A Tableau Calculus for Temporal Description Logic: The Constant Domain Case.

Carsten Lutz, Holger Sturm, Frank Wolter, and Michael Zakharyaschev

LTCS-Report 01-01

A Tableau Calculus for Temporal Description Logic: The Constant Domain Case.

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 [19] interpreted in models with constant domains. We use the method of quasimodels [17, 15] 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. [14, 13, 2, 19, 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 [16] for the propositional temporal logic PTL.

The first TDL tableau system was constructed by Schild [13], 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 [19]. 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 [15]. 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. [19]).

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 [16] 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 [17, 18, 19] 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 [15], 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.

2 Basic definitions

We begin by introducing the temporal description logic $\mathsf{PTL}_{\mathcal{ALC}}$ of [19].

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:

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

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

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

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 [17, 19, 15].

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 [19], 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 3.1 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 [16] algorithm for PTL, the temporal saturation rules resemble those of Wolper's. Note also that the 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

$\mathcal{ALC}$-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 $X = \{\neg\psi, \bigcirc\neg(\varphi\mathcal{U}\psi)\}$

Figure 1: Saturation rules for formulas.

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 [16, 15] 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 (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, $v' : E\mathcal{U}F$

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

Figure 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

Figure 3: Generating saturation rules.

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 $\{C \,|\, v : C \in S\} \subseteq \{C \,|\, v' : C \in S\}$. 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 $\longrightarrow_{\bullet}$. A constraint system S' is called a *saturation* of a constraint system S iff there exists a sequence $S_0, \ldots, S_n$ be a sequence of constraint systems such that $S = S_0$, $S' = S_n$, and, for every $i < n$, there is a saturation rule $\longrightarrow_{\bullet}$ for which $S_i \longrightarrow_{\bullet} S_{i+1}$.

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 $\{\#\varphi \,|\, \varphi \in \Phi\}$. 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 $(S_n \,|\, n \in \mathbb{N})$ 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 4.1 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 4.2 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 4.3 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 4.4 *A* $\mathsf{PTL}_{\mathcal{ALC}}$*-formula* ϑ *is satisfiable iff* ϑ *is quasi-satisfiable.*

The proof is delivered in Section A.

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 (in the following called 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 5.1 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

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

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

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

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

$v_1,\dots,v_n$ are the marked variables in $l(g')$, $G' = G \uplus \{g_1,\dots,g_n\}$, and,
for $1 \leq i \leq n$, 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

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

$X = \{\min([v']_{l(g')}) \mid v'$ is a marked variable in $l(g')\}$,
Y_i is the ith subset of X (for some ordering),
$G' = G \uplus \{g_1,\dots,g_{2^{|X|}}\}$, and, for $1 \leq i \leq 2^{|X|}$, 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 $Y_i = \{v_1,\dots,v_n\}$:

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

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.

Figure 4: Tableau rules.

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 5.2 A tableau is *complete* if no tableau rule is applicable to it. Let $\mathcal{G}_0, \dots, \mathcal{G}_n$ be a sequence of tableaux such that the associated orders $\ll_0, \dots, \ll_n$ 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 5.3 *If the tableau rules are applied according to the tableau strategy, then a completion is reached after at most $2^{2^{r(|\vartheta|)}}$ steps, where r is a polynomial function.*

The proof is delivered in Section B. Note that our algorithm is not optimal w.r.t. the worst case, i.e., it is a 2ExpTime-algorithm solving an ExpSpace-complete problem [3]. However, the same applies to Wolper's tableau algorithm for propositional temporal logic [16]: An ExpTime-algorithm solves a PSpace-complete problem. Nevertheless, Wolper's algorithm is considered very "practical", i.e., well-suited for implementation.

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 5.4 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 5.5 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 5.6 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 $\mathcal{G} = \mathcal{G}_0, \mathcal{G}_1, \dots$ 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 5.7 *A* $\mathsf{PTL}_{\mathcal{ALC}}$*-formula* ϑ *is satisfiable iff there is a completion of* ϑ *of which the root is not eliminated.*

The result will be proved in Section C.

