC}$.

DEFINITION 2 (semantics). An $\mathcal{ALC}$*-interpretation* is a pair $\mathcal{I} = (\Delta_{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta_{\mathcal{I}}$ is a non-empty set called the *domain* of $\mathcal{I}$ and $\cdot^{\mathcal{I}}$ maps each concept name A to a subset $A^{\mathcal{I}}$ of $\Delta_{\mathcal{I}}$, each role name R to a binary relation $R^{\mathcal{I}}$ on $\Delta_{\mathcal{I}}$, and each object name a to an element $a^{\mathcal{I}}$ in $\Delta_{\mathcal{I}}$.

A $\mathbf{K}_{\mathcal{ALC}}$*-model* is a triple of the form $\mathfrak{M} = (W, \lhd, I)$, where

- W is a non-empty set (of *worlds*),
- $\lhd$ is a map associating with each $i < \omega$ a binary relation $\lhd_i$ on W, and
- I associates with each $w \in W$ an $\mathcal{ALC}$-interpretation
$$I(w) = (\Delta_w, \cdot^{I(w)})$$
such that, for any $w, w' \in W$, we have
  - $\Delta_w = \Delta_{w'}$ and
  - $a^{I(w)} = a^{I(w')}$ for all $a \in N_O$.

We now define the *value* $C^{I(w)}$ of a concept C in a world w in $\mathfrak{M}$ by taking

$$\begin{aligned}
\top^{I(w)} &= \Delta_w,\\
(\neg C)^{I(w)} &= \Delta_w \setminus C^{I(w)},\\
(C \sqcap D)^{I(w)} &= C^{I(w)} \cap D^{I(w)},\\
(C \sqcup D)^{I(w)} &= C^{I(w)} \cup D^{I(w)},\\
(\exists R.C)^{I(w)} &= \{d \in \Delta_w \mid \exists d' \in C^{I(w)}\, (dR^{I(w)}d')\},\\
(\forall R.C)^{I(w)} &= \{d \in \Delta_w \mid \forall d' \in \Delta_w\, (dR^{I(w)}d' \Rightarrow d' \in C^{I(w)})\},\\
(\Diamond_i C)^{I(w)} &= \{d \in \Delta_w \mid \exists w' \in W\, (w \lhd_i w' \;\&\; d \in C^{I(w')})\},\\
(\Box_i C)^{I(w)} &= \{d \in \Delta_w \mid \forall w' \in W\, (w \lhd_i w' \Rightarrow d \in C^{I(w')})\}.
\end{aligned}$$

The *truth-relation* $(\mathfrak{M}, w) \models \varphi$ for a $\mathbf{K}_{\mathcal{ALC}}$-formula φ is defined as follows:

$(\mathfrak{M}, w) \models C = D$ iff $C^{I(w)} = D^{I(w)}$.

$(\mathfrak{M}, w) \models a : C$ iff $a^{I(w)} \in C^{I(w)}$.

$(\mathfrak{M}, w) \models \neg\varphi$ iff $(\mathfrak{M}, w) \not\models \varphi$.

$(\mathfrak{M}, w) \models \varphi \wedge \psi$ iff $(\mathfrak{M}, w) \models \varphi$ and $(\mathfrak{M}, w) \models \psi$.

$(\mathfrak{M}, w) \models \varphi \vee \psi$ iff $(\mathfrak{M}, w) \models \varphi$ or $(\mathfrak{M}, w) \models \psi$.

$(\mathfrak{M}, w) \models \Diamond_i\varphi$ iff there is $w' \in W$ such that $w \lhd_i w'$ and $(\mathfrak{M}, w') \models \varphi$.

$(\mathfrak{M}, w) \models \Box_i\varphi$ iff $(\mathfrak{M}, w') \models \varphi$ for all $w' \in W$ such that $w \lhd_i w'$.

Remark 1. We do not make the *unique name assumption* (UNA): for two distinct object names $a, b \in N_O$, we may have $a^{I(w)} = b^{I(w)}$ for some $w \in W$. However, with minor changes, all the results in this paper can be proved for models satisfying UNA. Note that the object names are assumed to be *global*, i.e., for all $a \in N_O$ and $w, w' \in W$, we have $a^{I(w)} = a^{I(w')}$. On the contrary, the values of concept names are defined *locally*. Neither of these assumptions is essential; in particular, global concepts can be defined via local ones: for a formula φ, define *paths*(φ) inductively as follows

- $paths(C = D) = paths(a : C) = \{\epsilon\}$,
- $paths(\neg\psi) = paths(\psi)$,
- $paths(\psi_1 \wedge \psi_2) = paths(\psi_1 \vee \psi_2) = paths(\psi_1) \cup paths(\psi_2)$,
- $paths(\Diamond_i\psi) = paths(\Box_i\psi) = \{\epsilon\} \cup \{iw \mid w \in paths(\psi)\}$.

For any $w \in paths(\varphi)$ with $w = i_1 \cdots i_k$, we use $\Box_w$ to denote the nesting $\Box_{i_1} \cdots \Box_{i_k}$. Now let φ be a formula containing a concept name A that should be global and let $i_1, \ldots, i_k$ be the modal indices occurring in φ. It is not hard to see that globality of A (at least w.r.t. those worlds that are "relevant" for evaluating φ) can be enforced by taking the conjunction of φ and the formula $\bigwedge_{w \in paths(\varphi)} \bigwedge_{i \in \{i_1,\ldots,i_k\}} \Box_w(\top = ((\neg A \sqcup \Box_i A) \sqcap (A \sqcup \Box_i \neg A)))$.

Let us return to the discussion of the semantics. By requiring $\Delta_w = \Delta_{w'}$ for all $w, w' \in W$, we choose models with *constant domains*. In models with *expanding domains* we would only have $\Delta_w \subseteq \Delta_{w'}$ for any $w, w' \in W$ such that $w \lhd_i w'$ for some $i < \omega$. Models with *varying domains* impose no restrictions on the domains at all. As noted in the introduction, both varying domains and expanding domains can be simulated using constant domains [37]. ∎

We are interested in the following inference problems.

DEFINITION 3 (inference). A formula φ is *satisfiable* if there exist a model $\mathfrak{M} = (W, \lhd, I)$ and a world $w \in W$ such that $(\mathfrak{M}, w) \models \varphi$. A concept C is *satisfiable* if there exist $\mathfrak{M} = (W, \lhd, I)$ and $w \in W$ such that $C^{I(w)} \neq \emptyset$. A concept C is *subsumed* by a concept D if $C^{I(w)} \subseteq D^{I(w)}$ for all models $\mathfrak{M} = (W, \lhd, I)$ and all $w \in W$.

Remark 2. Note that concept satisfiability and concept subsumption can be reduced to the satisfiability of formulas. A concept C is satisfiable iff the formula $a : C$ is satisfiable and a concept C subsumes a concept D iff the formula $a : (D \sqcap \neg C)$ is unsatisfiable.

In the DL literature, concept satisfiability and subsumption are often considered w.r.t. TBoxes, i.e., sets of concept equations $C = D$. It is easily seen that reasoning w.r.t. TBoxes can also be reduced to formula satisfiability. Note, however, that TBoxes encoded via formulas are local in the same way as concept names are: the TBox encoded into the formula is only valid for the "root" world, all other worlds have their own set of valid $C = D$ equations. In contrast to concept names, global TBoxes cannot be defined. See the conclusion for more comments on this issue.

In what follows, we only consider the satisfiability of formulas. ∎

Say that a formula φ is *equivalent* to a formula ψ when $(\mathfrak{M}, w) \models \varphi$ iff $(\mathfrak{M}, w) \models \psi$ for every model $\mathfrak{M}$ and every world w in it. Similarly, a concept C is *equivalent* to a concept D if $C^{I(w)} = D^{I(w)}$ for all models $\mathfrak{M} = (W, \lhd, I)$ and all $w \in W$.

The formula $C = D$ is clearly equivalent to $(\neg C \sqcup D) \sqcap (\neg D \sqcup C) = \top$. So without loss of generality we can assume that in every atomic formula of the form $C = D$ the concept D is $\top$. Furthermore, we generally assume formulas and concepts to be in negation normal form.

DEFINITION 4 (negation normal form). A concept C is said to be in *negation normal form* (NNF) if negation occurs in C only in front of concept names. A formula φ is in *negation normal form* if negation occurs in φ only in front of concept names and atomic formulas of the form $C = D$.

Each concept C can be transformed into an equivalent concept in NNF by pushing negation inwards with the help of de Morgan's law, the duality between $\exists$ and $\forall$, and between $\Diamond$ and $\Box$. The NNF of $\neg C$ will be denoted by ${\sim}C$. Similarly, each formula can be transformed into an equivalent one in NNF by employing de Morgan's law, the duality between $\Diamond$ and $\Box$, and the fact that $\neg(a : C)$ is equivalent to $a : \neg C$.

3. Tableau algorithms and constant domains

In this section, we introduce tableau algorithms in general, discuss difficulties in designing such algorithms for modalized description logics under the constant domain assumption, and give an overview of the tableau procedure developed in this paper.

To decide whether a given formula ϑ is satisfiable, a tableau algorithm tries to construct a model for ϑ by repeatedly applying *completion rules* to an appropriate data structure. In the case of $\mathcal{ALC}$, this data structure is usually a *constraint system* [4, 5, 17], consisting of *constraints* which are formulas, expressions of the form $x : C$, or expressions of the form $(x, x') : R$, where x and x' are *variables* or object names from N_O. For now, it is convenient to think of variables and object names as representing domain objects.

In the case of $\mathbf{K}_{\mathcal{ALC}}$, a more complex data structure is needed. In this paper, it is a *completion tree* whose edges represent accessibility relations and whose nodes are labeled with constraint systems representing $\mathcal{ALC}$ interpretations. The tableau algorithm starts with a completion tree containing only a single node labeled with a constraint system containing only ϑ (and possibly some additional constraints required for some technical reasons). The completion rules are applied until (i) a contradictory completion tree is obtained or (ii) a contradiction-free completion tree is found which is *complete* in the sense that no more rule is applicable to it. Here, a completion tree contains a contradiction if, for example, one of its nodes is labeled with a constraint system containing both $x : A$ and $x : \neg A$. To illustrate how completion rules look like, we sketch some standard rules which can be found in most tableau algorithms for description logics (see e.g., [4, 5, 17, 23]).

1. If a constraint system S contains the formula $C = \top$ and a variable or an object name x, then we add $x : C$ to S.
2. If a constraint system S contains $x : C \sqcup D$, then we add to S either $x : C$ or $x : D$.
3. If a constraint system S contains $x : \exists R.C$, then we add to S two constraints $v : C$ and $(x, v) : R$, where v is a fresh variable that was not used in S before.
4. If the label S of a node g in a completion tree $\boldsymbol{T}$ contains the constraint $x : \Diamond_i C$, then we add to $\boldsymbol{T}$ a new node g' as a successor of g and label it with the constraint system containing $x : C$ and $x : D$ for every $x : \Box_i D$ in S.

The second rule above is nondeterministic: its application yields more than one possible outcome. In the presence of nondeterministic rules, a tableau algorithm terminates successfully if the completion rules can be applied in such a way that the result is a complete and contradiction-free completion tree.

