Tableaux for temporal description logic with constant domains

Carsten Lutz,[1] Holger Sturm,[2] Frank Wolter,[3] and Michael Zakharyaschev[4]

[1] LuFG Theoretical Computer Science, RWTH Aachen, Ahornstraße 55, 52074 Aachen, Germany
[2] Fachbereich Philosophie, Universität Konstanz, 78457 Konstanz, Germany
[3] Institut für Informatik, Universität Leipzig, Augustus-Platz 10-11, 04109 Leipzig, Germany
[4] Department of Computer Science, King's College, Strand, London WC2R 2LS, U.K.

emails: lutz@cs.rwth-aachen.de, holger.sturm@uni-konstanz.de, wolter@informatik.uni-leipzig.de, mz@dcs.kcl.ac.uk

Abstract. We show how to combine the standard tableau system for the basic description logic $\mathcal{ALC}$ and Wolper's tableau calculus for propositional temporal logic PTL (with the temporal operators 'next-time' and 'until') in order to design a terminating sound and complete tableau-based satisfiability-checking algorithm for the temporal description logic $\mathsf{PTL}_{\mathcal{ALC}}$ of [20] interpreted in models with constant domains. We use the method of quasimodels [18, 16] to represent models with infinite domains, and the technique of minimal types [11] to maintain these domains constant. The combination is flexible and can be extended to more expressive description logics or even to decidable fragments of first-order temporal logics.

1 Introduction

Temporal description logics (TDLs) are knowledge representation formalisms intended for dealing with *temporal conceptual knowledge*. In other words, TDLs combine the ability of description logics (DLs) to represent and reason about conceptual knowledge with the ability of temporal logics (TLs) to reason about time. A dozen TDLs designed in the last decade (see e.g. [15, 14, 2, 20, 3, 10] and survey [1]) showed that the equation TDL = DL + TL may have different, often very complex solutions, partly because of the rich choice of DLs and TLs, but primarily because of principle difficulties in combining systems; see [7]. With rare exceptions, the work so far has been concentrated on theoretical foundations of TDLs (decidability and undecidability, computational complexity, expressive power). The investigation of 'implementable' algorithms is still at the embryo stage, especially for the TDLs with non-trivial interactions between their DL and TL components. The problem we are facing is as follows: is it possible to combine the existing implementable reasoning procedures for the interacting DL and TL components into a reasonably efficient (on 'real world problems') algorithm for their TDL hybrid? As the majority of the existing reasoning mechanisms for DLs are based on the tableau approach, a first challenging step would be to combine a tableau system for a DL with Wolper's tableaux [17] for the propositional temporal logic PTL.

The first TDL tableau system was constructed by Schild [14], who merged the basic description logic $\mathcal{ALC}$ with PTL by allowing applications of the temporal operator $\mathcal{U}$ (until) and its derivatives only to concepts. For example, he defines a concept Mortal by taking

$$\mathsf{Mortal} = \mathsf{Living\_being} \sqcap (\mathsf{Living\_being}\ \mathcal{U}\ \Box\neg\mathsf{Living\_being}),$$

where $\Box$ means 'always in the future.' The resulting language is interpreted in models based on the flow of time $\langle\mathbb{N},<\rangle$ and, for each $n \in \mathbb{N}$, specifying an $\mathcal{ALC}$-model that describes the state of the knowledge base at moment n. Schild obtains his sound, complete and terminating tableau system (for checking concept satisfiability) simply by putting together the tableau rules of $\mathcal{ALC}$ and PTL. The reason behind this 'trivial' solution is that, in Schild's logic, there is no actual interaction between the temporal operators of PTL and the constructors of $\mathcal{ALC}$; the logic is the *fusion* or *independent join* of its components.

A more sophisticated combination $\mathsf{PTL}_{\mathcal{ALC}}$ of $\mathcal{ALC}$ and PTL allowing applications of temporal and Boolean operators to both concepts and TBox axioms was constructed in [20]. Using $\mathsf{PTL}_{\mathcal{ALC}}$, one can express statements like 'in all times all living beings are mortal' or 'living beings will never die out completely:'

$$\Box(\mathsf{Living\_being} \sqsubseteq \mathsf{Mortal}), \quad \Box\Diamond\neg(\mathsf{Living\_being} = \bot),$$

where $\Diamond$ means 'some time in the future.' The degree of interaction between the DL and TL components in $\mathsf{PTL}_{\mathcal{ALC}}$ depends on the 'domain assumption' the intended models comply with. A tableau system for $\mathsf{PTL}_{\mathcal{ALC}}$ interpreted in models with *expanding* $\mathcal{ALC}$ domains (which means that when moving from earlier moments of time to later ones, the domains of $\mathcal{ALC}$-models can get larger and larger, but never shrink) was designed in [16]. The interaction between the components becomes even stronger if we consider models with constant domains, where an introduction of a domain element at moment n forces us to introduce the same element at all previous moments as well. This makes the problem of constructing tableaux for $\mathsf{PTL}_{\mathcal{ALC}}$ with constant domains considerably more difficult.

The choice of the domain assumption—*expanding, varying, decreasing,* or *constant*—depends on the knowledge to be represented. One can argue, for instance, whether the domain element representing a living being A in a model exists before A's birth or after A's death. However, in many applications such as reasoning about temporal entity relationship (ER) diagrams [2, 3], expanding domains do not suffice and must be replaced by constant ones. Apart from being appropriate for applications, the constant domain assumption is the most general case in the sense that reasoning with expanding (or varying) domains can be reduced to reasoning with constant domains (see e.g. [20]).

The main aim of this paper is to design a terminating, sound, and complete tableau system for checking satisfiability of $\mathsf{PTL}_{\mathcal{ALC}}$*-formulas in models with constant domains.*

This is achieved by

- combining (in a modular way) the standard tableaux for $\mathcal{ALC}$ with Wolper's [17] tableaux for PTL,
- using so-called *quasimodel* representations of constraint systems, and
- using so-called *minimal type representations* of domain elements introduced in subsequent states.

Quasimodels [18–20] are abstractions of models representing elements by their types and the evolution of elements in time by certain functions called runs. As was shown in [16], quasimodels make it possible to cope with $\mathsf{PTL}_{\mathcal{ALC}}$ models having infinite $\mathcal{ALC}$ domains (an example showing that $\mathsf{PTL}_{\mathcal{ALC}}$ does not have the finite domain property can be found in Section 2). The concept of 'minimal partial types' is the main new idea of this paper which is used to maintain the $\mathcal{ALC}$ domains constant.