As a consequence of Theorems 5.3 and 5.7 we obtain

Theorem 5.8 *There is an effective tableau procedure which, given a* $\mathsf{PTL}_{\mathcal{ALC}}$*-formula* ϑ*, decides whether* ϑ *is satisfiable.*

[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).

6 Conclusion

This paper—a continuation of the series [13, 4, 15, 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. The algorithm runs in double exponential time—thus paralleling the complexity of Wolper's original PTL-algorithm [16] 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.

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 $\mathsf{S5}$ and PTL (cf. [12]), thus contributing to a new exciting field in modal logic studying the behavior of multi-dimensonal modal systems [7].

A Proof of Theorem 4.4

We are going to show that a $\mathsf{PTL}_{\mathcal{ALC}}$-formula ϑ is satisfiable iff it is quasi-satisfiable.

For the direction from left to right suppose that ϑ is satisfiable. Then there is some model $\mathfrak{M} = (\mathbb{N}, <, I)$ with $\mathfrak{M}, 0 \models \vartheta$. Fix $n \in \mathbb{N}$. For each $d \in \Delta$ put $t^{I,n}(d) = \{C \in Fg(\vartheta) \mid d \in C^{I,n}\}$. Next define equivalence relations $\sim_n$ on Δ by putting $d \sim_n d'$ iff $t^{I,n}(d) = t^{I,n}(d')$. Consider the equivalence classes modulo $\sim_n$, abbreviated by $[d]_n$. Obviously, $\{[d]_n \mid d \in \Delta^n\}$ is finite. Choose for each equivalence class $[d]_n$ a variable $v_{[d]_n}$. Define mappings γ_n which map objects names a and variables $v_{[d]_n}$ to sets of domain objects $d \in \Delta$ in the obvious way, i.e.,

$$\gamma_n(x) = \begin{cases} \{a^{I,n}\} & \text{if } x \text{ is an object name } a \\ [d]_n & \text{if } x \text{ is a variable } v_{[d]_n} \end{cases}$$

The constraint system S_n corresponding to n is defined as the union of the following five sets:

- $\{\varphi \in Fg(\vartheta) \mid \mathfrak{M}, n \models \varphi\}$,
- $\{a : C \mid a \in ob(\vartheta), C \in Fg(\vartheta), a^{I,n} \in C^{I,n}\}$,
- $\{v_{[d]_n} : C \mid d \in \Delta, C \in Fg(\vartheta), d \in C^{I,n}\}$,

- $\{vRv' \mid \exists d \in \gamma(v) \text{ and } d' \in \gamma(v') \text{ with } (d, d') \in R^{I,n}\}$.

where all formulas of the form $\varphi\mathcal{U}\psi$ and $x : \varphi\mathcal{U}\psi$ are $\mathcal{U}$-marked in S_n. It is not difficult to prove that S_n is a quasiworld for ϑ, we leave this to the reader.

Now, define Q as the sequence $(S_n \mid n \in \mathbb{N})$. We need to show that Q is a quasimodel for ϑ. That ϑ is quasi-satisfiable is then an immediate consequence of the fact that $\vartheta \in S_0$. So it only remains to check that Q satisfies the conditions from Definition 4.3. The first condition is obvious. For the second, let v be some variable that occurs in S_n. By definition, there is some $d \in \Delta$ with $v = v_{[d]_n}$. For each $m \in \mathbb{N}$, we define $r_v(m) = v_{[d]_m}$. Since $\mathfrak{M}$ has constant domain, r_v is well-defined. It is easy to see that r_v is a run in Q (in the sense of Definition 4.2) and $r_v(n) = v$. This proves the second condition. The two remaining ones are proved from the definition of the quasiworlds S_n by using the semantics of $\mathcal{PTL}_{\mathcal{ALC}}$ together with the closure conditions on $Fg(\vartheta)$.