To illustrate some difficulties in designing a tableau algorithm for $\mathbf{K}_{\mathcal{ALC}}$ with constant domains, we consider here an example from [5]. Suppose that we have a completion tree $\boldsymbol{T}$ with one node g labeled with the constraint system

$$\mathcal{L}(g) = \{v : \top,\ (\Diamond_i \exists R.C) = \top\}.$$

An application of the first rule above yields an additional constraint $v : \Diamond_i \exists R.C$. By applying the fourth rule, we construct a new node g' in $\boldsymbol{T}$ with the label $\mathcal{L}(g') = \{v : \exists R.C\}$, which is then extended to

$$\mathcal{L}(g') = \{v : \exists R.C,\ (v, v') : R,\ v' : C\}$$

by an application of the third rule. Since we assume constant domains and since the variables represent domain objects, the presence of v' in $\mathcal{L}(g')$ forces us to add v' to $\mathcal{L}(g)$. This can be done by extending $\mathcal{L}(g)$ with the constraint $v' : \top$, which triggers the first rule again: now it adds $v' : \Diamond_i \exists R.C$ to $\mathcal{L}(g)$. This constraint and the fourth rule give a new node g'' with label $\mathcal{L}(g'') = \{v' : \exists R.C\}$ which is then extended by the third rule with $(v', v'') : R$ and $v'' : C$. Thus we obtain a new variable v'' which has to be added to $\mathcal{L}(g)$ and $\mathcal{L}(g')$. As these steps are to be repeated infinitely many times, the algorithm does not terminate.

A standard way to overcome termination problems of this kind is to use *blocking* (see e.g., [5, 22, 23]). For a constraint system S and a variable or an object name v, let $t_S(v)$ denote the set of all concepts C such that S contains $v : C$. A variable v is said to be *blocked* in S if there is another variable or an object name x such that $t_S(v) \subseteq t_S(x)$ and x was introduced earlier than v. For example, in the completion tree constructed above, we could regard the variable v' as blocked in $\mathcal{L}(g)$ because after the introduction of v' to $\mathcal{L}(g)$ we have $t_{\mathcal{L}(g)}(v') \subseteq t_{\mathcal{L}(g)}(v)$. No completion rule is applied to blocked variables, which yields termination. By using blocking, the tableau algorithm constructs a representation of a model rather than the model itself.

In the case of $\mathbf{K}_{\mathcal{ALC}}$, it turned out to be difficult to find a proper blocking strategy. Baader and Laux [5] experimented with several such strategies, but failed to identify a suitable one that ensures termination, soundness and completeness of the resulting algorithm. The interested reader is referred to [5] for a thorough discussion and examples.

Instead of using blocking in the classical sense, the algorithm presented in this paper pursues a more general approach. In our algorithm, variables represent *partial types* of domain objects rather than domain objects themselves. Given an $\mathcal{ALC}$-interpretation $(\Delta_\mathcal{I}, \cdot^\mathcal{I})$, by the *type* of a domain object $d \in \Delta_\mathcal{I}$ we mean the set of all concepts C such that $d \in C^\mathcal{I}$. Since this set is infinite, we restrict attention only to those concepts that are 'relevant' to constructing a model for the input formula (for more details see the next section). Blocking then becomes much easier. The completion rules have to ensure that no (partial) type t is introduced into a constraint system already containing a supertype of t. More precisely, if an application of a rule leads to an introduction of a new variable v to a completion system $\mathcal{L}(g)$ such that in the resulting completion system S' we have $t_{S'}(v) \subseteq t_{\mathcal{L}(g)}(x)$, for some variable or object name x, then the variable v *is not* introduced to $\mathcal{L}(g)$. Dealing only with types, the algorithm constructs not a model satisfying the input formula, but its representation known as a *quasimodel* [35].

We illustrate the use of quasimodels by the following example. Suppose that $\boldsymbol{T}$ is a completion tree consisting of a single node g labeled with

$$\mathcal{L}(g) = \{\Box_i D = \top, v : \Diamond_i \exists R.C, v : \Box_i \neg C, v : \Box_i D\},$$

An application of the fourth rule above generates a successor node g' of g with the label $\{v : \exists R.C, v : \neg C, v : D\}$, which is then extended by the third rule to

$$\mathcal{L}(g') = \{v : \exists R.C, v : \neg C, v : D, v' : C\}.$$

Note that in quasimodels we do not need to keep trace of how roles connect objects—in our case $(v, v') : R$—this information can be recovered in a canonical way later. The constructed completion tree represents models with a set of worlds $W = \{w, w'\}$ such that

- $\lhd_i = \{\langle w, w'\rangle\}$,
- in the interpretation $I(w)$, there are domain objects 'of type v', and
- in the interpretation $I(w')$, there are domain objects of type v and of type v'.

Let us see now how the algorithm copes with constant domains. Fix a model described by the completion tree and let d be an object in $I(w')$ of type v'. As we make the constant domain assumption, d is also an element of the domain of $I(w)$. However, in $I(w)$ this element cannot be of type v because otherwise d would satisfy $\neg C$ in $I(w')$, which is impossible, since it also satisfies C. A straightforward approach to attack this problem would be to introduce a new type to $\mathcal{L}(g)$ (thus overruling blocking). But then again we would face the problem of termination. We take a different way.

Our solution is to generate a set of *minimal partial types* in *each* constraint system $\mathcal{L}(g)$ so that every domain object in the corresponding $\mathcal{ALC}$-interpretation $I(w)$ be of exactly one of the types in the set. To this end we distinguish between two kinds of variables. A variable may be *marked* in a constraint system, which indicates that it represents a minimal (partial) type, or it may be *unmarked*, which means that the variable represents an 'ordinary' type. We illustrate the difference between marked and unmarked variables as well as the role of minimal partial types by reconsidering the example above.

According to the minimal type strategy, we have to introduce to $\mathcal{L}(g)$ a marked variable v_m together with the constraint $v_m : \top$ *before* generating the node g'. For nearly all completion rules (a notable exception is the second rule above), marked variables are treated like unmarked ones. An application of the first rule adds $v_m : \Box_i D$ to $\mathcal{L}(g)$. This constraint means that *every* domain object in the $\mathcal{ALC}$-interpretation $I(w)$ is in $(\Box_i D)^{I(w)}$. After that we construct the node g' and the variable v' as above. In models described by the resulting completion tree, domain objects may be of types v and v' in $I(w')$ and of types v and v_m in $I(w)$. Again, we face the problem of finding a 'predecessor type' for v', i.e., a type for objects in $I(w)$ which are of type v' in $I(w')$. According to the minimal type strategy, we must choose this predecessor among the marked variables in $\mathcal{L}(g)$, in our case this can only be v_m. However, since the constraint $v_m : \Box_i D$ is in $\mathcal{L}(g)$ and v_m was chosen as the predecessor type for v', we must add $v' : D$ to $\mathcal{L}(g')$. Figure 1 shows the resulting completion tree. Note that with the minimal type strategy, there is no need to reconsider constraint systems that have already been treated, which helps to avoid termination problems.

To conclude this section, we give a brief overview of how the set of minimal types is generated. Consider a completion tree consisting of a node g labeled with

$$\mathcal{L}(g) = \{A = \top, B \sqcup C = \top, v : C\}$$

Again we start by introducing a single marked variable v_m together with the constraint $v_m : \top$. Applications of the first rules above add both $v_m : A$ and $v_m : B \sqcup C$. According to the second rule, we must now decide where to put v_m: to B or to C. However, it may be the case that neither of these two choices is the correct one: that all domain objects in interpretations corresponding to $\mathcal{L}(g)$ satisfy $B \sqcup C$ does not imply that all of them satisfy B or that all of them satisfy C. So for marked variables disjunction must be treated in a special way. Namely, first we introduce a new marked variable v'_m which is a 'copy' of v_m, i.e., we have $v'_m : A$ and $v'_m : B \sqcup C$ in $\mathcal{L}(g)$.

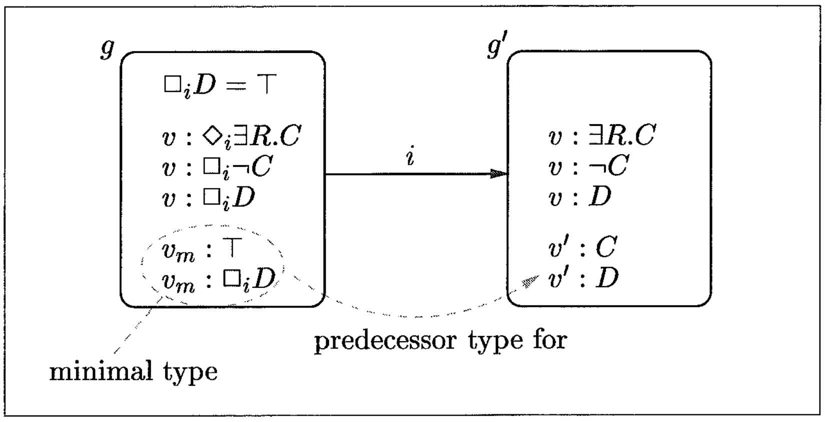

Figure 1. The fully expanded completion tree (see text).

And then we add constraints $v_m : B$ and $v'_m : C$ saying that each object is either of type v_m, and so belongs to B, or of type v'_m, and so belongs to C. To be more precise, we need a nondeterministic rule. In one case, we explore both disjuncts as has been just described; in the two additional cases, we explore only one of the disjuncts (which is necessary to deal with disjuncts that lead to a contradiction). Similar modifications are required for all nondeterministic rules dealing with marked variables.

4. Constraint systems

Let us now define formally what we mean by a constraint system for a given $\mathbf{K}_{\mathcal{ALC}}$-formula ϑ. Denote by

- $\mathsf{ob}(\vartheta)$ the set of all object names occurring in ϑ;
- $\mathsf{con}(\vartheta)$ the set of all concepts occurring in ϑ;
- $\mathsf{for}(\vartheta)$ the set of all subformulas of ϑ.

The *fragment* (of $\mathbf{K}_{\mathcal{ALC}}$) *induced by* ϑ is defined as the set

$$\mathsf{Fg}(\vartheta) = \mathsf{ob}(\vartheta) \cup \mathsf{for}(\vartheta) \cup \mathsf{con}(\vartheta) \cup \{\sim C \mid C \in \mathsf{con}(\vartheta)\} \cup \{\top\}.$$

Fix a countably infinite set V of (*individual*) *variables* so that $V \cap N_O = \emptyset$. The variables in V and the object names in $\mathsf{ob}(\vartheta)$ will be called *terms* for ϑ. We will assume that $N_O \cup V$ is well-ordered by an order $<_{N_O \cup V}$. Given a non-empty subset X of $N_O \cup V$, we denote by $\min(X)$ the first *variable* in X with respect to $<_{N_O \cup V}$. Throughout this paper, we denote variables with v and u and terms with x and y.

DEFINITION 5 (constraint system). A *constraint* for ϑ is either a formula in $\mathsf{for}(\vartheta)$ or an atom of the form $x : C$, where C is a concept in $\mathsf{Fg}(\vartheta)$ and x a term for ϑ. A *constraint system* for ϑ is a finite set S of constraints for ϑ such that

1. each variable occurring in S is either *marked* or *unmarked*;
2. $a : \top$ is in S for every $a \in \mathsf{ob}(\vartheta)$,
3. S contains at least one atom of the form $x : C$.

A variable v is called *fresh* for S if v does not occur in S.

Remark 3. Note that our constraint systems do not contain constraints of the form $(x, x') : R$. The reason for this is that our tableau algorithm constructs quasimodels which do not explicitly provide interpretations for the roles. ∎

The completion rules of the tableau algorithm are divided into two classes:

- *local rules* operate exclusively on constraint systems, while
- *global rules* operate on completion trees; they involve more than one constraint system.

The local rules are shown in Fig. 2, where the operation '+' is defined as follows.

DEFINITION 6 ('+' operation). Let S be a constraint system and Φ a set of concepts. Then

- $S + \Phi$ is S, if S contains a marked variable v for which
$$\Phi = \{E \mid v : E \in S\};$$
- $S + \Phi$ is $S \cup \{v : E \mid E \in \Phi\}$ otherwise, where v is fresh for S and marked in $S + \Phi$.