Although the formula-satisfiability problem for $\mathsf{PTL}_{\mathcal{ALC}}$ is rather complex—as is shown in [3], it is ExpSpace-complete—we hope that the tableau system constructed in this paper will lead to a 'reasonably efficient' implementation of the $\mathsf{PTL}_{\mathcal{ALC}}$ reasoning services. However, in order to achieve an acceptable run-time behavior, it is still necessary to devise suitable optimization strategies for the algorithm. We believe that such strategies can be found, since, as shown in e.g. [9], related tableau algorithms are amenable to optimization.

It is to be noted that the developed approach can be used to design tableau algorithms for other combinations of description and modal logics (in particular, temporal epistemic logics of [6]). For instance, [11] gives a solution to the open problem of Baader and Laux [4] by constructing tableaux for their combination of the modal logic K with $\mathcal{ALC}$ interpreted in models with constant domains.

The paper is accompanied by a technical report [12] containing full proofs of all theorems.

2 Basic definitions

We begin by introducing the temporal description logic $\mathsf{PTL}_{\mathcal{ALC}}$ of [20].

Let $N_C = \{C_0, C_1, \dots\}$, $N_R = \{R_0, R_1, \dots\}$, and $N_O = \{a_0, a_1, \dots\}$ be countably infinite sets of *concept names*, *role names*, and *object names*, respectively. $\mathsf{PTL}_{\mathcal{ALC}}$*-concepts* are defined inductively: all the C_i as well as $\top$ are concepts, and if C, D are concepts and $R \in N_R$, then $C \sqcap D$, $\neg C$, $\exists R.C$, $\bigcirc C$, and $C \mathcal{U} D$ are concepts.

$\mathsf{PTL}_{\mathcal{ALC}}$*-formulas* are defined as follows: if C, D are concepts and $a, b \in N_O$, then $C = D$, $a : C$, and aRb are atomic formulas; and if φ and ψ are formulas, then so are $\neg\varphi$, $\varphi \wedge \psi$, $\bigcirc\varphi$, and $\varphi \mathcal{U} \psi$.

The intended models of $\mathsf{PTL}_{\mathcal{ALC}}$ are natural *two-dimensional* hybrids of standard models of $\mathcal{ALC}$ and PTL. More precisely, a $\mathsf{PTL}_{\mathcal{ALC}}$*-model* is a triple

$\mathfrak{M} = \langle \mathbb{N}, <, I \rangle$, where $<$ is the standard ordering of $\mathbb{N}$ and I a function associating with each $n \in \mathbb{N}$ an $\mathcal{ALC}$-model $I(n) = \left\langle \Delta, R_0^{I(n)}, \dots, C_0^{I(n)}, \dots, a_0^{I(n)}, \dots \right\rangle$, in which Δ, the (constant) *domain* of $\mathfrak{M}$, is a non-empty set, the $R_i^{I(n)}$ are binary relations on Δ, the $C_i^{I(n)}$ subsets of Δ, and the $a_i^{I(n)}$ are elements of Δ such that $a_i^{I(n)} = a_i^{I(m)}$, for every $n, m \in \mathbb{N}$.

(Note that in the given definition, the object names are assumed to be *global*, while the concept names are interpreted *locally*. Neither of these assumptions is essential; in particular, global concepts can be defined via local ones and $\mathcal{U}$.)

The *extension* $C^{I(n)}$ of a concept C in $\mathfrak{M}$ at a moment n is defined in the following way:

$$\top^{I(n)} = \Delta;$$
$$(C \sqcap D)^{I(n)} = C^{I(n)} \cap D^{I(n)};$$
$$(\neg C)^{I(n)} = \Delta \setminus C^{I(n)};$$
$$(\exists R.C)^{I(n)} = \{d \in \Delta \mid \exists d' \in C^{I(n)} \ dR^{I(n)}d'\};$$
$$(C\mathcal{U}D)^{I(n)} = \{d \in \Delta \mid \exists m \geq n \ (d \in D^{I(m)} \ \& \ \forall k \ (n \leq k < m \rightarrow d \in C^{I(k)}))\};$$
$$(\bigcirc C)^{I(n)} = C^{I(n+1)}.$$

The *truth-relation* $\mathfrak{M}, n \models \varphi$ for the Boolean operators is standard and

$$\mathfrak{M}, n \models C = D \text{ iff } C^{I(n)} = D^{I(n)};$$
$$\mathfrak{M}, n \models a : C \text{ iff } a^{I(n)} \in C^{I(n)};$$
$$\mathfrak{M}, n \models aRb \text{ iff } a^{I(n)} R^{I(n)} b^{I(n)};$$
$$\mathfrak{M}, n \models \varphi\mathcal{U}\psi \text{ iff } \exists m \geq n \ (\mathfrak{M}, m \models \psi \ \ \& \ \ \forall k \ (n \leq k < m \rightarrow \mathfrak{M}, k \models \varphi));$$
$$\mathfrak{M}, n \models \bigcirc\varphi \text{ iff } \mathfrak{M}, n+1 \models \varphi.$$

The only reasoning task we consider in this paper is satisfiability of $\mathsf{PTL}_{\mathcal{ALC}}$-formulas, a formula φ being *satisfiable* if there are a model $\mathfrak{M}$ and a moment $n \in \mathbb{N}$ such that $\mathfrak{M}, n \models \varphi$. Other standard inference problems for $\mathsf{PTL}_{\mathcal{ALC}}$—concept satisfiability, subsumption, ABox consistency, etc.—can be easily reduced to satisfiability of formulas.

There are two main difficulties in designing a tableau system for $\mathsf{PTL}_{\mathcal{ALC}}$. First, as was mentioned in the introduction, there exist formulas satisfiable only in models with infinite domains. For example, such is the conjunction of the formulas

$$\Box\neg\big((C \sqcap \bigcirc\neg C) = \bot\big), \quad \Box\big(\neg C \sqsubseteq \Box\neg C\big),$$

where $\Box C = \neg(\top\mathcal{U}\neg C)$ and $\bot = \neg\top$. To tackle this difficulty, we employ the standard tableaux for $\mathcal{ALC}$ for constructing *finite representations* of infinite models and keep track of the development of their elements in time by using quasimodels as introduced in [18, 20, 16].

The second difficulty is that at moment $n+1$ the $\mathcal{ALC}$ tableau algorithm can introduce an element which does not exists at moment n. To ensure that all elements always have their immediate predecessors, at each time point we create certain 'marked' elements satisfying as few conditions as possible, and use them as those predecessors if necessary.