Now we turn to the direction from right to left, which forms the interesting part of the theorem. Suppose ϑ is quasi-satisfied in a quasimodel $Q = (S_n \mid n \in \mathbb{N})$. Denote by Δ the set of all runs in Q and put, for each $n \in \mathbb{N}$,

$$I(n) = (\Delta, R_0^{I,n}, \dots, C_0^{I,n}, \dots, a_0^{I,n}, \dots)$$

where

- $a^{I,n} = r_a$, for every $a \in ob(\vartheta)$,
- $C_i^{I,n} = \{r \in \Delta \mid r(n) : C_i \in S_n\}$,
- For all runs r, r' of the form r_a, r_b with $a, b \in ob(\vartheta)$ put $rR^{I,n}r'$ iff $r(n)Rr'(n) \in S_n$. Otherwise, put $rR^{I,n}r'$ iff $r(n)Rr'(n) \in S_n$ or $\{C \mid r(n) : \neg\exists R.C \in S_n\} \subseteq \{C \mid r'(n) : \neg C \in S_n\}$.

We show that $\mathfrak{M} = (\mathbb{N}, <, I)$ is as required. To this end we first prove:

(I) For every $n \in \mathbb{N}$ and every $r \in \Delta$, if $(r(n) : C) \in S_n$ then $r \in C^{I,n}$.

This is proved by induction on the construction of C. Throughout the proof we heavily exploit the fact that the S_n are saturated, hence closed under all saturation rules. Note that in the induction we make use of a measure of the complexity of concepts according to which a concept C that has not the form of a negation and its negation $\neg C$ are of the same complexity.

Fix $n \in \mathbb{N}$ and suppose $r \in \Delta$. For atomic C the claim follows from the choice of the model. Let $C \doteq \neg D$, with D atomic. Since S_n is clash-free, $(r(n) : D) \notin S_n$. Hence $r \notin D^{I,n}$, which implies $r \in C^{I,n}$. Next, consider $C \doteq (D \sqcap E)$. By the closure of S_n under the rule $\longrightarrow_\sqcap$ we obtain $(r(n) : D) \in S_n$ and $(r(n) : E) \in S_n$. From the induction hypothesis we get $r \in D^{I,n}$ and $r \in E^{I,n}$, hence, by semantics, $r \in (D \sqcap E)^{I,n}$. The cases $C \doteq \neg(D \sqcap E)$ and $C \doteq \neg\neg D$ are proved in a similar way. One just has to use the fact that S_n is closed under the rules $\longrightarrow_{\neg\sqcap}$ and $\longrightarrow_{\neg\neg c}$, respectively.

Suppose $C \doteq \exists R.D$ and $r(n) : \exists R.D \in S_n$.

Case 1. Suppose $r(n)$ is not blocked. Then the closure under the rule $\longrightarrow_{\exists}$ provides some variable v with $\{v : D, r(n)Rv\} \subseteq S_n$. We find a run $r' \in \Delta$ with $r'(n) = v$. From this we obtain $rR^{I,n}r'$, by definition, as well as $r' \in D^{I,n}$, by induction hypothesis. Hence $r \in (\exists R.D)^{I,n}$.

Case 2. Suppose $r(n)$ is blocked. We find an unblocked variable v such that $\{C \,|\, v : C \in S_n\} \supseteq \{C \,|\, r(n) : C \in S_n\}$. In particular, $v : \exists R.D \in S_n$. We obtain a variable v' with $\{v' : D, vRv'\} \subseteq S_n$. Take a run r' with $r'(n) = v'$. Then $rR^{I,n}r'$ and, by induction hypthesis, $r' \in D^{I,n}$.

Now we check the case $C \doteq \neg\exists R.D$. Suppose $rR^{I,n}r'$. By definition, we have $r(n)Rr'(n) \in S_n$ or $\{C \,|\, r(n) : \neg C \in S_n\} \supseteq \{C \,|\, r(n) : \neg\exists R.C \in S_n\}$. In both cases $r'(n) : \neg D \in S_n$. This is clear in the latter case while in the first case it follows from the non-applicability of the $\longrightarrow_{\neg\exists}$ rule. By induction hypothesis, $r' \in (\neg D)^{I,n}$. Since r' was chosen arbitrarily, it is shown that $r \in (\neg\exists R.D)^{I,n}$.

Next, let $C \doteq \bigcirc D$ and suppose $(r(n) : \bigcirc D) \in S_n$. By the first clause in Definition 4.2, $(r(n+1) : D) \in S_{n+1}$. Hence, by induction hypothesis, $r \in D^{I,n+1}$. From the latter we obtain $r \in (\bigcirc D)^{I,n}$ by semantics. Now consider $C \doteq \neg\bigcirc D$. Since S_n is saturated, we get $(r(n) : \bigcirc\neg D) \in S_n$. The rest follows from the preceding case.