Note that we have two rules dealing with the $\sqcup$ constructor. The rule $\mathsf{R}_\sqcup$ is standard; it takes care of unmarked variables. The need for the rule $\mathsf{R}_{\sqcup'}$ operating with marked variables was explained in the previous section. Suppose $S = \{v : C \sqcup D\}$ and v is marked. This means that each domain object in models $\mathcal{I}$ described by S belongs to $(C \sqcup D)^\mathcal{I}$. The addition of $v : C$ (or $v : D$) to S, i.e., option (i) in $\mathsf{R}_{\sqcup'}$, would mean then that each object of $\mathcal{I}$ belongs to $C^\mathcal{I}$ (or, respectively, to $D^\mathcal{I}$). This excludes from consideration models $\mathcal{I}$ in which some objects are in $C^\mathcal{I}$ and some in $D^\mathcal{I}$. That is why we need option (ii) which creates a marked copy v' of v and then adds to S both $v : C$ and $v' : D$ to say that every domain object in models described by $S' = \{v : C \sqcup D, v : C, v' : D\}$ is either of type v or of type v'.

Local rules on formulas

$\mathsf{R}_\wedge$ If $(\varphi \wedge \psi) \in S$ and $\{\varphi, \psi\} \not\subseteq S$,
then set $S := S \cup \{\varphi, \psi\}$.

$\mathsf{R}_\vee$ If $(\varphi \vee \psi) \in S$ and $\{\varphi, \psi\} \cap S = \emptyset$,
then set $S := S \cup \{\theta\}$, where $\theta = \varphi$ or $\theta = \psi$.

Local non-generating rules on concepts

$\mathsf{R}_\sqcap$ If $(x : C \sqcap D) \in S$ for a term x and $\{x : C, x : D\} \not\subseteq S$,
then set $S := S \cup \{x : C, x : D\}$.

$\mathsf{R}_\sqcup$ If $(x : C \sqcup D) \in S$ for unmarked x and
$\{x : C, x : D\} \cap S = \emptyset$, then set $S := S \cup \{x : E\}$,
where $E = C$ or $E = D$.

$\mathsf{R}_{\sqcup'}$ If $(v : C \sqcup D) \in S$ for a marked variable v
and $\{v : C, v : D\} \cap S = \emptyset$, then either
(i) set $S := S \cup \{v : E\}$, where $E = C$ or $E = D$, or
(ii) set $S := (S \cup \{v : C\}) + (\{D\} \cup \{E \mid (v : E) \in S\})$.

$\mathsf{R}_=$ If $(C = \top) \in S$, a term x occurs in S, but $(x : C) \notin S$,
then set $S := S \cup \{x : C\}$.

Local generating rules

$\mathsf{R}_\neq$ If $\neg(C = \top) \in S$ and there is no term x in S such that
$(x : \sim C) \in S$, then choose a fresh variable v for S and set
$S := S \cup \{v : \sim C\}$ with unmarked v.

$\mathsf{R}_\exists$ If $(x : \exists R.C) \in S$ and there is no term y in S
such that $\{C\} \cup \{D \mid (x : \forall R.D) \in S\} \subseteq \{D \mid (y : D) \in S\}$,
then choose a fresh variable v for S and set
$S := S \cup \{v : C\} \cup \{v : D \mid (x : \forall R.D) \in S\}$ with unmarked v.

Figure 2. Local completion rules for $\mathbf{K}_{\mathcal{ALC}}$.

The majority of tableau algorithms for description logics contain two separate rules for dealing with the $\exists$ and $\forall$ constructors (see, e.g., [4, 5, 17, 23]). Our (local) set of rules contains only a single rule $\mathsf{R}_\exists$ which deals with both constructors. The reason for this is that we don't keep track of role relationships among objects.

The following definition introduces complete constraint systems and specifies what it means for a constraint system to have a clash.

DEFINITION 7 (clash). Say that a constraint system S contains a *clash* if either of the two conditions holds:

1. $\{x : A, x : \neg A\} \subseteq S$ for some term x and some concept name A,
2. $x : \neg\top$ is in S for some term x.

Otherwise we say that S is *clash-free.*

A constraint system S is *complete* if no completion rule is applicable to S.

5. Quasimodels

In this section we show how $\mathbf{K}_{\mathcal{ALC}}$-models can be represented in the form of quasimodels. Unlike [35], here we characterize quasimodels syntactically (cf. [32]).

DEFINITION 8 (quasiworld). Let ϑ be a $\mathbf{K}_{\mathcal{ALC}}$-formula. A *quasiworld* for ϑ is a complete clash-free constraint system for ϑ all variables in which are unmarked.

A frame $\mathfrak{F} = \langle W, \lhd \rangle$ whose worlds are (labeled with) quasiworlds for ϑ will be called a *ϑ-frame*. More precisely, a ϑ-frame is a triple $\mathfrak{F} = \langle W, \lhd, \sigma \rangle$, where $W \neq \emptyset$, $\lhd$ gives a binary relation $\lhd_i$ on W for every $i < \omega$, and σ is a map from W into the set of quasiworlds for ϑ.

DEFINITION 9 (run). Let $\mathfrak{F} = \langle W, \lhd, \sigma \rangle$ be a ϑ-frame. A *run* r in $\mathfrak{F}$ is a function associating with every $w \in W$ a term $r(w)$ occurring in the quasiworld $\sigma(w)$ in such a way that

- if $(r(w) : \Diamond_i C) \in \sigma(w)$ then there exists a $w' \in W$ such that $w \lhd_i w'$ and $(r(w') : C) \in \sigma(w')$;
- if $(r(w) : \Box_i C) \in \sigma(w)$ and $w \lhd_i w'$, then $(r(w') : C) \in \sigma(w')$.

DEFINITION 10 (quasimodel). A ϑ-frame $\mathfrak{F} = \langle W, \lhd, \sigma \rangle$ is called a *quasimodel* for ϑ if the following conditions hold:

1. for every object name $a \in \mathsf{ob}(\vartheta)$, the function r_a defined by $r_a(w) = a$, for $w \in W$, is a run in $\mathfrak{F}$;

2. for every $w \in W$ and every variable v in $\sigma(w)$, there exists a run r in $\mathfrak{F}$ such that $r(w) = v$;
3. for every $w \in W$ and every $\Diamond_i \varphi \in \sigma(w)$, there exists a $w' \in W$ such that $w \lhd_i w'$ and $\varphi \in \sigma(w')$;
4. for every $w \in W$ and every $\Box_i \varphi \in \sigma(w)$, if $w \lhd_i w'$ then $\varphi \in \sigma(w')$.

We say that ϑ is *quasi-satisfiable* if there is a quasimodel $\mathfrak{F} = \langle W, \lhd, \sigma \rangle$ for ϑ such that $\vartheta \in \sigma(w)$ for some $w \in W$.

THEOREM 1. *A* $\mathbf{K}_{\mathcal{ALC}}$*-formula* ϑ *in NNF is satisfiable iff it is quasi-satisfiable.*

PROOF. ($\Rightarrow$) Suppose ϑ is satisfiable. Then there is a model $\mathfrak{M} = \langle W, \lhd, I \rangle$ such that $(\mathfrak{M}, w_\vartheta) \models \vartheta$ for some $w_\vartheta \in W$. Let Δ be the domain of $\mathfrak{M}$. For all $w \in W$ and $d \in \Delta$ we put

$$\tau^{I(w)}(d) = \{C \in \mathsf{Fg}(\vartheta) \mid d \in C^{I(w)}\}.$$

Let

$$T_w = \{\tau^{I(w)}(d) \mid d \in \Delta\}.$$

For each $t = \tau^{I(w)}(d)$, take an individual variable v_t and define a constraint system $\sigma(w)$ as the union of the following sets:

- $\{\varphi \in \mathsf{for}(\vartheta) \mid (\mathfrak{M}, w) \models \varphi\}$,
- $\{a : C \mid a \in \mathsf{ob}(\vartheta),\ C \in \mathsf{Fg}(\vartheta),\ a^{I(w)} \in C^{I(w)}\}$,
- $\{v_t : C \mid C \in t\}$, for $t \in T_w$.

All variables are unmarked in $\sigma(w)$. We show that $\mathfrak{F} = \langle W, \lhd, \sigma \rangle$ is a quasimodel for ϑ.

It should be clear that the $\sigma(w)$ are quasiworlds for ϑ and that $\mathfrak{F}$ satisfies conditions (1), (3), and (4) in Definition 10. Let us check (2). Suppose that $w \in W$ and v_t is a variable from $\sigma(w)$. Take a $d \in \Delta$ with $\tau^{I(w)}(d) = t$ and define a function r with domain W by putting $r(u) = v_{\tau^{I(u)}(d)}$, for each $u \in W$. It is easy to see that r is a run in $\mathfrak{F}$ coming through v_t. That ϑ is quasi-satisfiable follows from $\vartheta \in \sigma(w_\vartheta)$.

($\Leftarrow$) Suppose that ϑ is quasi-satisfied in a world $w_\vartheta \in W$ of a quasimodel $\langle W, \lhd, \sigma \rangle$, i.e., $\vartheta \in \sigma(w_\vartheta)$.

Define a $\mathbf{K}_{\mathcal{ALC}}$-model $\mathfrak{M} = \langle W, \lhd, I \rangle$ with $I(w) = \langle \Delta, \cdot^{I(w)} \rangle$ as follows:

- Δ is the set of all runs in $\langle W, \lhd, \sigma \rangle$;
- $a^{I(w)} = r_a$, for all $a \in \mathsf{ob}(\vartheta)$;

- $A^{I(w)} = \{r \in \Delta \mid (r(w) : A) \in \sigma(w)\}$, for all concept names A in $\mathsf{Fg}(\vartheta)$;
- for every pair $r_1, r_2 \in \Delta$ and every role name R, we have $r_1 R^{I(w)} r_2$ iff

$$\{C \in \mathsf{con}(\vartheta) \mid (r_1(w) : \forall R.C) \in \sigma(w)\} \subseteq \{C \in \mathsf{con}(\vartheta) \mid (r_2(w) : C) \in \sigma(w)\}.$$

We claim that ϑ is satisfied in $\mathfrak{M}$.

Let us observe first that

(I) for all $w \in W$, $C \in \mathsf{Fg}(\vartheta)$, and $r \in \Delta$, if $(r(w) : C) \in \sigma(w)$ then $r \in C^{I(w)}$.

The proof is by induction on the construction of C. The only non-trivial steps are $C = \exists R.D$, $C = \Diamond_i D$, and $C = \Box_i D$.

Suppose $C = \exists R.D$ and $(r(w) : \exists R.D) \in \sigma(w)$. As $\sigma(w)$ is closed under the rule $\mathsf{R}_\exists$, we can find a term x such that $(x : D) \in \sigma(w)$ and

$$\{E \mid (r(w) : \forall R.E) \in \sigma(w)\} \subseteq \{E \mid (x : E) \in \sigma(w)\}.$$

Moreover, there must be a run r' such that $r'(w) = x$. But then $rR^{I(w)}r'$ and, by the induction hypothesis, $r' \in D^{I(w)}$. Therefore, $r \in (\exists R.D)^{I(w)}$.

Now suppose that $C = \Diamond_i D$ and $(r(w) : \Diamond_i D) \in \sigma(w)$. By the first clause in Definition 9, there exists $w' \in W$ such that $w \lhd_i w'$ and $(r(w') : D) \in \sigma(w')$. So, by the induction hypothesis, $r \in D^{I(w')}$, from which $r \in (\Diamond_i D)^{I(w)}$.