3 Constraint systems

In this section, we introduce constraint systems which serve a two-fold purpose. First, they form a basis for defining quasimodels, which, in contrast to [20], are defined purely syntactically. Second, constraint systems are the underlying data structure of the tableau algorithm to be devised. Intuitively, a constraint system describes an $\mathcal{ALC}$-model.

In what follows, without loss of generality we assume that all equalities are of the form $C = \top$. ($C = D$ is clearly equivalent to $\big(\neg(C \sqcap \neg D) \sqcap \neg(D \sqcap \neg C)\big) = \top$.) Often we shall write $C \neq \top$ instead of $\neg(C = \top)$.

Constraint systems are formulated in the following language L_C. Let V be a fixed countably infinite set of (individual) *variables*. We assume V to be disjoint from the set N_O of object names. Elements of $V \cup N_O$ are called *L_C-terms*. If φ is a $\mathsf{PTL}_{\mathcal{ALC}}$-formula, C a concept, R a role, and x, y are L_C-terms, then φ, $x : C$, and xRy are called *L_C-formulas*.

We assume that V comes equipped with a well-order $<_V$. Let X be a non-empty subset of V. Then $\min(X)$ denotes the first variable in X with respect to $<_V$. Variables may occur in constraint systems either *marked* or *unmarked*; certain formulas may occur *$\mathcal{U}$-marked* or *$\mathcal{U}$-unmarked*. As we said above, marked variables are used to deal with constant domains. $\mathcal{U}$-markedness will be explained after the saturation rules have been introduced.

Definition 1. A *constraint system S* is a finite (non-empty) set of L_C-formulas such that

- each variable in S is either *marked* or *unmarked*,
- each formula in S of the form $\varphi \mathcal{U} \psi$ or $x : (C \mathcal{U} D)$ is either *$\mathcal{U}$-marked* or *$\mathcal{U}$-unmarked*,
- S contains $\min(V) : \top$.

We will say that a constraint system S is saturated if it satisfies a number of closure conditions. With a few exceptions, these conditions require that if S contains a formula φ of a certain form, then S contains some other formulas composed from subformulas and subconcepts of φ (possibly using additional negation and $\bigcirc$). For example, S is closed under conjunction if whenever S contains $\psi_1 \wedge \psi_2$, then it contains both conjuncts ψ_1 and ψ_2 as well. We formulate the closure conditions as the *saturation rules* in Fig. 1–3. Later these rules will also be used as rules of our tableau algorithm. A constraint system S is called *saturated* if none of the saturation rules can be applied to it.

A few remarks below will help the reader to understand the rules. As the temporal part of our tableaux is based on Wolper's [17] algorithm for PTL, the temporal saturation rules resemble those of Wolper's. Note also that the

$\mathcal{ALC}$-rules for formulas

$\underline{S \longrightarrow_{\neg\neg} \{\varphi\} \cup S}$ if
$\neg\neg\varphi \in S$ and $\varphi \notin S$

$\underline{S \longrightarrow_{\wedge} \{\varphi, \psi\} \cup S}$ if
$\varphi \wedge \psi \in S$ and $\{\varphi, \psi\} \not\subseteq S$

$\underline{S \longrightarrow_{\neg\wedge} \{\neg\theta\} \cup S}$ if
$\neg(\varphi \wedge \psi) \in S$, $\neg\varphi \notin S$, and $\neg\psi \notin S$
$\theta \doteq \varphi$ or $\theta \doteq \psi$

Temporal rules for formulas

$\underline{S \longrightarrow_{\neg\bigcirc} \{\bigcirc\neg\varphi\} \cup S}$ if
$\neg\bigcirc\varphi \in S$ and $\bigcirc\neg\varphi \notin S$

$\underline{S \longrightarrow_{\mathcal{U}} X \cup S}$ if
$\varphi\mathcal{U}\psi$ appears $\mathcal{U}$-unmarked in S
$X = \{\psi\}$ or $X = \{\varphi, \bigcirc(\varphi\mathcal{U}\psi)\}$
$\varphi\mathcal{U}\psi$ is $\mathcal{U}$-marked in $X \cup S$

$\underline{S \longrightarrow_{\neg\mathcal{U}} X \cup S}$ if
$\neg(\varphi\mathcal{U}\psi) \in S$, $\{\neg\psi, \neg\varphi\} \not\subseteq S$, and $\{\neg\psi, \bigcirc\neg(\varphi\mathcal{U}\psi)\} \not\subseteq S$
$X = \{\neg\psi, \neg\varphi\}$ or $X = \{\neg\psi, \bigcirc\neg(\varphi\mathcal{U}\psi)\}$

Fig. 1. Saturation rules for formulas.

saturation rules $\longrightarrow_{\neg\wedge}$, $\longrightarrow_{\mathcal{U}}$, $\longrightarrow_{\neg\mathcal{U}}$, $\longrightarrow_{\neg\sqcap}$, $\longrightarrow_{\mathcal{U}c}$, and $\longrightarrow_{\neg\mathcal{U}c}$ are *disjunctive*: they have more than one possible outcome. In this section, it is convenient to view these rules as nondeterministic. Later, when the saturation rules are regarded as tableau rules, we will apply them deterministically, i.e., consider *all* of their possible outcomes. Unless otherwise stated, we assume rules to introduce $\mathcal{U}$-unmarked formulas. Intuitively, $\mathcal{U}$-markedness is needed to ensure that the $\longrightarrow_{\mathcal{U}}$ and $\longrightarrow_{\mathcal{U}c}$ rules are applied exactly once to each formula $\varphi\mathcal{U}\psi$ and $x : C\mathcal{U}D$, respectively. For example, we must ensure that the $\longrightarrow_{\mathcal{U}}$ rule is applied (once) to $\varphi\mathcal{U}\psi$ even if the constraint system under consideration already contains φ and $\bigcirc(\varphi\mathcal{U}\psi)$. This is required to make the tableau algorithm complete (see [17, 16] for an example and a more detailed discussion).