Let $C \doteq (D\mathcal{U}E)$. Then, by the second clause of Definition 4.2, there is some $m \geq n$ such that: $(r(m) : E) \in S_m$ and for every $n \leq i < m$ it holds that $(r(i) : D) \in S_i$. From this the result can be easily obtained by making use of the induction hypothesis. To complete the proof, it remains to consider the case $C \doteq \neg(D\mathcal{U}E)$. So suppose $(r(n) : \neg(D\mathcal{U}E)) \in S_n$. By induction on m we show the following claim:

(A) For all $m \geq n$, $\{(r(m) : \neg E)\,, (r(m) : \neg(D\mathcal{U}E))\} \subseteq S_m$ or there is some i such that $n \leq i < m$ and $(r(i) : \neg D) \in S_i$.

For the start, let $m = n$. The claim is an immediate consequence of the assumption together with the fact that S_n is closed under the rule $\longrightarrow_{\neg\mathcal{U}c}$. For the induction step assume that the claim has already been proved for $m = k$. We distinguish two cases: First, suppose there is some i such that $n \leq i < k$ and $(r(i) : \neg D) \in S_i$. In this case the induction step follows immediately. Suppose now there is no i with this property. Then $\{(r(k) : \neg E)\,, (r(k) : \neg(D\mathcal{U}E))\} \subseteq S_k$. By the closure under the rule $\longrightarrow_{\neg\mathcal{U}c}$, one of the following holds: (i) $(r(k) : \neg D) \in S_k$ or (ii) $(r(k) : \neg\bigcirc(D\mathcal{U}E)) \in S_k$. In the first case the induction step is trivial. In the second case we get $(r(k) : \bigcirc\neg(D\mathcal{U}E)) \in S_k$, by the closure under the rule $\longrightarrow_{\neg\bigcirc c}$. By making use of the first clause of Definition 4.2, we infer $(r(k+1) : \neg(D\mathcal{U}E)) \in S_{k+1}$. The result is obtained by the closure of S_{k+1} under $\longrightarrow_{\neg\mathcal{U}c}$. This completes the induction step, and hence the proof of (A).

Now we come back to the proof of (I). It is easy to see that (A) yields:

(B) For all $m \geq n$, $r \in (\neg E)^{I,m}$ or there is some i such that $n \leq i < m$ and $r \in (\neg D)^{I,i}$,

by induction hypothesis. From the latter we infer the desired result $r \in (\neg(D\mathcal{U}E))^{I,n}$ by a simple semantical argument. Hence (I) has been shown.

In the next step we show the following claim:

(II) For every $n \in \mathbb{N}$ and every $\varphi \in Fg(\vartheta)$, if $\varphi \in S_n$ then $\mathfrak{M}, n \models \varphi$.

Again, the claim is shown by induction. Let φ be atomic and suppose $\varphi \in S_n$. We distinguish three cases: firstly, suppose there is some object name a and some concept C such that $\varphi \doteq (a : C)$. By the first clause of Definition 4.3, we obtain $(r_a(n) : C) \in S_n$. Hence, by (I), $r_a \in C^{I,n}$. Note that $a^{I,n}$ was defined as r_a. So, by semantics, $\mathfrak{M}, n \models a : C$. Secondly, suppose there is some concept C with $\varphi \doteq (C = \top)$. Fix $r \in \Delta$. Now, consider the term $r(n)$. Since S_n is closed under the rule $\longrightarrow_{=}$, we get $(r(n) : C) \in S_n$. An application of (I) yields $r \in C^{I,n}$. Finally, suppose $\varphi \doteq aRb$. $r_a R^{I,n} r_b$ follows immediately from the definition. Hence $\mathfrak{M}, n \models \neg(aRb)$.