Finally, let $C = \Box_i D$. By the second clause of Definition 9, we then have $(r(w') : D) \in \sigma(w')$, and so $r \in D^{I(w')}$, for all $w' \in W$ such that $w \lhd_i w'$. It follows that $r \in (\Box_i D)^{I(w)}$.

Our second observation is that

(II) for every $w \in W$ and every $\varphi \in \mathsf{for}(\vartheta)$, if $\varphi \in \sigma(w)$ then $(\mathfrak{M}, w) \models \varphi$.

This is also proved by induction. Let φ be atomic and $\varphi \in \sigma(w)$. Consider two cases. First, suppose $\varphi = (a : C)$. By the first clause of Definition 10, we have $(r_a(w) : C) \in \sigma(w)$. Hence, by (I), $r_a \in C^{I(w)}$. Recall that $a^{I(w)}$ was defined as r_a. So $(\mathfrak{M}, w) \models a : C$. Second, assume $\varphi = (C = \top)$. Let $r \in \Delta$. As $\sigma(w)$ is closed under $\mathsf{R}_=$, we then have $(r(w) : C) \in \sigma(w)$. It follows from (I) that $r \in C^{I(w)}$.

Next, let $\varphi = \neg\psi$ with atomic ψ. Since ϑ is in NNF, ψ has the form $(C = \top)$. As $\sigma(w)$ is closed under $\mathsf{R}_{\neq}$, we have $(x : {\sim}C) \in \sigma(w)$ for some x. By the second clause in Definition 10, there exists a run r such that $r(w) = x$. Moreover, it follows from (I) that $r \in ({\sim}C)^{I(w)}$. So there is a $d \in \Delta$ such that $d \in ({\sim}C)^{I(w)}$, from which $(\mathfrak{M}, w) \models \neg(C = \top)$.

The induction step is rather straightforward (it is based on (3) and (4) in Definition 10); we leave the details to the reader.

It follows from (II) that $(\mathfrak{M}, w_\vartheta) \models \vartheta$. ■

6. The algorithm

We are in a position now to define completion trees, the global completion rules, and the tableau algorithm itself. Fix a countably infinite set N of *nodes*.

DEFINITION 11 (completion tree). A *completion tree* for a $\mathbf{K}_{\mathcal{ALC}}$-formula ϑ is a tree $\boldsymbol{T}$ whose nodes $g \in N$ are labeled with constraint systems $\mathcal{L}(g)$ for ϑ and whose edges (g, g') are labeled with natural numbers i such that $\Diamond_i$ or $\Box_i$ occurs in ϑ. If (g, g') is labeled with i then we say that g' is an *i-successor* of g in $\boldsymbol{T}$.

The global completion rules operate on completion trees. To introduce the rules we require the following definitions. Given a constraint system S and $i < \omega$, define an equivalence relation $\sim^i_S$ on the set of variables (not *terms*) occurring in S by taking

$$v \sim^i_S v' \quad \text{iff} \quad \{C \mid (v : \Box_i C) \in S\} = \{C \mid (v' : \Box_i C) \in S\}.$$

Denote by $[v]^i_S$ the equivalence class (with respect to $\sim^i_S$) generated by a variable v and put

$$S^i_\sim = \bigcup_{v \text{ occurs in } S} \{\min([v]^i_S)\}.$$

The global completion rules intended for constructing a completion tree for a formula ϑ are shown in Fig. 3. The $\mathsf{R}_{\Diamond f}$ rule adds successors to completion trees due to the existence of formulas $\Diamond_i \varphi$ and the $\mathsf{R}_{\Diamond c}$ rule adds successors due to the existence of constraints $a : \Diamond_i C$ and $v : \Diamond_i C$. The $\mathsf{R}_\downarrow$ and $\mathsf{R}_{\downarrow'}$ rules implement the choice of predecessor types among marked variables, where $\mathsf{R}_\downarrow$ chooses predecessor types for unmarked variables and $\mathsf{R}_{\downarrow'}$ for marked variables. Analogously to the case of $\mathsf{R}_\sqcup$ and $\mathsf{R}_{\sqcup'}$, we need two different rules since $\mathsf{R}_\downarrow$ is nondeterministic. More precisely, in the case of marked variables it is not sufficient to choose a single predecessor type, but we must consider arbitrary combinations of choices of predecessor types. The interested reader may check that, e.g., the satisfiable formula

$$(\top = \Box_i \Box_i C \sqcup \Box_i \Box_i D) \wedge (a : \Diamond_i \Diamond_i (\exists R.\neg C \sqcap \exists R.\neg D))$$

is judged unsatisfiable iff the $R_\downarrow$ rule is used for marked variables instead of the $R_{\downarrow'}$ rule.

Global generating rules

$\mathsf{R}_{\Diamond f}$ If $\Diamond_i\varphi \in \mathcal{L}(g)$ and $\varphi \notin \mathcal{L}(g')$, for all i-successors g' of g, then construct a new i-successor g' of g and set $\mathcal{L}(g')$ to the union of the following sets:

$$\{\varphi\} \quad \{\psi \mid \Box_i\psi \in \mathcal{L}(g)\} \quad \{a : \top \mid a \in \mathsf{ob}(\vartheta)\} \quad \{v : \top\}$$
$$\{a : C \mid (a : \Box_i C) \in \mathcal{L}(g)\} \quad \bigcup_{u \in (\mathcal{L}(g))^i_\sim} \{u : C \mid (u : \Box_i C) \in \mathcal{L}(g)\}$$

where v is the only marked variable in $\mathcal{L}(g')$ and $v \notin (\mathcal{L}(g))^i_\sim$.

$\mathsf{R}_{\Diamond c}$ If $(x : \Diamond_i C) \in \mathcal{L}(g)$ and for all i-successors g' of g and terms y, $\{C\} \cup \{E \mid (x : \Box_i E) \in \mathcal{L}(g)\} \not\subseteq \{E \mid (y : E) \in \mathcal{L}(g')\}$, then construct a new i-successor g' of g and set $\mathcal{L}(g')$ to the union of the following sets:

$$\{v' : C\} \quad \{\psi \mid \Box_i\psi \in \mathcal{L}(g)\} \quad \{v' : D \mid (x : \Box_i D) \in \mathcal{L}(g)\}$$
$$\{a : \top \mid a \in \mathsf{ob}(\vartheta)\} \quad \{v : \top\}$$
$$\{a : C \mid (a : \Box_i C) \in \mathcal{L}(g)\} \quad \bigcup_{u \in (\mathcal{L}(g))^i_\sim} \{u : C \mid (u : \Box_i C) \in \mathcal{L}(g)\}$$

where v is the only marked variable in $\mathcal{L}(g')$, $v \neq v'$, and $v, v' \notin (\mathcal{L}(g))^i_\sim$.

Global non-generating rules

$\mathsf{R}_\downarrow$ If g' is an i-successor of g, v an unmarked variable in $\mathcal{L}(g')$, and for no term x in $\mathcal{L}(g)$ we have

$$\{C \mid (x : \Box_i C) \in \mathcal{L}(g)\} \subseteq \{\, C \mid (v : C) \in \mathcal{L}(g')\},$$

then nondeterministically choose a marked variable v' in $\mathcal{L}(g)$ and set $\mathcal{L}(g') := \mathcal{L}(g') \cup \{v : C \mid (v' : \Box_i C) \in \mathcal{L}(g)\}$.

$\mathsf{R}_{\downarrow'}$ If g' is an i-successor of g, v a marked variable in $\mathcal{L}(g')$, for no term x in $\mathcal{L}(g)$ we have

$$\{C \mid (x : \Box_i C) \in \mathcal{L}(g)\} \subseteq \{\, C \mid (v : C) \in \mathcal{L}(g')\},$$

and X is the set of marked variables occurring in $\mathcal{L}(g)$, then nondeterministically choose a non-empty subset $Y = \{v_1, \dots, v_k\}$ of X and set
$S_1 := \mathcal{L}(g') \cup \{v : D \mid (v_1 : \Box_i D) \in \mathcal{L}(g)\}$,
$S_j := S_{j-1} + (\{D \mid (v_j : \Box_i D) \in \mathcal{L}(g)\} \cup \{E \mid (v : E) \in \mathcal{L}(g')\})$,
for all $1 < j \leq k$, and $\mathcal{L}(g') := S_k$.

Figure 3. Global completion rules for $\mathbf{K}_{\mathcal{ALC}}$.

```
define procedure sat($\boldsymbol{T}$)
  if $\boldsymbol{T}$ contains a clash then
    return unsatisfiable
  if a completion rule $r$ different from $\mathsf{R}_{\Diamond f}$ and $\mathsf{R}_{\Diamond c}$ is applicable to $\boldsymbol{T}$
    then apply $r$ to $\boldsymbol{T}$
    return sat($\boldsymbol{T}$)
  if a completion rule $r$ which is either $\mathsf{R}_{\Diamond f}$ or $\mathsf{R}_{\Diamond c}$ is applicable to $\boldsymbol{T}$
    then apply $r$ to $\boldsymbol{T}$
    return sat($\boldsymbol{T}$)
  return satisfiable
```

Figure 4. The satisfiability-checking algorithm for $\mathbf{K}_{\mathcal{ALC}}$.

We say that a completion tree $\boldsymbol{T}$ contains a *clash* if there exists a node g in $\boldsymbol{T}$ such that $\mathcal{L}(g)$ contains a clash; otherwise $\boldsymbol{T}$ is called *clash-free.* $\boldsymbol{T}$ is said to be *complete* if no completion rule is applicable to $\boldsymbol{T}$.

To decide whether a given formula ϑ in negation normal form is satisfiable, we form the initial completion tree $\boldsymbol{T}_\vartheta$ consisting of a single node g_0 labeled with the initial constraint system

$$S_\vartheta = \{\vartheta\} \cup \{a : \top \mid a \in \mathsf{ob}(\vartheta)\} \cup \{v : \top\},$$

where v is a marked individual variable. After that we repeatedly apply both local and global completion rules in such a way that the $\mathsf{R}_{\Diamond f}$ and $\mathsf{R}_{\Diamond c}$ rules are applied only if no other rule is applicable. The tableau algorithm is shown in Fig. 4 in a pseudocode notation. Note that, if the rules $\mathsf{R}_\exists$, $\mathsf{R}_{\neq}$, $\mathsf{R}_{\sqcup'}$, and $\mathsf{R}_{\downarrow'}$ are applied with higher precedence than $\mathsf{R}_{\Diamond f}$ and $\mathsf{R}_{\Diamond c}$ but with lower precedence than all the remaining rules, the introduction of duplicate types is prevented. This is, however, not crucial for termination and correctness of the algorithm.

7. Termination and correctness

In this section we show that the algorithm described above terminates and that it is sound and complete.

By the *length* $|\varphi|$ of a formula φ we mean the number of symbols used to construct φ. The *modal depth* $\mathsf{md}(\varphi)$ of φ is the length of the longest chain of nested modal operators in φ (both in subformulas and subconcepts); the modal depth $\mathsf{md}(x : C)$ of a constraint $x : C$ is defined analogously. The modal depth $\mathsf{md}(S)$ of a constraint system S is the maximal modal depth of constraints in S.

The *depth* of a tree is the number of edges in its longest branch; the *outdegree* is the maximal number of immediate successors of nodes in the tree.

LEMMA 1. *Let $\boldsymbol{T}$ be a completion tree for a formula ϑ constructed by the algorithm and let g be a node in $\boldsymbol{T}$. Then the number of constraints of the form $x : C$ in $\mathcal{L}(g)$ is bounded by $2^{c|\vartheta|^2}$, where c is a constant.*

PROOF. Let us first determine an upper bound for the number of distinct terms per node label. By the definition of completion trees and constraint systems, all object names occurring in node labels are from $\mathsf{ob}(\vartheta)$. So the number of distinct object names in a label does not exceed $|\vartheta|$.