As was already noted, marked variables are needed to cope with constant domains. For now, we just observe that the disjunctive rules treat marked and unmarked variables differently. Intuitively, in case of marked variables it is not sufficient to consider only one of the possible outcomes of the disjunctive rule application per constraint system, but we must additionally consider both possible outcomes together. For example, if we have $S = \{v : E\mathcal{U}F, v : \neg(C \sqcap D)\}$ and v is marked in S, then we should consider not only the obvious saturations $S_1 = S \cup \{v : \neg C\}$ and $S_2 = S \cup \{v : \neg D\}$, but also

$$S_3 = \{v : E\mathcal{U}F, v : \neg(C \sqcap D), v : \neg C, v' : E\mathcal{U}F, v' : \neg(C \sqcap D), v' : \neg D\},$$

where, v is marked in S_1, S_2, S_3 and v' is marked in S_3. In S_3, we created a 'marked copy' v' of v and saturated v in one possible way and v' in the other. In the formulation of the rules, copies are made by using $\mathrm{copy}(S, v, v')$ which denotes the set $\{v' : C \mid v : C \in S\}$, where v is marked and v' is a fresh variable

$\mathcal{ALC}$**-rules for concepts**

$\underline{S \longrightarrow_{\neg\neg c} \{x : C\} \cup S}$ if

$x : \neg\neg C \in S$ and $x : C \notin S$

$\underline{S \longrightarrow_{\sqcap} \{x : C, x : D\} \cup S}$ if

$x : C \sqcap D \in S$ and $\{x : C, x : D\} \not\subseteq S$

$\underline{S \longrightarrow_{\neg\sqcap} X \cup S}$ if

$x : \neg(C \sqcap D) \in S$, $x : \neg C \notin S$ and $x : \neg D \notin S$

$X = \{x : \neg C\}$ or $X = \{x : \neg D\}$ or
  x marked in S and $X = (\text{copy}(S, x, v) \cup \{x : \neg C, v : \neg D\})$
  where v is marked in $X \cup S$ and the first new variable from V

$\underline{S \longrightarrow_{=} \{x : C\} \cup S}$ if

$C = \top \in S$, x occurs in S, and $x : C \notin S$

$\underline{S \longrightarrow_{\neg\exists} \{y : \neg C\} \cup S}$ if

$x : \neg\exists R.C \in S$, $xRy \in S$, and $y : \neg C \notin S$

Temporal rules for concepts

$\underline{S \longrightarrow_{\neg\bigcirc c} \{x : \bigcirc\neg C\} \cup S}$ if

$x : \neg\bigcirc C \in S$ and $x : \bigcirc\neg C \notin S$

$\underline{S \longrightarrow_{\mathcal{U}c} X \cup S}$ if

$x : C\mathcal{U}D$ appears $\mathcal{U}$-unmarked in S

$X = \{x : D\}$ or $X = \{x : C, x : \bigcirc(C\mathcal{U}D)\}$ or
  x marked in S and $X = (\text{copy}(S, x, v) \cup \{x : D, v : C, v : \bigcirc(C\mathcal{U}D)\})$
  where v is marked in $X \cup S$ and the first new variable from V
$x : C\mathcal{U}D$ and $v : C\mathcal{U}D$ (if introduced) are $\mathcal{U}$-marked in $X \cup S$

$\underline{S \longrightarrow_{\neg\mathcal{U}c} X \cup S}$ if

$x : \neg(C\mathcal{U}D) \in S$, $\{x : \neg D, x : \neg C\} \not\subseteq S$, and $\{x : \neg D, x : \bigcirc\neg(C\mathcal{U}D)\} \not\subseteq S$

$X = \{x : \neg D, x : \neg C\}$ or $X = \{x : \neg D, x : \bigcirc\neg(C\mathcal{U}D)\}$ or
  x marked in S and $X = (\text{copy}(S, x, v) \cup \{x : \neg D, x : \neg C, v : \neg D, v : \bigcirc\neg(C\mathcal{U}D)\})$
  where v is marked in $X \cup S$ and the first new variable from V

Fig. 2. Non-generating saturation rules for concepts.

$\underline{S \longrightarrow_{\neq} \{v : \neg C\} \cup S}$ if

$C \neq \top \in S$ and there exists no y with $y : \neg C \in S$

v is the first new variable from V

$\underline{S \longrightarrow_{\exists} \{v : C, xRv\} \cup S}$ if

$x : \exists R.C \in S$, there is no y such that $\{xRy, y : C\} \subseteq S$ and x is not blocked in S by an unmarked variable; v is unmarked and the first new variable from V

Fig. 3. Generating saturation rules.

(not used in S). Note that by definition of L_C-formulas, marked variables do not occur in complex formulas such as $x : C \wedge x : D$ and thus such formulas need not be considered for copy. We generally assume that copies preserve $\mathcal{U}$-markedness: in the example above, $v' : E\mathcal{U}F$ is $\mathcal{U}$-marked in S_3 iff $v : E\mathcal{U}F$ is $\mathcal{U}$-marked in S.

To ensure termination of repeated applications of the saturation rules, we use a 'blocking' technique, c.f. [5]. Blocked variables are defined as follows. For now, assume that each constraint system is equipped with a strict partial order $\ll$ on the set of terms. Say that a variable v in a constraint system S is *blocked by a variable* v' in S if $v' \ll v$ and $\{C \mid v : C \in S\} \subseteq \{C \mid v' : C \in S\}$. Later, when we consider sequences of constraint systems obtained by repeated rule applications, $\ll$ will denote the order of introduction of terms. Note that only variables, rather than object names, may block terms. Also, only variables can be blocked.

A constraint system S is said to be *clash-free* if it contains no formulas $\neg\top$ and $x : \neg\top$ and neither a pair of the form $x : C$, $x : \neg C$, nor a pair of the form φ, $\neg\varphi$. We write $S \longrightarrow_\bullet S'$ to say that the constraint system S' can be obtained from S by an application of the saturation rule $\longrightarrow_\bullet$.

Let $S_0, \ldots, S_n$ be a sequence of constraint systems such that, for every $i < n$, there is a saturation rule $\longrightarrow_\bullet$ for which $S_i \longrightarrow_\bullet S_{i+1}$ and in case $\longrightarrow_\bullet$ is a generating rule, no non-generating rule is applicable to S_i (where non-generating rules are from Fig. 1 and 2 while generating rules are from Fig. 3). Then we say that $S_0, \ldots, S_n$ is built *according to the saturation strategy*. If this is the case and no saturation rule is applicable to S_n, then we call S_n a *saturation* of S_0.

4 Quasimodels

As was already said, $\mathsf{PTL}_{\mathcal{ALC}}$ does not have the finite domain property, and so our tableau algorithm constructs abstractions of models, called quasimodels, rather than models themselves.

Quasimodels are based on the idea of *concept types*. A concept type is simply a set of concepts that are 'relevant' to the tested formula and satisfied by an element of the domain. The 'fragment' of relevant concepts and formulas is defined as follows. Let Φ be a set of formulas. Denote by $Sb(\Phi)$ the set of all subformulas of formulas in Φ, by $ob(\Phi)$ the set of all object names that occur in Φ, by $rol(\Phi)$ the set of all roles in Φ, and by $con(\Phi)$ the set of all concepts in Φ. If $\#$ is a unary operator, say, $\neg$ or $\bigcirc$, then $\#(\Phi)$ is the union of Φ and $\{\#\varphi \mid \varphi \in \Phi\}$. The *fragment $Fg(\Phi)$ generated by Φ* is defined as the union of the following four sets: $ob(\Phi)$, $rol(\Phi)$, $\bigcirc(\neg con(\Phi \cup \{\top\}))$ and $\bigcirc(\neg Sb(\Phi \cup \{\top\}))$.