Next, consider $\varphi \doteq \neg\psi$, with ψ atomic. Again, we distinguish three cases. The first case, where ψ has the form $a : C$, is clear. For the second case, suppose $\psi \doteq (C = \top)$. Assume $(C \neq \top) \in S_n$. That S_n is closed under the rule $\longrightarrow_{\neq}$ supplies a term v with $(v : \neg C) \in S_n$. By the second clause in Definition 4.3 we have a run r such that $r(n) = v$. Moreover, by an application of (I), it follows that $r \in (\neg C)^{I,n}$. So there is some $d \in \Delta$ with $d \in (\neg C)^{I,n}$, that is $d \notin C^{I,n}$. But from the latter we infer $\mathfrak{M}, n \models C \neq \top$. Finally, suppose $\varphi \doteq \neg(aRb)$. Then, by the definition, we do not have $r_a R^{I,n} r_b$. Hence $\mathfrak{M}, n \models aRb$.

The proof of the induction step can be left to the reader. The different clauses are all rather similar to the corresponding ones in the proof of (I). The only interesting cases are $\varphi \doteq (\psi \mathcal{U} \chi)$ and $\varphi \doteq \bigcirc\psi$ together with their negations. Here one uses the third and fourth clause from Definition 4.3, where the first and second clause from Definition 4.2 have been used in the proof of (I).

In order to complete the proof of the theorem we reason as follows: By assumption, there is some $n \in \mathbb{N}$ such that $\vartheta \in S_n$. So an application of (II) yields $\mathfrak{M}, n \models \vartheta$, which means that ϑ is satisfied in $\mathfrak{M}$.

B Proof of Theorem 5.3

In this section, we show that if the tableau rules are applied according to the tableau strategy, then a completion is reached after at most double exponentially many steps (in the length of the input formula ϑ).

By the *length* $|\varphi|$ of a formula φ we mean the number of occurences of symbols used to construct φ. We first establish an upper bound for the number of certain constraints per node label.

Lemma B.1 *Let $\mathcal{G}_0, \ldots, \mathcal{G}_n = \mathcal{G}$ be built according to the tableau strategy with $\mathcal{G}_0 = \mathcal{G}_\vartheta$. Let $l(g)$ be a node in $\mathcal{G}$. Then the number of constraints of the form $x : C$ in $l(g)$ is bounded by $2^{p(|\vartheta|)}$, for some polynomial function p.*

Proof. Let us first determine an upper bound for the number of distinct terms per node label. All object names occurring in node labels are from $ob(\vartheta)$. So the number of distinct object names in a label does not exceed $|\vartheta|$.

The root node and the labels $l(g)$ of nodes g introduced by an application of the rule $\Longrightarrow_{\bigcirc}$ do not contain more than $2^{4|\vartheta|}$ distinct unmarked variables and a single marked one—$2^{4|\vartheta|}$ is the number of distinct subsets of concepts in $Fg(\vartheta)$ and the upper bound on the number of marked variables is due to the use of equivalence classes in the $\Longrightarrow_{\bigcirc}$ rule.

We now consider the number of variables introduced in a path $\vec{g} = g_0 \prec \cdots \prec g_n$ in which g_0 is the root node or introduced by $\Longrightarrow_{\bigcirc}$ and no g_i, $i < n$, is a state. First for the marked variables, which are introduced by the rules $\Longrightarrow_{\neg\sqcap}$, $\Longrightarrow_{\mathcal{U}c}$, $\Longrightarrow_{\neg\mathcal{U}c}$, and $\Longrightarrow_{\downarrow'}$. Define a tree T whose nodes are the marked variables in $l(g_n)$ and whose edges are labeled with either $\Longrightarrow_{\neg\sqcap}$, $\Longrightarrow_{\mathcal{U}c}$, $\Longrightarrow_{\neg\mathcal{U}c}$, or $\Longrightarrow_{\downarrow'}$ as follows:

- The root node is the initial marked variable in g_0.
- If a rule $\Longrightarrow_{\bullet} \in \{\Longrightarrow_{\neg\sqcap}, \Longrightarrow_{\mathcal{U}c}, \Longrightarrow_{\neg\mathcal{U}c}, \Longrightarrow_{\downarrow'}\}$ 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 $\Longrightarrow_{\bullet}$ for $1 \leq i \leq k$.