At the moment of its generation, the node g (its label, to be more precise) contains not more than $2^{2|\vartheta|}$ distinct unmarked variables and a single marked one (see the definitions of the $\mathsf{R}_{\Diamond f}$ and $\mathsf{R}_{\Diamond c}$ rules and that of $\mathsf{Fg}(\vartheta)$). Consider now the rules that can introduce new variables in $\mathcal{L}(g)$. First for the unmarked variables: $\mathsf{R}_{\neq}$ can add at most $|\vartheta|$ new variables and $\mathsf{R}_{\exists}$ at most $2^{2|\vartheta|}$ (i.e., not more than the number of distinct subsets of concepts in $\mathsf{Fg}(\vartheta)$). We now consider marked variables, which are introduced by the $\mathsf{R}_{\sqcup'}$ and $\mathsf{R}_{\downarrow'}$ rules. Define a tree T whose nodes are the marked variables in $\mathcal{L}(g)$ and whose edges are labeled with either $\mathsf{R}_{\sqcup'}$ or $\mathsf{R}_{\downarrow'}$ as follows:

- The root node is the initial marked variable in $\mathcal{L}(g)$.
- If a completion rule $r \in \{\mathsf{R}_{\sqcup'}, \mathsf{R}_{\downarrow'}\}$ is applied to a marked variable v generating new marked variables $v_1, \ldots, v_k$, then v_i is successor of v in T and the edge between v and v_i is labelled with r for $1 \leq i \leq k$.

Using the definition of $\mathsf{R}_{\sqcup'}$, $\mathsf{R}_{\downarrow'}$, and of $\mathsf{Fg}(\vartheta)$, it is not hard to see that the depth of T is bounded by $2|\vartheta|$. Moreover, each node has at most $2|\vartheta| + 2^{2|\vartheta|}$ successors: at most $2|\vartheta|$ outgoing edges labelled with $\mathsf{R}_{\sqcup'}$ and at most $2^{2|\vartheta|}$ outgoing edges labelled with $\mathsf{R}_{\downarrow'}$. Hence, the number of nodes in the tree is bounded by $(2|\vartheta| + 2^{2|\vartheta|})^{2|\vartheta|+1} - 1 \leq 2^{8|\vartheta|^2}$ which is hence the maximum number of marked variables in $\mathcal{L}(g)$.

To sum up, we can have at most $2|\vartheta| + 2 \cdot 2^{2|\vartheta|} + 2^{8|\vartheta|^2}$ distinct terms in $\mathcal{L}(g)$, and so at most

$$2|\vartheta| \cdot (2|\vartheta| + 2 \cdot 2^{2|\vartheta|} + 2^{8|\vartheta|^2})$$

distinct constraints of the form $x : C$. ■

LEMMA 2. *Let $\boldsymbol{T}$ be a completion tree for ϑ constructed by the algorithm. Then the depth of $\boldsymbol{T}$ is bounded by $|\vartheta|$ and the outdegree of $\boldsymbol{T}$ does not exceed $2^{d|\vartheta|}$, where d is a constant.*

PROOF. If g' is a successor of g in $\boldsymbol{T}$, then clearly

$$\mathsf{md}(\mathcal{L}(g')) \lneqq \mathsf{md}(\mathcal{L}(g)).$$

So the depth of $\boldsymbol{T}$ is at most $\mathsf{md}(\vartheta) \leq |\vartheta|$.

Now we compute the outdegree. Let g be a node in $\boldsymbol{T}$. Each successor of g in $\boldsymbol{T}$ is generated by an application of the $\mathsf{R}_{\Diamond f}$ rule to some formula $\Diamond_i\varphi$ or by an application of the $\mathsf{R}_{\Diamond c}$ rule to some constraint $x : \Diamond_i C$ in $\mathcal{L}(g)$. The number of applications of the $\mathsf{R}_{\Diamond f}$ rule is obviously bounded by the number of distinct formulas in $\mathcal{L}(g)$, i.e., by $|\vartheta|$. Moreover, by definition of the $\mathsf{R}_{\Diamond c}$ rule, the number of applications of this rule is bounded by $2^{2|\vartheta|}$ (i.e., the number of distinct subsets of concepts in $\mathsf{Fg}(\vartheta)$). Thus, the outdegree of $\boldsymbol{T}$ does not exceed $|\vartheta| + 2^{2|\vartheta|}$. ■

We are in a position now to prove termination.

THEOREM 2. *Having started on the initial completion tree $\boldsymbol{T}_\vartheta$, the (nondeterministic) completion algorithm terminates after at most $2^{|\vartheta|^d}$ steps, where d is a constant.*

PROOF. In view of Lemma 2, there is a constant f such that the number of nodes in each completion tree constructed by the algorithm is at most $2^{|\vartheta|^f}$. As every global generating rule adds a new node, the number of applications of such rules is bounded by the same number.

Now let us compute the number of applications of local rules on formulas. Since $|\mathsf{for}(\vartheta)| \leq |\vartheta|$, formulas are never deleted from node labels, and since each local rule on formulas introduces a new formula to a node label, there may be at most $|\vartheta|$ applications of rules of this type per node. So the total number of applications of local rules on formulas is bounded by $2^{|\vartheta|^f} \cdot |\vartheta|$.

Finally, each of the local non-generating rules on concepts, local generating rules and global non-generating rules adds a new constraint of the form $x : C$ to a constraint system $\mathcal{L}(g)$. By Lemma 1, the number of such constraints per node is bounded by $2^{c|\vartheta|^2}$ for some constant c. Thus, the number of applications of these rules per node is at most $2^{c|\vartheta|^2}$. The total number of such rule applications is then bounded by $2^{|\vartheta|^f} \cdot 2^{c|\vartheta|^2}$. ■

Theorem 2 states that the nondeterministic tableau algorithm terminates after exponentially many steps (in the length of the input formula). Together with the soundness and completeness yet to be established, this provides us with a NEXPTIME satisfiability-checking procedure for $\mathbf{K}_{\mathcal{ALC}}$-formulas. Since the satisfiability problem for $\mathbf{K}_{\mathcal{ALC}}$-formulas is NEXPTIME-complete [27], our algorithm is optimal with respect to the worst case complexity.

Let us now turn to soundness.

DEFINITION 12 (local correctness). Say that a ϑ-frame $\mathfrak{F} = \langle W, \lhd, \sigma \rangle$ is *locally correct* if it satisfies the following conditions:

(i) if $(\Diamond_i \varphi) \in \sigma(w)$ for some $w \in W$, then there exists a $w' \in W$ such that $w \lhd_i w'$ and $\varphi \in \sigma(w')$;

(ii) if $(a : \Diamond_i C) \in \sigma(w)$ for some $w \in W$ and $a \in \mathsf{ob}(\vartheta)$, then there exists a $w' \in W$ such that $w \lhd_i w'$ and $(a : C) \in \sigma(w')$;

(iii) if $(v : \Diamond_i C) \in \sigma(w)$ for some $w \in W$ and $\Psi = \{E \mid (v : \Box_i E) \in \sigma(w)\}$, then there exist a world $w' \in W$ and a term x such that $w \lhd_i w'$ and

$$\Psi \cup \{C\} \subseteq \{E \mid (x : E) \in \sigma(w')\};$$

(iv) if $w \lhd_i w'$ then:

(a) $(\Box_i \varphi) \in \sigma(w)$ implies $\varphi \in \sigma(w')$,
(b) $\{E \mid (a : \Box_i E) \in \sigma(w)\} \subseteq \{E \mid (a : E) \in \sigma(w')\}$ for all $a \in \mathsf{ob}(\vartheta)$,
(c) for each variable v in $\sigma(w)$, there exists a term x in $\sigma(w')$ such that $\{E \mid (v : \Box_i E) \in \sigma(w)\} \subseteq \{E \mid (x : E) \in \sigma(w')\}$,
(d) for each variable v in $\sigma(w')$, there exists a term x in $\sigma(w)$ such that $\{E \mid (x : \Box_i E) \in \sigma(w)\} \subseteq \{E \mid (v : E) \in \sigma(w')\}$.

It is not hard to see that each quasimodel for ϑ is also a locally correct ϑ-frame. However, the converse does not hold. Consider, for example, the ϑ-frame $\mathfrak{F} = \langle W, \lhd, \sigma \rangle$ with

- $W = \{w, w'\}$,
- $\lhd_i = \{\langle w, w' \rangle\}$,
- $\sigma(w) = \{v : \Diamond_i C, v : \Diamond_i D\}$ and $\sigma(w') = \{v : C, v' : D\}$

(it does not matter how ϑ actually looks like). $\mathfrak{F}$ is obviously locally correct, but it cannot be a quasimodel for ϑ: there exists no run r with $r(w) = v$, because whatever choice $r(w') = v$ or $r(w') = v'$ we make, the first property in the definition of runs does not hold.

However, it is not difficult to modify $\mathfrak{F}$ and convert it in a quasimodel. Indeed, construct a new ϑ-frame $\mathfrak{F}'$ by duplicating the world w' in $\mathfrak{F}$, i.e., by adding a new world w'' to $\mathfrak{F}$ in such a way that $w \lhd_i w''$ and $\sigma(w'') = \sigma(w')$. It should be clear that $\mathfrak{F}'$ is both a locally correct ϑ-frame *and* a quasimodel for ϑ. The next lemma generalizes this observation.

LEMMA 3. *A $\mathbf{K}_{\mathcal{ALC}}$-formula ϑ is quasi-satisfiable iff there exists a locally correct ϑ-frame $\mathfrak{F} = \langle W, \lhd, \sigma\rangle$ with a world $w_\vartheta \in W$ such that $\vartheta \in \sigma(w_\vartheta)$.*

PROOF. The implication ($\Rightarrow$) is trivial. Let us prove ($\Leftarrow$). As in the example above, we construct a quasimodel $\mathfrak{F}'$ satisfying ϑ by duplicating worlds in the given locally correct ϑ-frame $\mathfrak{F}$.

Let $\overline{W}$ be a set containing $2 \cdot |\mathsf{con}(\vartheta)| + 1$ 'copies' $w^{(1)}, \dots, w^{(2\cdot|\mathsf{con}(\vartheta)|+1)}$ of each world $w \in W$, i.e.,

$$\overline{W} = \{w^{(i)} \mid w \in W \text{ and } 1 \leq i \leq 2 \cdot |\mathsf{con}(\vartheta)| + 1\}.$$

By $\lfloor w^{(i)} \rfloor$ we denote the 'parent' $w \in W$ of $w^{(i)}$.

Define a *thread* $\mathbf{p}$ in $\mathfrak{F}$ as a pair of sequences

$$x_1, \dots, x_k \quad \text{and} \quad i_1, \dots, i_{k-1}$$

with $x_1, \dots, x_k \in \overline{W}$ and $i_1, \dots, i_{k-1}$ integers such that $x_1 = w_\vartheta^{(1)}$ and $\lfloor x_j \rfloor \lhd_{i_j} \lfloor x_{j+1} \rfloor$ in $\mathfrak{F}$, for $1 \leq j < k$. Denote by $\mathsf{tail}(\mathbf{p})$ the last world in $\mathbf{p}$ (i.e., x_k in our case) and, whenever $\lfloor \mathsf{tail}(\mathbf{p}) \rfloor \lhd_{i_k} \lfloor x_{k+1} \rfloor$, let $\mathbf{p}^{i_k} x_{k+1}$ be the thread

$$x_1, \dots, x_k, x_{k+1} \quad \text{and} \quad i_1, \dots, i_{k-1}, i_k.$$

Now we define a ϑ-frame $\mathfrak{F}' = \langle W', \lhd', \sigma' \rangle$ by taking

- W' to be the set of all threads in $\mathfrak{F}$,
- $\mathbf{p} \lhd'_i \mathbf{p}'$ iff $\mathbf{p}' = \mathbf{p}^i x$ for some $x \in \overline{W}$,
- $\sigma'(\mathbf{p}) = \sigma(\lfloor \mathsf{tail}(\mathbf{p}) \rfloor)$.