Roughly, a quasimodel is a sequence $(S_n \mid n \in \mathbb{N})$ of saturated constraint systems that satisfies certain conditions which control interactions between the S_n and ensure that quasimodels can be reconstructed into real models. Unlike standard tableaux, where a variable usually represents an element of a model, a variable in a quasimodel represents a concept type. More precisely, if a constraint system contains a variable v, then the corresponding $\mathcal{ALC}$-models contain at least one—but potentially (infinitely) many—elements of the type represented by v. As our $\mathsf{PTL}_{\mathcal{ALC}}$-models have constant domains, we need some means to keep track of the types representing the same element at different moments of time. This can be done using a function r, called a *run*, which associates with each $n \in \mathbb{N}$ a term $r(n)$ from S_n. Thus $r(0), r(1), \dots$ are type representations of one and the same element at moments $0, 1, \dots$.

We are in a position now to give precise definitions. Fix a $\mathsf{PTL}_{\mathcal{ALC}}$-formula ϑ.

Definition 2. A *quasiworld* for ϑ is a saturated clash-free constraint system S satisfying the following conditions:

- $\{a \mid \exists C\ (a : C) \in S\} = ob(\vartheta)$,
- $con(S) \subseteq Fg(\vartheta)$ and $rol(S) \subseteq Fg(\vartheta)$,
- for every formula $\varphi \in S$, if φ is a $\mathsf{PTL}_{\mathcal{ALC}}$-formula then $\varphi \in Fg(\vartheta)$,
- all variables in S are unmarked.

One should not be confused by that all variables in quasiworlds are unmarked. Marked variables are—as we shall see later on—important for the *construction* of a quasimodel. After the construction, marked variables can simply be 'unmarked' (note that this operation preserves saturatedness of constraint systems).

Definition 3. A sequence $Q = (S_n \mid n \in \mathbb{N})$ of quasiworlds for ϑ is called a *ϑ-sequence.* A *run* in Q is a function r associating with each $n \in \mathbb{N}$ a term $r(n)$ from S_n such that

- for every $m \in \mathbb{N}$ and every concept C, if $(r(m) : \bigcirc C) \in S_m$ then we have $(r(m+1) : C) \in S_{m+1}$,
- for all $m \in \mathbb{N}$, if $(r(m) : C\mathcal{U}D) \in S_m$ then there is $k \geq m$ such that $(r(k) : D) \in S_k$ and $(r(i) : C) \in S_i$ whenever $m \leq i < k$.

Definition 4. A ϑ-sequence Q is called a *quasimodel* for ϑ if the following hold:

- for every object name a in Q, the function r_a defined by $r_a(n) = a$, for all $n \in \mathbb{N}$, is a run in Q,
- for every $n \in \mathbb{N}$ and every variable v in S_n, there is a run r in Q such that $r(n) = v$,
- for every $n \in \mathbb{N}$ and every $\bigcirc\varphi \in S_n$, we have $\varphi \in S_{n+1}$,
- for every $n \in \mathbb{N}$ and every $(\varphi\mathcal{U}\psi) \in S_n$, there is $m \geq n$ such that $\psi \in S_m$ and $\varphi \in S_k$ whenever $n \leq k < m$.

We say that ϑ is *quasi-satisfiable* if there are a quasimodel $Q = (S_n \mid n \in \mathbb{N})$ for ϑ and $n \in \mathbb{N}$ such that $\vartheta \in S_n$.

Theorem 1. *A $\mathsf{PTL}_{\mathcal{ALC}}$-formula ϑ is satisfiable iff ϑ is quasi-satisfiable.*

5 The tableau algorithm

In this section, we present a tableau algorithm for checking satisfiability of $\mathsf{PTL}_{\mathcal{ALC}}$-formulas in models with constant domains. Before going into technical details, we explain informally how quasimodels for an input formula ϑ are constructed and, in particular, how marked variables help to maintain constant domains.

Intuitively, marked variables represent so-called 'minimal types.' If a constraint system S contains marked variables $v_1, \ldots, v_k$ then *every* element of an $\mathcal{ALC}$-model corresponding to S is described by one of the v_i. It should now be clear why the disjunctive saturation rules must be applied in a special way to marked variables. Consider, for example, the $\longrightarrow_{\neg\sqcap}$ rule and assume that there is a single marked variable v_m in S and that $v_m : \neg(C \sqcap D) \in S$. In the context of minimal types, this means that every element in corresponding $\mathcal{ALC}$-models satisfies $\neg(C \sqcap D)$. From this, however, it does not follow that every element satisfies $\neg C$ or that every element satisfies $\neg D$. Hence, the $\longrightarrow_{\neg\sqcap}$ rule cannot be applied in the same way as for unmarked variables.

Here is a simple example illustrating the construction of quasimodels with minimal types. Consider the formula

$$\vartheta = \big((\neg(\bigcirc C \sqcap \bigcirc\neg C)) = \top\big) \wedge a : \bigcirc\exists R.C.$$

With this formula we associate the initial constraint system $S_\vartheta = \{\vartheta, v_m : \top\}$ containing ϑ and a single marked variable v_m. By applying saturation rules, we obtain then the constraint system $S_0 = \{a : \bigcirc\exists R.C, v_m : \bigcirc C, v'_m : \bigcirc\neg C\}$ (slightly simplified for brevity) that describes the $\mathcal{ALC}$-model for time moment 0. The constraint system for moment 1 is $\{a : \exists R.C, v_1 : C, v_2 : \neg C, v_m : \top\}$ (where v_m is the only marked variable) which can then be extended to the system $S_1 = \{a : \exists R.C, v_m : \top, v_1 : C, v_2 : \neg C, aRv, v : C\}$ by the saturation rules. Note that we introduced a new (unmarked) variable v. Every element d which is of type v at moment 1 must—according to the constant domain assumption—also exist at moment 0. But what is the type of d at that moment (i.e., the 'predecessor type' of d at 1)? By the definition of minimal types, we must only choose among marked variables. So either d is of type v_m at 0, which means that we must add $v : C$ to S_1, or d is of type v'_m at 0, and so we must add $v : \neg C$ to S_1. The former choice yields an (initial fragment of a) quasimodel, while the latter leads to a clash. For a more detailed discussion we refer the reader to [11].