The depth of T is bounded by $4\,|\vartheta| + 1$: By definition of the (saturation and) tableau rules, each path in T may contain at most $4\,|\vartheta|$ edges labelled with $\Longrightarrow_{\neg\sqcap}$, $\Longrightarrow_{\mathcal{U}c}$, or $\Longrightarrow_{\neg\mathcal{U}c}$ and at most a single edge labelled $\Longrightarrow_{\downarrow'}$. Moreover, each node has at most $4\,|\vartheta| + 2^{4|\vartheta|}$ successors: at most $4\,|\vartheta|$ outgoing edges labelled with $\Longrightarrow_{\neg\sqcap}$, $\Longrightarrow_{\mathcal{U}c}$, or $\Longrightarrow_{\neg\mathcal{U}c}$, and at most $2^{4|\vartheta|}$ outgoing edges labelled with $\Longrightarrow_{\downarrow'}$. Hence, the number of nodes in the tree (which is the maximum number of marked variables in $l(g_n)$) is bounded by $2^{q(|\vartheta|)}$ for some polynomial function q.

We now consider unmarked variables which are introduced by the $\Longrightarrow_{\neq}$ and $\Longrightarrow_{\exists}$ rules in the sequence $\vec{g}$. The rule $\Longrightarrow_{\neq}$ can obviously add at most $|\vartheta|$ new variables. For the $\Longrightarrow_{\exists}$ rule, we distinguish applications to constraints $x : \exists R.C$ where (i) x is an object name, (ii) x is a marked variable, and (iii) x is an unmarked variable. We may obviously have at most $|\vartheta| \cdot 4\,|\vartheta|$ rule applications of type (i). By the upper bound established for marked variables, we may have at most $2^{q(|\vartheta|)} \cdot 4\,|\vartheta|$ applications of type (ii). Now for type (iii). By the tableau strategy, a variable v will never be blocked in $\vec{g}$ after the $\Longrightarrow_{\exists}$ rule has been applied to a constraint $v : \exists R.C$. Moreover, by definition of blocking, there may exist at most $2^{4|\vartheta|}$ unblocked unmarked variables per constraint system. It follows that the $\Longrightarrow_{\exists}$ rule is applied to at most $2^{4|\vartheta|}$ distinct unmarked variables, i.e., there may be at most $4\,|\vartheta| \cdot 2^{4|\vartheta|}$ applications of type (iii). Summing up, there clearly exists a polynomial function q' such that the number of unmarked variables in $l(g_n)$ is bounded by $2^{q'(|\vartheta|)}$.

For each term x, there may obviously exist at most $4\,|\vartheta|$ constraints of the form $x : C$. Hence, if we take together the upper bounds for the numbers of objects, marked variables, and unmarked variables, Lemma B.1 immediately follows. $\dashv$

The lemma just established is helpful for determining the maximum number of constraint systems appearing during a run of the algorithm that are not variants of one another. In the following, we call two constraint systems *vdistinct* iff they are not variants of one another.

Lemma B.2 *Let $\mathcal{G}_0, \ldots, \mathcal{G}_n = \mathcal{G}$ be built according to the tableau strategy with $\mathcal{G}_0 = \mathcal{G}_\vartheta$. Then $\mathcal{G}$ contains at most $2^{2^{q(|\vartheta|)}}$ node labels that are pairwise vdistinct, where q is a polynomial function.*

Proof. Let us first determine the maximum size of node labels in $\mathcal{G}$. From Lemma B.1, we know that each label may contain at most $2^{p(|\vartheta|)}$ constraints of the form $x : C$. Moreover, each label may obviously contain at most $4\,|\vartheta|$ formulas, namely those in $Fg(\vartheta)$. Hence, it remains to consider constraints of the form xRv where x is a term and v is a variable (note that constraints aRb with a and b object names are formulas and have thus already been considered). Such constraints are only introduced by the $\Longrightarrow_{\exists}$ rule along with constraints of the form $x : C$. Hence, the number of constraints of the form xRv is also bounded by $2^{p(|\vartheta|)}$. Summing up, there exists a polynomial function q such that the size of each node label in $\mathcal{G}$ is bounded $2^{q(|\vartheta|)}$. Since all involved object names, concepts, formulas, and roles are from $Fg(\vartheta)$, there exist at most $2^{2^{q(|\vartheta|)}}$ node labels which are pairwise distinct up to variable renaming. $\dashv$