It is readily seen that $\lhd'_i \cap \lhd'_j = \emptyset$ whenever $i \neq j$ and that the structure $\langle W', \bigcup_{i<\omega} \lhd'_i \rangle$ is an intransitive tree.

Using the fact that $\mathfrak{F}$ is a locally correct ϑ-frame, it is straightforward to show that $\mathfrak{F}'$ is a locally correct ϑ-frame as well (the proof is left to the reader). Moreover, it is easy to see that $\mathfrak{F}'$ satisfies the following stronger version of condition (iii) in the definition of locally correct ϑ-frames:

(iii′) If $(v : \Diamond_i C) \in \sigma'(w)$, for some $w \in W'$, then there exist pairwise distinct $w_1, \dots, w_k \in W'$ such that $k = 2 \cdot |\mathsf{con}(\vartheta)| + 1$, $w \lhd'_i w_j$ for $1 \leq j \leq k$, and there are terms $x_1, \dots, x_k$ for which

$$\{E \mid (v : \Box_i E) \in \sigma'(w)\} \cup \{C\} \subseteq \{E \mid (x_j : E) \in \sigma'(w_j)\}.$$

We now show that $\mathfrak{F}'$ is a quasimodel for ϑ, i.e., that it satisfies conditions 1–4 from Definition 10. Conditions 1, 3, and 4 follow immediately from (ii), (i), and (iv) in Definition 12.

Let us prove condition 2 claiming that, for every variable v in every $\sigma'(w_0)$, $w_0 \in W'$, there is a run r coming through v. We construct r by induction. To begin with, we put $r(w_0) = v$. Now two cases are possible.

Case $\downarrow$: Suppose that $r(w')$ has been already defined and $w \lhd_i' w'$ with undefined $r(w)$. By (iv.d) in Definition 12, there is a term x such that

$$\{E \mid (x : \Box_i E) \in \sigma'(w)\} \subseteq \{E \mid (r(w') : E) \in \sigma'(w')\}.$$

Then we put $r(w) = x$. We proceed with Case $\downarrow$ till (in finitely many steps) we reach the root of $\mathfrak{F}'$. After that we switch to

Case $\uparrow$: Suppose that $r(w)$ has already been defined, but there is $w' \rhd_i' w$ with undefined $r(w')$. Let $\Diamond_{i_1} C_1, \ldots, \Diamond_{i_k} C_k$ be all distinct concepts in $\mathsf{Fg}(\vartheta)$ of the form $\Diamond_i C$ such that $(r(w) : \Diamond_{i_j} C_j) \in \sigma'(w)$, $1 \leq j \leq k$ .By definition of Fg, we have $k \leq 2 \cdot |\mathsf{con}(\vartheta)|$. For every such $\Diamond_{i_j} C_j$ we choose by (iii') a world w_j with undefined $r(w_j)$ and a term x_j such that

$$\{E \mid (v : \Box_{i_j} E) \in \sigma'(w)\} \cup \{C_j\} \subseteq \{E \mid (x_j : E) \in \sigma'(w_j)\}$$

and $w_j \neq w_l$ whenever $j \neq l$. Put $r(w_j) = x_j$. If we still have a world $w' \rhd_i' w$ with undefined $r(w')$, then we use condition (iv.c) in Definition 12, according to which there is a term x such that

$$\{E \mid (r(w) : \Box_i E) \in \sigma'(w)\} \subseteq \{E \mid (x : E) \in \sigma'(w')\}.$$

Then we set $r(w') = x$.

It should be clear from the definition that r is a run in $\mathfrak{F}'$ coming through v in $\sigma'(w)$. Thus $\mathfrak{F}'$ is a quasimodel for ϑ. That ϑ is satisfied in $\mathfrak{F}'$ follows from $\vartheta \in \sigma(w_\vartheta)$. ■

THEOREM 3 (soundness). *If there is a complete and clash-free completion tree for a* $\mathbf{K}_{\mathcal{ALC}}$*-formula* ϑ*, then* ϑ *is satisfiable.*

PROOF. Let $\boldsymbol{T}$ be a complete and clash-free completion tree for ϑ. Define a structure $\mathfrak{F} = \langle W, \lhd, \sigma \rangle$ by taking

- W to be the set of nodes in $\boldsymbol{T}$,
- $w \lhd_i w'$ iff w' is an i-successor of w in $\boldsymbol{T}$,
- $\sigma(w) = \mathsf{unmark}(\mathcal{L}(w))$,

where $\mathsf{unmark}(\mathcal{L}(w))$ is the constraint system obtained by 'unmarking' the marked variables in $\mathcal{L}(w)$. By Lemma 3 and Theorem 1, it is sufficient to show that $\mathfrak{F}$ is a locally correct ϑ-frame.

Clearly, every $\sigma(w)$, for $w \in W$, is a saturated clash-free constraint system, i.e., a quasiworld for ϑ. So $\mathfrak{F}$ is a ϑ-frame. Let us see why it is locally correct. Conditions (i)–(iii) are satisfied simply because the rules $\mathsf{R}_{\Diamond f}$ and $\mathsf{R}_{\Diamond c}$ are not applicable to $\boldsymbol{T}$ in view of its completeness.

Let $(\Box_i \varphi) \in \mathcal{L}(w)$ and $w \lhd_i w'$. Then w' has been generated by an application of a global generating rule (either $\mathsf{R}_{\Diamond f}$ or $\mathsf{R}_{\Diamond c}$). As these rules are applied only when no other rule is applicable, $\Box_i \varphi$ was already in $\mathcal{L}(w)$ by the moment of the application of that rule, and so $\varphi \in \mathcal{L}(w')$. This proves (iv.a). Conditions (iv.b) and (iv.c) are proved analogously, and (iv.d) follows from that the rules $\mathsf{R}_{\downarrow}$ and $\mathsf{R}_{\downarrow'}$ are not applicable to $\boldsymbol{T}$. ■

THEOREM 4 (completeness). *If a $\mathbf{K}_{\mathcal{ALC}}$-formula ϑ is satisfiable then, having started from $\boldsymbol{T}_\vartheta$, the satisfiability-checking algorithm for $\mathbf{K}_{\mathcal{ALC}}$ will construct a complete and clash-free completion tree for ϑ.*

PROOF. If ϑ is satisfiable then it is quasi-satisfiable, and so by Lemma 3, there are a locally correct ϑ-frame $\mathfrak{F} = \langle W, \lhd, \sigma \rangle$ and a world $w_\vartheta \in W$ such that $\vartheta \in w_\vartheta$. We use $\mathfrak{F}$ as a 'guide' for applications of the non-deterministic rules to construct a complete and clash-free completion tree for ϑ. We assume that, for every $w \in W$, $(x : \top) \in \sigma(w)$ whenever x is a variable which occurs in $\sigma(w)$. (This is achieved by adding $(x : \top)$ to $\sigma(w)$ whenever necessary. The resulting structure is still a locally correct ϑ-frame.)

Say that a completion tree $\boldsymbol{T}$ for ϑ is *$\mathfrak{F}$-compatible* if the following holds:

1. there is a map π from the set of nodes in $\boldsymbol{T}$ to W such that
   - if g' is an i-successor of g in $\boldsymbol{T}$, then $\pi(g) \lhd_i \pi(g')$ and
   - if $\varphi \in \mathcal{L}(g)$ then $\varphi \in \sigma(\pi(g))$, for every $\varphi \in \mathsf{for}(\vartheta)$;
2. for each node g in $\boldsymbol{T}$, there is a total surjective function τ_g from the set of terms in $\sigma(\pi(g))$ to the set of marked variables in $\mathcal{L}(g)$ such that if $(v : C) \in \mathcal{L}(g)$ and $\tau_g(x) = v$ then $(x : C) \in \sigma(\pi(g))$, and
3. for each node g in $\boldsymbol{T}$, there is a total function π_g from the set of unmarked terms in $\mathcal{L}(g)$ to the set of terms in $\sigma(\pi(g))$ such that if $(x : C) \in \mathcal{L}(g)$ then $(\pi_g(x) : C) \in \sigma(\pi(g))$.

Claim. If a completion tree $\boldsymbol{T}$ for ϑ is $\mathfrak{F}$-compatible and $\boldsymbol{T}'$ is the result of an application of a rule R to $\boldsymbol{T}$, then $\boldsymbol{T}'$ is $\mathfrak{F}$-compatible as well.

Proof. Let $\boldsymbol{T}$ be an $\mathfrak{F}$-compatible completion tree, g a node in $\boldsymbol{T}$, and let π, τ_g, and π_g be the functions supplied by the definition of $\mathfrak{F}$-compatibility. Consider all possible cases for R.

Suppose that the $\mathsf{R}_\wedge$ rule is applicable to a formula $\varphi \wedge \psi$ in $\mathcal{L}(g)$. Since $\boldsymbol{T}$ is $\mathfrak{F}$-compatible, $(\varphi \wedge \psi) \in \sigma(\pi(g))$ and since $\mathfrak{F}$ is a ϑ-frame, $\sigma(\pi(g))$ is saturated, which means, in particular, that $\{\varphi, \psi\} \subseteq \sigma(\pi(g))$. The application of $\mathsf{R}_\wedge$ to $\mathcal{L}(g)$ adds φ and ψ to $\mathcal{L}(g)$. Then the very same functions π, τ_g, and π_g ensure that the resulting completion tree $\boldsymbol{T}'$ is $\mathfrak{F}$-compatible.

Suppose that the $\mathsf{R}_\vee$ rule is applicable to a formula $\varphi \vee \psi$ in $\mathcal{L}(g)$. Then, as we know, $(\varphi \vee \psi) \in \sigma(\pi(g))$, and so either φ or ψ is in $\sigma(\pi(g))$. By applying the $\mathsf{R}_\vee$ rule to $\mathcal{L}(g)$ accordingly, we clearly obtain an $\mathfrak{F}$-compatible completion tree.

Suppose that the $\mathsf{R}_\sqcap$ rule is applicable to a constraint $x : C \sqcap D$ in $\mathcal{L}(g)$. Let y be a term in $\sigma(\pi(g))$ such that either $\pi_g(x) = y$ (x is unmarked in $\mathcal{L}(g)$) or $\tau_g(y) = x$ (x is marked in $\mathcal{L}(g)$). In both cases we have

$$(y : C \sqcap D) \in \sigma(\pi(g)).$$

The non-applicability of $\mathsf{R}_\sqcap$ to $\sigma(\pi(g))$ means that $\{y : C, y : D\} \subseteq \sigma(\pi(g))$. The application of the $\mathsf{R}_\sqcap$ rule adds $x : C$ and $x : D$ to $\mathcal{L}(g)$. Hence, the functions π, τ_g, and π_g are as required for the resulting completion tree $\boldsymbol{T}'$.

Suppose the $\mathsf{R}_\sqcup$ rule is applicable to a constraint $x : C \sqcup D$ in $\mathcal{L}(g)$. Then x is unmarked in $\mathcal{L}(g)$. Clearly, either $\pi_g(x) : C$ or $\pi_g(x) : D$ is in $\sigma(\pi(g))$. By applying the $\mathsf{R}_\sqcup$ rule to $\mathcal{L}(g)$ accordingly, we see that the functions π, π_g, and τ_g are as required (since x is unmarked in $\mathcal{L}(g)$, there is no term y in $\sigma(\pi(g))$ such that $\tau_g(y) = x$).