We can now define the tableau algorithm. In general, tableau algorithms try to construct a (quasi)model for the input formula by repeatedly applying *tableau rules* to an appropriate data structure. Let us first introduce this data structure.

Definition 5. A *tableau* for a $\mathsf{PTL}_{\mathcal{ALC}}$-formula ϑ is a triple $\mathcal{G} = (G, \prec, l)$, where $(G, \prec)$ is a finite tree and l a labelling function associating with each $g \in G$ a constraint system $l(g)$ for ϑ such that $S_\vartheta = \{\vartheta\} \cup \{\min(V) : \top\} \cup \{a : \top \mid a \in ob(\vartheta)\}$ is associated with the root of $\mathcal{G}$, where $\min(V)$ is marked and ϑ is $\mathcal{U}$-unmarked if it is of the form $\varphi\mathcal{U}\psi$ or $x : (C\mathcal{U}D)$.

To decide whether ϑ is satisfiable, the tableau algorithm for $\mathsf{PTL}_{\mathcal{ALC}}$ goes through two phases. In the first phase, the algorithm starts with an initial tableau $\mathcal{G}_\vartheta$ and exhaustively applies the tableau rules to be defined below. Eventually we obtain a tableau $\mathcal{G}$ to which no more rule is applicable; it is called a *completion* of $\mathcal{G}_\vartheta$. In the second phase, we eliminate those parts of $\mathcal{G}$ that contain obvious contradictions or eventualities which are not realized. After that we are in a position to deliver a verdict: ϑ is satisfiable iff the resulting tableau $\mathcal{G}'$ is not empty, i.e., iff the root of $\mathcal{G}$ has not been eliminated.

Let us first concentrate on phase 1. The initial tableau $\mathcal{G}_\vartheta$ associated with ϑ is defined as $(\{g^r\}, \prec^r, l)$, where $\prec^r = \emptyset$ and $l(g^r) = S_\vartheta$. To define the tableau rules, we require a number of auxiliary notions. Let S be a constraint system and x a term occurring in S. Denote by $A_x(S)$ the set $\{C \mid (x : \bigcirc C) \in S\}$ and define an equivalence relation $\sim_S$ on the set of variables (not terms) in S by taking $v \sim_S u$ iff $A_v(S) = A_u(S)$. The equivalence class generated by v is denoted by $[v]_S$. Finally, let $[S]_\sim$ denote the set of all equivalence classes $[v]_S$.

Similar to the local blocking strategy on variables of constraint systems, we need a global blocking strategy on the nodes of tableaux. To define this kind of blocking, it is convenient to abstract from variable names.

Let S and S' be constraint systems. S' is called a *variant* of S if there is a bijective function π from the variables occurring in S onto the variables occurring in S' which respects markedness (i.e., unmarked variables are mapped to unmarked variables and marked variables to marked variables) and S' is obtained from S by replacing each variable v from S with $\pi(v)$. In this case π is called a *renaming*.

Like constraint systems, tableaux are equipped with a strict partial order $\ll$ on the set of nodes which indicates the order in which the nodes of the tableau have been introduced. The tableau rules are shown in Fig. 4. Intuitively, the $\Longrightarrow_\bigcirc$ rule generates a new time point, while the other rules infer additional knowledge about an already existing time point. For every saturation rule $\longrightarrow_s$ we have a corresponding tableau rule $\Longrightarrow_s$. The $\Longrightarrow_\downarrow$ and $\Longrightarrow_{\downarrow'}$ rules deal with constant domains and use the notion of ancestor which is defined as follows.

Let $\mathcal{G} = (G, \prec, l)$ be a tableau for ϑ. A node $g \in G$ is called a *state* if only the $\Longrightarrow_\bigcirc$ rule is applicable to g. The node g is an *ancestor* of a node $g' \in G$ if there is a sequence of nodes $g_0, \ldots, g_n$ such that $g_0 = g$, $g_n = g'$, $g_i \prec g_{i+1}$ for $i < n$, and g_0 is the only state in the sequence.

As to the $\Longrightarrow_\bigcirc$ rule, recall that variables represent types rather than elements. In view of this, when constructing the next time point, we 'merge' variables satisfying the same concepts (by using the equivalence classes). Actually, this idea is crucial for devising a terminating tableau algorithm despite the lack of the finite domain property. The $\Longrightarrow_\downarrow$ rule formalizes the choice of a predecessor type as was sketched in the example above. Since we have to *choose* a predecessor type, the rule behaves similar to a disjunctive saturation rule, which means that we must apply the rule in a different way for marked variables. That is why we need the $\Longrightarrow_{\downarrow'}$ rule: for marked variables, it considers arbitrary combinations of choices of predecessor types.

The tableau rules are applied until no further rule application is possible. To ensure termination, we must follow a certain strategy of rule applications.

Definition 6. A tableau is *complete* if no tableau rule is applicable to it. Let $\mathcal{G}_0, \ldots, \mathcal{G}_n$ be a sequence of tableaux such that the associated orders $\ll_0, \ldots, \ll_n$

$\underline{(G, \prec, l) \Longrightarrow_s (G', \prec', l')}$

if g is a leaf in G, the saturation rule $\longrightarrow_s$ is applicable to $l(g)$,
$S_1, \ldots, S_n$ are the possible outcomes of the application of $\longrightarrow_s$ to $l(g)$,
$G' = G \uplus \{g_1, \ldots, g_n\}$ and, for $1 \le i \le n$, $\prec' = \prec \cup \{(g, g_i)\}$ and $l'(g_i) = S_i$

$\underline{(G, \prec, l) \Longrightarrow_{\bigcirc} (G', \prec', l')}$

if $G' = G \uplus \{g'\}$, $\prec' = \prec \cup \{(g, g')\}$ for some leaf $g \in G$,
$l'(g')$ is the union of the following sets:

$$\{a : \top\} \cup \{a : C \mid (a : \bigcirc C) \in l(g)\}, \text{ for } a \in ob(l(g)),$$
$$\{\psi \mid \bigcirc\psi \in l(g)\},$$
$$\{\min([v]_{l(g)}) : \top\} \cup \{\min([v]_{l(g)}) : C \mid (\min([v]_{l(g)}) : \bigcirc C) \in l(g)\},$$
$$\text{for } [v]_{l(g)} \in [l(g)]_{\sim},$$
$$\{v' : \top\},$$

where v' is the only marked variable in $l(g')$,
and there is no $g'' \in G$ with $g'' \ll g$ such that $l(g'')$ is a variant of $l(g)$
(i.e., the rule is not blocked)

$\underline{(G, \prec, l) \Longrightarrow_{\downarrow} (G', \prec', l')}$

if g is a leaf in G, v is an unmarked variable in $l(g)$, g' is the ancestor of g,
for no term x in $l(g')$ do we have

$$\{C \mid (x : \bigcirc C) \in l(g')\} \subseteq \{\, C \mid (v : C) \in l(g)\},$$

$v_1, \ldots, v_n$ are the marked variables in $l(g')$, $G' = G \uplus \{g_1, \ldots, g_n\}$, and,
for $1 \le i \le n$, we have $\prec' = \prec \cup \{(g, g_i)\}$ and

$$l'(g_i) := l(g) \cup \{v : C \mid (v_i : \bigcirc C) \in l(g')\}.$$

$\underline{(G, \prec, l) \Longrightarrow_{\downarrow'} (G', \prec', l')}$

if g is a leaf in G, v is a marked variable in $l(g)$, g' is the ancestor of g,
for no term x in $l(g')$ do we have

$$\{C \mid (x : \bigcirc C) \in l(g')\} \subseteq \{\, C \mid (v : C) \in l(g)\},$$

$X = \{\min([v']_{l(g')}) \mid v' \text{ is a marked variable in } l(g')\}$,
Y_i is the ith subset of X (for some ordering),
$G' = G \uplus \{g_1, \ldots, g_{2^{|X|}}\}$, and, for $1 \le i \le 2^{|X|}$, we have $\prec' = \prec \cup \{(g, g_i)\}$ and
$l'(g_i)$ is the union of $l(g)$ and the following sets, where we assume $Y_i = \{v_1, \ldots, v_n\}$:

$$\{v : C \mid (v_1 : \bigcirc C) \in l(g)\}$$
$$\mathrm{copy}(l(g), v, v'_j) \text{ for } 1 < j \le n$$
$$\{v'_j : C \mid (v_j : \bigcirc C) \in l(g')\} \text{ for } 1 < j \le n$$

Here, all newly introduced variables v'_j are marked in $l'(g_i)$.

Note: For all rules, we assume that $l'(g) = l(g)$ for all $g \in G$. $A \uplus B$ denotes the disjoint union of A and B.

Fig. 4. Tableau rules.

describe the order of node introduction and, for every $i < n$, there is a tableau rule $\Longrightarrow_\bullet$ such that $\mathcal{G}_i \Longrightarrow_\bullet \mathcal{G}_{i+1}$ and

- if the rule is one of the generating rules $\Longrightarrow_{\neq}$ or $\Longrightarrow_{\exists}$, then no tableau rule different from $\Longrightarrow_{\neq}$, $\Longrightarrow_{\exists}$, and $\Longrightarrow_{\bigcirc}$ is applicable to $\mathcal{G}_i$,
- if the rule is $\Longrightarrow_{\bigcirc}$, then no other tableau rule is applicable to $\mathcal{G}_i$.

Then $\mathcal{G}_0, \dots, \mathcal{G}_n$ is said to be built *according to the tableau strategy.* If this is the case, $\mathcal{G}_0 = \mathcal{G}_\vartheta$, and $\mathcal{G}_n$ is complete, then $\mathcal{G}_n$ is called a *completion* of ϑ.

The following lemma claims that the tableau strategy ensures termination.

Theorem 2. *If the tableau rules are applied according to the tableau strategy, then a completion is reached after finitely many steps.*

Let us now turn to the second phase of the algorithm, i.e., to the elimination phase. We begin by defining which nodes are blocked.

Definition 7. Let $\mathcal{G} = (G, \prec, l)$ be a tableau for ϑ. A state $g \in G$ is *blocked* by a state $g' \in G$ if $g' \ll g$ and $l(g')$ is a variant of $l(g)$. We define a new relation $\overline{\prec}$ by taking $g \overline{\prec} g'$ if either $g \prec g'$, or g has a successor g'' that is blocked by g'.

An important part of the elimination process deals with so-called eventualities. An L_C-formula $\alpha \in S$ is called an *eventuality* for a constraint system S if α is either of the form $x : C\mathcal{U}D$ or of the form $\varphi\mathcal{U}\psi$. An eventuality is said to be *unmarked* if it is not of the form $v : C\mathcal{U}D$ for any marked variable v. All eventualities occurring in the tableau have to be 'realized' in the following sense.

Definition 8. Let $\mathcal{G} = (G, \prec, l)$ be a tableau for ϑ, $g \in G$, and let α be an eventuality for $l(g)$. Then α is *realized* for g in $\mathcal{G}$ if there is a sequence of unblocked nodes $g_0 \overline{\prec} g_1 \dots \overline{\prec} g_n$ in G with $g = g_0$, $n \geq 0$, such that the following holds:

(1) if α is $\varphi\mathcal{U}\psi$ then $\psi \in l(g_n)$;

(2) if α is $v : C\mathcal{U}D$, with v unmarked or marked variable, then there are variables v_i from $l(g_i)$, $i \leq n$, with $v_0 = v$, $v_1, \dots, v_n$ unmarked, $(v_n : D) \in l(g_n)$, and, for all i, $0 < i \leq n$, we have

- if g_{i-1} is a state, then $\{C \mid (v_{i-1} : \bigcirc C) \in l(g_{i-1})\} \subseteq \{C \mid (v_i : C) \in l(g_i)\}$,
- if g_{i-1} is not a state, then $\{C \mid (v_{i-1} : C) \in l(g_{i-1})\} \subseteq \{C \mid (v_i : C) \in l(g_i)\}$;

(3) if α is $a : C\mathcal{U}D$, for some object name a, then $(a : D) \in l(g_n)$.

Intuitively, the variables $v_0, \dots, v_n$ in (2) describe the same element at different moments of time. It should be clear that in a tableau representing a quasimodel, all eventualities have to be realized. Apart from removing nodes that contain clashes, to remove nodes with non-realized eventualities is the main aim of the elimination phase.