Suppose that the $\mathsf{R}_{\sqcup'}$ rule is applicable to $v : C \sqcup D$ in $\mathcal{L}(g)$. Then v is marked in $\mathcal{L}(g)$. Let Y be the set of terms y in $\sigma(\pi(g))$ for which $\tau_g(y) = v$. Since τ_g is surjective, Y is non-empty. The non-applicability of the $\mathsf{R}_\sqcup$ rule to $\sigma(\pi(g))$ (recall that all variables in $\sigma(\pi(g))$ are unmarked) means that $\{y : C, y : D\} \cap \sigma(\pi(g)) \neq \emptyset$ for each $y \in Y$. Put

$$\begin{aligned} Y_C &= \{y \in Y \mid (y : C) \in \sigma(\pi(g))\}, \\ Y_D &= Y \setminus Y_C. \end{aligned}$$

The application of $\mathsf{R}_{\sqcup'}$ adds either (i) $v : C$ or (ii) $v : D$ to $\mathcal{L}(g)$, or (iii) it creates 'a marked copy' v' of v, for which, additionally, $(v' : D) \in \mathcal{L}(g)$ holds, and then adds $v : C$ to $\mathcal{L}(g)$. If $Y_C = \emptyset$, apply the rule in such a way that $v : D$ is added. If $Y_D = \emptyset$, apply the rule so that $v : C$ is added. Otherwise we apply the rule in the third possible way. In the first two cases,

π, π_g, and τ_g are as required for the resulting completion tree $\boldsymbol{T}'$. In the third case, define

$$\tau_g'(y) = \begin{cases} v' & \text{if } y \in Y_D \\ \tau_g(y) & \text{otherwise} \end{cases}$$

and $\tau_h' = \tau_h$ for all $h \neq g$. The functions π, π_g, and τ_g' ensure that $\boldsymbol{T}'$ is $\mathfrak{F}$-compatible.

Suppose that the $\mathsf{R}_=$ rule is applicable to a formula $C = \top$ and a term x in $\mathcal{L}(g)$. Then $(C = \top) \in \sigma(\pi(g))$. Let y be a term in $\sigma(\pi(g))$ such that either $\pi_g(x) = y$ (x is unmarked in $\mathcal{L}(g)$) or $\tau_g(y) = x$ (x is marked in $\mathcal{L}(g)$). Non-applicability of $\mathsf{R}_=$ to $\sigma(\pi(g))$ means that $(y : C) \in \sigma(\pi(g))$. Hence, after the application of $\mathsf{R}_=$ (which adds $x : C$ to $\mathcal{L}(g)$), the functions π, τ_g, and π_g will be as required for the resulting completion tree $\boldsymbol{T}'$.

Suppose $\mathsf{R}_{\neq}$ is applicable to $\neg(C = \top)$ in $\mathcal{L}(g)$. Then

$$\neg(C = \top) \in \sigma(\pi(g))$$

and there is a term y in $\sigma(\pi(g))$ such that $(y : {\sim}C) \in \sigma(\pi(g))$. By applying $\mathsf{R}_{\neq}$ to $\mathcal{L}(g)$, we introduce a new (unmarked) variable x. Define π_g' as the extension of π_g to x with $\pi_g'(x) = y$, and put $\pi_h' = \pi_h$ for all $h \neq g$. The functions π, τ_g, and π_g' are then as required for the resulting completion tree $\boldsymbol{T}'$.

Let $\mathsf{R}_\exists$ be applicable to $x : \exists R.C$ in $\mathcal{L}(g)$. Let y be a term in $\sigma(\pi(g))$ such that either $\pi_g(x) = y$ (x is unmarked in $\mathcal{L}(g)$) or $\tau_g(y) = x$ (x is marked in $\mathcal{L}(g)$). In both cases, we have $(y : \exists R.C) \in \sigma(\pi(g))$ and can proceed as in the $\mathsf{R}_{\neq}$ case (note that the newly generated variable is unmarked in any case).

Now we come to the global rules and suppose that $\mathsf{R}_{\Diamond f}$ is applicable to $\Diamond_i \varphi$ in $\mathcal{L}(g)$. Let $\pi(g) = w$. Then $(\Diamond_i \varphi) \in \sigma(w)$. The rule application generates an i-successor g' of g. In view of (i) in Definition 12, there is a $w' \in W$ such that $w \lhd_i w'$ and $\varphi \in \sigma(w')$. Set $\pi(g') = w'$. It remains to define $\pi_{g'}$ and $\tau_{g'}$. The terms occurring in $\mathcal{L}(g')$ are the object names in $\mathsf{ob}(\vartheta)$, one marked variable v, and a set of unmarked variables $v_1, \ldots, v_k$. Set

1. $\pi_{g'}(a) = a$ for every $a \in \mathsf{ob}(\vartheta)$,
2. $\pi_{g'}(v_j) = \min\{x \mid \{E \mid (v_j : E) \in \mathcal{L}(g')\} \subseteq \{E \mid (x : E) \in \sigma(w')\}\}$ for $1 \leq j \leq k$, and
3. $\tau_{g'}(x) = v$ for every term x in $\sigma(w)$.

The function $\pi_{g'}$ is well-defined for all unmarked variables $v_1, \ldots, v_k$ in $\mathcal{L}(g)$. Indeed, fix a $j \in \{1, \ldots, k\}$. By the definition of the $\mathsf{R}_{\Diamond f}$ rule, there is a variable v such that

$$\{E \mid (v : \Box_i E) \in \mathcal{L}(g)\} = \{E \mid (v_j : E) \in \mathcal{L}(g')\}.$$

By the definition of $\mathfrak{F}$-compatibility and (iv.b), (iv.c) in Definition 12, it follows that there is a term x such that

$$\{E \mid (v_j : E) \in \mathcal{L}(g')\} \subseteq \{E \mid (x : E) \in \sigma(w')\}.$$

It is easy to see that the defined functions π, τ_g, and π_g are as required. The case of $\mathsf{R}_{\Diamond c}$ is considered analogously.

Suppose that $\mathsf{R}_{\downarrow}$ is applicable to a variable v in a $\mathcal{L}(g')$ and $\pi(g') = w'$. Then v is unmarked in $\mathcal{L}(g')$ and there is a node g such that g' is i-successor of g in $\boldsymbol{T}$. Let $\pi_{g'}(v) = x$ and $\pi(g) = w$. By the definition of $\mathfrak{F}$-compatibility, we have $w \lhd_i w'$, and by (iv.d) in Definition 12, there is a term y such that

$$\{E \mid (y : \Box E) \in \sigma(w)\} \subseteq \{E \mid (x : E) \in \sigma(w')\}.$$

The rule application nondeterministically chooses a marked variable v' in $\mathcal{L}(g)$ and augments $\mathcal{L}(g')$ by $\{v : D \mid (v' : \Box_i D) \in \mathcal{L}(g)\}$. Take $v' = \tau_g(y)$ (which exists, since τ_g is total). The functions π, τ_g, and π_g are as required for the resulting completion tree $\boldsymbol{T}'$. Indeed, let $(v' : \Box_i D) \in \mathcal{L}(g)$. Since $v' = \tau_g(y)$, we have $(y : \Box_i D) \in \sigma(w)$ by the definition of $\mathfrak{F}$-compatibility. By the choice of y, $(x : D) \in \sigma(w')$ and, since $\pi_{g'}(v) = x$, we can safely add $v : D$ to $\mathcal{L}(g')$.

Finally, assume that the $\mathsf{R}_{\downarrow'}$ rule is applicable to a marked variable v_1 in $\mathcal{L}(g')$ and that $\pi(g') = w'$. Then there is a node g such that g' is an i-successor of g. Let X be the set of terms x in $\sigma(\pi(g'))$ for which $\tau_{g'}(x) = v_1$ and let $\pi(g) = w$. As $\tau_{g'}$ is surjective, $X \neq \emptyset$. For each $x \in X$, fix a term y_x such that

$$\{E \mid (y_x : \Box E) \in \sigma(w)\} \subseteq \{E \mid (x : E) \in \sigma(w')\}$$

(such terms exist by (iv.d) in Definition 12). The rule application chooses a non-empty subset Y of the marked variables in $\mathcal{L}(g)$. Let

$$Y = \{v' \mid \exists x \in X \ . \ \tau_g(y_x) = v'\}.$$

Since τ_g is total, Y is non-empty. Let $v'_1, \dots, v'_k$ be all its elements. The application of $\mathsf{R}_{\downarrow'}$ does the following:

- it generates $k-1$ 'marked copies' $v_2, \dots, v_k$ of v_1 and then
- augments $\mathcal{L}(g')$ with $\{v_j : D \mid (v'_j : \Box_i D) \in \mathcal{L}(g)\}$ for $1 \leq j \leq k$.

Put

$$\tau'_g(x) = \begin{cases} v_j & \text{if } x \in X \text{ and } \tau_g(y_x) = v'_j \\ \tau_g(x) & \text{otherwise} \end{cases}$$

and $\tau'_h = \tau_h$ if $h \neq g$. It is obvious that the functions π and π_g are as required for the resulting completion tree $\boldsymbol{T}'$ (note that $\pi_g(v_j)$ is undefined for $1 \leq j \leq k$). We show that τ'_g is also as required. Assume that the rule application added a constraint $(v_j : D)$ to $\mathcal{L}(g')$ and fix a term x such that $\tau_{g'}(x) = v_j$. Then $(v'_j : \Box_i D) \in \mathcal{L}(g)$. By the definition of X and $\tau_{g'}$, we have $\tau_g(y_x) = v'_j$, which yields $(y_x : \Box_i D) \in \sigma(w)$. By the choice of y_x, we then obtain $(x : D) \in \sigma(w')$. The claim is proved.

Now, returning to the proof of the completeness theorem, we show that it follows from the claim above. Let $\boldsymbol{T}_\vartheta$ be the initial completion tree for ϑ, g the node in $\boldsymbol{T}_\vartheta$, and v the marked variable in $\mathcal{L}(g)$. Set $\pi(g) = w_\vartheta$ (recall that we have $\vartheta \in w_\vartheta$), $\tau_g(x) = v$ for all x in $\sigma(w)$, and $\pi_g(a) = a$ for all $a \in \mathsf{ob}(\vartheta)$. It is readily checked that these functions ensure that $\boldsymbol{T}_\vartheta$ is $\mathfrak{F}$-compatible.

By the claim above, the completion rules can be applied in such a way that the resulting completion trees are $\mathfrak{F}$-compatible. According to Theorem 2, we then eventually construct a complete $\mathfrak{F}$-compatible completion tree $\boldsymbol{T}$. It remains to show that $\boldsymbol{T}$ is clash-free. Suppose otherwise. Let π, τ_g, and π_g (g a node in $\boldsymbol{T}$) be the functions supplied by the definition of $\mathfrak{F}$-compatibility. Two cases are possible.

Case 1: there is a node g in $\boldsymbol{T}$ such that $\{x : A, x : \neg A\} \subseteq \mathcal{L}(g)$ for some term x and concept name A. Suppose first that x is unmarked. Then we have $\{\pi_g(x) : A, \pi_g(x) : \neg A\} \subseteq \sigma(\pi(g))$, contrary to $\mathfrak{F}$ being a ϑ-frame. Now assume that x is marked. Since τ_g is surjective, there is a term y in $\sigma(\pi(g))$ such that $\tau_g(y) = x$. But then $\{y : A, y : \neg A\} \subseteq \sigma(\pi(g))$, which is again a contradiction.