Definition 9. Let $\mathcal{G} = (G, \prec, l)$ be a tableau for ϑ. We use the following rules to eliminate points in $\mathcal{G}$:

(e_1) if $l(g)$ contains a clash, eliminate g and all its $\prec^*$-successors (where '$\prec^*$-successor' is the transitive closure of '$\prec$-successor');

(e_2) if all $\overline{\prec}$-successors of g have been eliminated, eliminate g as well;

(e_3) if $l(g)$ contains an unmarked eventuality not realized for g, eliminate g and all its $\overline{\prec}^*$-successors.[1]

The elimination procedure is as follows. Say that a tableau $\mathcal{G}_1 = (G_1, \prec_1, l_1)$ is a *subtableau* of $\mathcal{G}_2 = (G_2, \prec_2, l_2)$ if $G_2 \supseteq G_1$ and $\mathcal{G}_1$ is the restriction of $\mathcal{G}_2$ to G_1. Obviously, if $\mathcal{G}_2$ is a tableau for ϑ and G_1 contains the root of G_2, then $\mathcal{G}_1$ is a tableau for ϑ. Suppose now that $\mathcal{G} = (G, \prec, l)$ is a completion of ϑ. We construct a decreasing sequence of subtableaux $\mathcal{G} = \mathcal{G}_0, \mathcal{G}_1, \ldots$ by iteratively eliminating nodes from G according to rules (e_1)–(e_3), with (e_1) being used only at the first step. (The two other rules are used in turns.) Since we start with a finite tableau, this process stops after finitely many steps, i.e., we reach a subtableau $\mathcal{G}' = (G', \prec', l')$ of $\mathcal{G}$ to which none of the elimination rules can be applied. We say that the root of $\mathcal{G}$ is *not eliminated* iff $G' \neq \emptyset$.

Theorem 3. *A* $\mathsf{PTL}_{\mathcal{ALC}}$*-formula* ϑ *is satisfiable iff there is a completion of* ϑ *of which the root is not eliminated.*

As a consequence of Theorems 2 and 3 we obtain

Theorem 4. *There is an effective tableau procedure which, given a* $\mathsf{PTL}_{\mathcal{ALC}}$*-formula* ϑ*, decides whether* ϑ *is satisfiable.*

6 Conclusion

This paper—a continuation of the series [14, 4, 16, 11]—develops a tableau reasoning procedure for the temporal description logic $\mathsf{PTL}_{\mathcal{ALC}}$ interpreted in two-dimensional models with constant $\mathcal{ALC}$ domains. As shown in [12], the algorithm runs in double exponential time—thus paralleling the complexity of Wolper's original PTL-algorithm [17] which solves a PSPACE-complete problem using exponential time. Despite the high complexity, we believe that the devised tableau algorithm is an important first step towards the use of TDLs as KR&R tools. A prototype implementation of the described algorithm is currently underway. Based on the experiences with this implementation, possible optimization startegies will be investigated using the work in [9] as a starting point.

An important feature of the developed algorithm is that the DL component can be made considerably more expressive, provided that the extension is also supported by a reasonable tableau procedure. One idea we are working on now is to extend this component to expressive fragments of first-order logic, thereby obtaining tableau procedures for fragments of first-order temporal logic (cf. [8]) having potential applications in a growing number of fields such as specification and verification of reactive systems, model-checking, query languages for temporal databases, etc.

[1] Of course, eventualities which are marked also have to be realized. However, the fact that all unmarked eventualities in a tableau are realized implies that all other eventualities are also realized (see proofs).

Another interesting aspect of this paper is that, with minor modifications, the constructed tableaux can be used as a satisfiability checking procedure for the Cartesian product of S5 and PTL (cf. [13]), thus contributing to a new exciting field in modal logic studying the behavior of multi-dimensonal modal systems [7].

References

1. A. Artale and E. Franconi. Temporal description logics. In L. Vila *et al.* eds., *Handbook of Time and Temporal Reasoning in AI.* MIT Press, 2001. (To appear.)
2. A. Artale and E. Franconi. Temporal ER modeling with description logics. In *Proceedings of ER'99*, 1999. Springer–Verlag.
3. A. Artale, E. Franconi, M. Mosurovic, F. Wolter, and M. Zakharyaschev. Temporal description logics for conceptual modelling: expressivity and complexity. Submitted, 2001.
4. F. Baader and A. Laux. Terminological logics with modal operators. In *Proceedings of IJCAI'95*, pp. 808–814, 1995. Morgan Kaufmann.
5. F. Baader and U. Sattler. Tableau algorithms for description logics. In R. Dyckhoff, ed., *Proceedings of Tableaux 2000*, vol. 1847 of LNAI, pp. 1–18, Springer, 2000.
6. R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about Knowledge.* MIT Press, 1995.
7. D. Gabbay, A. Kurucz, F. Wolter, and M. Zakharyaschev. *Many-Dimensional Modal Logics: Theory and Applications.* Elsevier, 2001. (To appear.)
8. I. Hodkinson, F. Wolter, and M. Zakharyaschev. Decidable fragments of first-order temporal logics. *Annals of Pure and Applied Logic*, 106:85–134, 2000.
9. I. Horrocks and P. F. Patel-Schneider. Optimising description logic subsumption. *Journal of Logic and Computation*, 9(3):267–293, 1999.
10. C. Lutz. Interval-based temporal reasoning with general TBoxes. In *Proceedings of IJCAI'01*, Morgan-Kaufmann, 2001.
11. C. Lutz, H. Sturm, F. Wolter, and M. Zakharyaschev. A tableau decision algorithm for modalized $\mathcal{ALC}$ with constant domains. Submitted, 2000.
12. C. Lutz, H. Sturm, F. Wolter, and M. Zakharyaschev. A tableau calculus for temporal description logic: The constant domain case. See http://www-lti.informatik.rwth-aachen.de/Forschung/Reports.html.
13. M. Marx, Sz. Mikulas, and S. Schlobach. Tableau calculus for local cubic modal logic and its implementation. *Journal of the IGPL*, 7:755–778, 1999.
14. K. Schild. Combining terminological logics with tense logic. In *Proceedings of the 6th Portuguese Conference on AI*, pp. 105–120, Porto, 1993.
15. A. Schmiedel. A temporal terminological logic. In *Proceedings AAAI'90*, pp. 640–645, 1990.
16. H. Sturm and F. Wolter. A tableau calculus for temporal description logic: The expanding domain case. Journal of Logic and Computation, 2001. (In print.)
17. P. Wolper. The tableau method for temporal logic: An overview. *Logique et Analyse*, 28:119–152, 1985.
18. F. Wolter and M. Zakharyaschev. Satisfiability problem in description logics with modal operators. In A. Cohn, *et al.* eds., *KR'98*, pp. 512–523, 1998.
19. F. Wolter and M. Zakharyaschev. Multi-dimensional description logics. In D. Thomas, ed., *Proceedings of IJCAI'99*, pp. 104–109, 1999.
20. F. Wolter and M. Zakharyaschev. Temporalizing description logic. In D. Gabbay and M. de Rijke, eds., *Frontiers of Combining Systems 2*, pp. 379–402. Studies Press/Wiley, 2000.