Case 2: Clashes of the form $\neg\top \in \mathcal{L}(g)$ or $(x : \neg\top) \in \mathcal{L}(g)$ are considered analogously. ■

8. Conclusion

In this paper, we developed a tableau algorithm that is capable of reasoning with a modalized description logic under the constant domain assumption. The presented work can be extended in at least two interesting directions.

First, the considered logic $\mathbf{K}_{\mathcal{ALC}}$ is still too weak for many application areas. For adequate temporal reasoning or reasoning about knowledge and belief, we are interested in extending our results to modalized DLs with a different modal component, such as, e.g., **S4**, **KD45**, or Since/Until Logic. We conjecture that the ideas underlying our algorithm can be applied to many modalized DLs. One such extension is already available: in [26], we develop a tableau algorithm for Until Logic based on $\mathcal{ALC}$ with constant domains. To do this, it was necessary to combine the ideas of marked variables and duplication of nondeterministic rules presented in this paper with blocking in the modal component as performed in [32]. Since a combination of these two techniques succeeded and blocking in the modal component is precisely what is needed for global TBoxes, it should be clear that we can extend $\mathbf{K}_{\mathcal{ALC}}$ with global TBoxes without loosing decidability (see Section 2).

Second, it would be interesting to investigate our claim of the presented tableau algorithm to be "practicable" in more detail. A straightforward implementation of the algorithm as described in this paper cannot be expected to have an acceptable run-time behavior. However, since the algorithm in this paper differs in several aspects from standard DL tableau algorithms, it is not clear if and how the known optimizations for tableau algorithms (see, e.g., [18, 19]) can be applied. Moreover, because of some of its unusual aspects, the presented algorithm may be amenable to new optimization techniques. For example, the marked variables could be used for early clash detection: Assume that constraints involving marked variables are expanded before constraints containing unmarked terms are expanded. If, during the expansion of constraints with unmarked variables, a constraint $v : C$ for some concept C is obtained and we have $v' : \neg C$ for all marked variables v' in the constraint system, then a clash can be reported immediately. An alternative strategy could be to delay the expansion of constraints with marked variables as long as possible. This means generating the minimal types 'on demand', i.e., only if the $\mathsf{R}_{\downarrow}$ rule needs to be applied. The application of this rule itself also calls for optimization, i.e., it seems possible to develop heuristics for guiding the nondeterministic choice of variables in this rule.

Acknowledgment. The work in this paper was supported by the DFG Projects Ba1122/3-1 and Wo583/3-1 on "Combinations of Modal and Description Logics".

References

[1] Artale, A., and E. Franconi, 'A temporal description logic for reasoning about actions and plans', *Journal of Artificial Intelligence Research (JAIR)* 9, 1998.

[2] Artale, A., and C. Lutz, 'A correspondence between temporal description logics', in *Proceedings of DL-99* (eds.: P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, P. Patel-Schneider), CEUR-WS, 1999, pp. 145–149.

[3] Baader, F., M. Buchheit, and B. Hollunder, 'Cardinality restrictions on concepts', *Artificial Intelligence* 88:195–213, 1996.

[4] Baader, F., and P. Hanschke, 'A scheme for integrating concrete domains into concept languages', in *Proceedings of IJCAI-91* (eds.: J. Mylopoulos, R. Reiter), Morgan Kaufmann, pp. 452–457, 1991.

[5] BAADER, F., and A. LAUX, 'Terminological logics with modal operators', in *Proceedings of the 14th International Joint Conference on Artificial Intelligence* (ed.: C. Mellish), Morgan Kaufmann, 1995, pp. 808–814.

[6] BAADER, F., and H.-J. OHLBACH, 'A multi-dimensional terminological knowledge representation language', *Journal of Applied Non-Classical Logics* 5:153–197, 1995.

[7] CALVANESE, D., G. DE GIACOMO, and M. LENZERINI, 'Reasoning in expressive description logics with fixpoints based on automata on infinite trees', in *Proceedings of IJCAI-99* (ed.: T. Dean), 1999, pp. 84–89.

[8] DE GIACOMO, G., and M. LENZERINI, 'TBox and ABox reasoning in expressive description logics', in *Proceedings of KR-96*, Morgan Kaufmann, 1996, pp. 316–327.

[9] DEVANBU, P. T., and D. J. LITMAN, 'Taxonomic plan reasoning', *Artificial Intelligence* 84:1–35, 1996.

[10] DONINI, F. M., M. LENZERINI, D. NARDI, and A. SCHAERF, 'Adding epistemic-operators to concept languages', in *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning* (eds.: W. Nebel, B. Rich, C. Swartout), Morgan Kaufmann, 1992, pp. 342–356.

[11] DONINI, F. M., M. LENZERINI, D. NARDI, and A. SCHAERF, 'Reasoning in description logics', in *Foundation of Knowledge Representation* (ed.: G. Brewka), CSLI Publications, 1996, pp. 191–236.

[12] FAGIN, R., J. HALPERN, Y. MOSES, and M. VARDI, *Reasoning About Knowledge*, MIT Press, 1995.

[13] GABBAY, D., and V. SHEHTMAN, 'Products of modal logics, part I', *Logic Journal of the IGPL* 6:73–146, 1998.

[14] HAARSLEV, V., and R. MÖLLER, 'An empirical evaluation of optimization strategies for ABox reasoning in expressive description logics', in *Proceedings of DL-99* (eds.: P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, P. Patel-Schneider), CEUR-WS, 1999, pp. 115–119.

[15] HAARSLEV, V., and R. MÖLLER, 'RACE system description' in *Proceedings of DL-99* (eds.: P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, P. Patel-Schneider), CEUR-WS, 1999, pp. 140–141.

[16] HODKINSON, I., F. WOLTER, and M. ZAKHARYASCHEV, 'Decidable fragments of first-order temporal logics', *Annals of Pure and Applied Logic* 106:85–134, 2000.

[17] HOLLUNDER, B., and W. NUTT, 'Subsumption algorithms for concept languages', DFKI Research Report RR-90-04, Deutsches Forschungszentrum für Künstliche Intelligenz, Kaiserslautern, 1990.

[18] HORROCKS, I., 'Optimising tableaux decision procedures for description logics', PhD thesis, University of Manchester, 1997.

[19] HORROCKS, I., 'Using an expressive description logic: fact or fiction?', in *Proceedings of KR-98* (eds.: A. Cohn, L. Schubert, S. C. Shapiro), Morgan Kaufmann, 1998, pp. 636–647.

[20] HORROCKS, I., 'FaCT and iFaCT', in *Proceedings of DL-99* (eds.: P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, P. Patel-Schneider), CEUR-WS, 1999, pp. 133–135.

[21] HORROCKS, I., P. PATEL-SCHNEIDER, and R. SEBASTIANI, 'An analysis of empirical testing for modal decision procedures', *Logic Journal of the IGPL* 8:293–323, 2000.

[22] HORROCKS, I., and U. SATTLER, 'A description logic with transitive and inverse roles and role hierarchies', *Journal of Logic and Computation* 9(3), 1999.

[23] HORROCKS, I., U. SATTLER, and S. TOBIES, 'Practical reasoning for expressive description logics', in *Proceedings of LPAR-99* (eds.: H. Ganzinger, D. McAllester, A. Voronkov), Lecture Notes in Artificial Intelligence 1705, Springer-Verlag, 1999, pp. 161–180.

[24] HUGHES, G., and M. CRESSWELL, *A New Introduction to Modal Logic*, Methuen, London, 1996.

[25] HUSTADT, U., and R. A. SCHMIDT, 'Issues of decidability for description logics in the framework of resolution', in *Automated Deduction in Classical and Non-classical Logic* (eds.: R. Caterra, G. Salzer), Lecture Notes in Artificial Intelligence 1761, Springer-Verlag, 1996, pp. 191–205.

[26] LUTZ, C., H. STURM, F. WOLTER, and M. ZAKHARYASCHEV, 'Tableaux for temporal description logic with constant domain', in *Proceedings of IJCAR-2001* (eds.: R. Gore, A. Leitsch, T. Nipkow), Siena, Italy, 2001, pp. 121–136, LNAI no. 2083.

[27] MOSUROVIC, M., and M. ZAKHARYASCHEV, 'On the complexity of description logics with modal operators', in *Proceedings of the 2nd Panhellenic Logic Symposion* (eds.: P. Kolaitos, G. Koletos), Delphi, Greece, 1999, pp. 166–171.

[28] PATEL-SCHNEIDER, P., 'DLP', in *Proceedings of DL-99* (eds.: P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, P. Patel-Schneider), CEUR-WS, 1999, pp. 140–141.

[29] SCHILD, K. D., 'A correspondence theory for terminological logics: preliminary report', in *Proceedings of IJCAI-91*, Sydney, Australia, 1991, pp. 466–471.

[30] SCHILD, K. D., 'Combining terminological logics with tense logic', in *Progress in Artificial Intelligence – 6th Portuguese Conference on Artificial Intelligence, EPIA-93* (eds.: M. Filgueiras, L. Damas), Lecture Notes in Artificial Intelligence, Springer-Verlag, 1993, pp. 105–120.

[31] SCHMIDT-SCHAUSS, M., and G. SMOLKA, 'Attributive concept descriptions with complements', *Artificial Intelligence* 48:1–26, 1991.

[32] STURM, H., and F. WOLTER 'A tableau calculus for temporal description logic: the expanding domain case', *Journal of Logic and Computation*, to appear.

[33] VAN BENTHEM, J., 'Temporal logic', in *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 4* (eds.: D. Gabbay, C. Hogger, J. Robinson), Oxford Scientific Publishers, 1996, pp. 241–350.

[34] WOLTER, F., 'The product of converse PDL and polymodal K', *Journal of Logic and Computation* 10:223–251, 2000.

[35] WOLTER, F., and M. ZAKHARYASCHEV, 'Satisfiability problem in description logics with modal operators', in *Proceedings of KR-98* (eds.: A. G. Cohn, L. Schubert, S. C. Shapiro), Morgan Kaufmann, 1998, pp. 512–523.

[36] WOLTER, F., and M. ZAKHARYASCHEV, 'Modal description logics: modalizing roles'. *Fundamenta Informaticae* 39:411–438, 1999.

[37] WOLTER, F., and M. ZAKHARYASCHEV, 'Multi-dimensional description logics', in *Proceedings of IJCAI-99, Volume 1* (ed.: D. Thomas), Morgan Kaufmann, 1999, pp. 104–109.

[38] WOLTER, F., and M. ZAKHARYASCHEV, 'Temporalizing description logic', in *Frontiers of Combining Systems* (eds.: D. Gabbay, M. de Rijke), Studies Press/Wiley, 1999, pp. 379–402.

[39] WOLTER, F., and M. ZAKHARYASCHEV, 'Dynamic description logic', In *Advances in Modal Logic, Volume 2* (eds.: K. Segerberg, M. de Rijke, H. Wansing, M. Zakharyaschev), CSLI Publications, 2000, pp. 431 –446.

[40] WOLTER, F., and M. ZAKHARYASCHEV, 'Decidable fragments of first-order modal logics', *Journal of Symbolic Logic* 66: 1415–1438, 2001.

CARSTEN LUTZ
LuFG Theoretical Computer Science, RWTH Aachen,
Ahornstraße 55, 52074 Aachen, Germany
lutz@cs.rwth-aachen.de

HOLGER STURM and FRANK WOLTER
Institut für Informatik
Universität Leipzig,
Augustus-Platz 10-11, 04109 Leipzig, Germany
{hsturm,wolter}@informatik.uni-leipzig.de

MICHAEL ZAKHARYASCHEV
Department of Computer Science
King's College
Strand, London WC2R 2LS, U.K.
mz@dcs.kcl.ac.uk