We are now ready to prove Theorem 5.3. Assume that the tableau algorithm computes a sequence $\mathcal{G}_0, \mathcal{G}_1, \ldots$ of tableaux with $\mathcal{G}_0 = \mathcal{G}_\vartheta$. By Lemma B.2, we may have at most $2^{2^{q(|\vartheta|)}}$ states that are pairwise vdistinct per tableau $\mathcal{G}_i$. By definition of blocking, this implies that we have at most $2^{2^{q(|\vartheta|)}}$ unblocked states in any $\mathcal{G}_i$. By definition of the tableau rules, we have that if a state g in a tableau $\mathcal{G}_i$ is blocked, then g is a leaf in $\mathcal{G}_i$. Also by definition of the tableau rules (see especially the $\Longrightarrow_{\downarrow'}$ rule), the branching factor of tableaux in the above sequence is bounded by $2^{2^{4|\vartheta|}}$. Summing up these facts, we obtain that the number of (blocked or unblocked) states in any $\mathcal{G}_i$ is bounded by $2^{2^{q(|\vartheta|)}} \cdot 2^{2^{4|\vartheta|}}$. Since every rule application except $\Longrightarrow_{\bigcirc}$ generates a new node that is labelled with a strict superset of the label of the (leaf) node to which the rule is applied (and $\Longrightarrow_{\bigcirc}$ itself is only applied to states), there exist at most $2^{4|\vartheta|}$ non-state nodes per state. It follows that the number of nodes in any tableau $\mathcal{G}_i$ is bounded by $2^{2^{r(|\vartheta|)}}$, where r is a polynomial function. Since every rule application adds a new node, the same upper bound applies to the number of rule applications which proves the theorem.

C Proof of Theorem 5.7

We firstly prove correctness:

Theorem C.1 *Suppose ϑ is a satisfiable $\mathsf{PTL}_{\mathcal{ALC}}$-formula and $\mathcal{G}$ is a completion of ϑ. Then the root of $\mathcal{G}$ is not eliminated.*

The theorem is a consequence of the following lemma. Here and in what follows we denote by $[n, m]$ the set $\{k \in \mathbb{N} \mid n \leq k \leq m\}$.

Lemma C.2 *Let $\mathcal{G} = (G, \prec, l)$ be a completion of ϑ and suppose that ϑ is satisfiable. Then there exists a sequence $\vec{g} = (g_i \mid i \in \mathbb{N})$ of clash-free nodes in G with $g_0 = g^r$, $g_i \overline{\prec}\, g_{i+1}$, for all $i \in \mathbb{N}$, such that the following holds for all $n \in \mathbb{N}$:*

- *if α is an eventuality of the form $\varphi \mathcal{U} \psi$ and $\alpha \in l(g_n)$, then there exists $m \geq n$ with $\psi \in l(g_m)$ (in this case we say that α is realized for g_n in $\vec{g}$ until m);*

- *if α is an eventuality of the form $(a : C\mathcal{U}D)$ and $\alpha \in l(g_n)$, then there exists $m \geq n$ with $(a : D) \in l(g_m)$ (in this case we say that α is realized for g_n in $\vec{g}$ until m);*
- *if α is an eventuality of the form $(v : C\mathcal{U}D)$, for some variable v which occurs unmarked in $l(g_n)$, then there exist $m \geq n$ and variables v_i which occur unmarked in $l(g_i)$, for all $i \in [n, m]$, with $v_0 = v$, $(v_m : D) \in l(g_m)$, and for all $i \in [n, m-1]$:*
  - *if $l(g_i)$ is a state, then $\{C \mid (v_i : \bigcirc C) \in l(g_i)\} \subseteq \{C \mid (v_{i+1} : C) \in l(g_{i+1})\}$,*
  - *if $l(g_i)$ is not a state, then $\{C \mid (v_i : C) \in l(g_i)\} \subseteq \{C \mid (v_{i+1} : C) \in l(g_{i+1})\}$,*

  (in this case we say that α is realized for g_n by $\vec{g}$ and the sequence $(v_i \mid n \leq i \leq m)$).

For suppose this lemma has been proved. Then the following proves Theorem C.1.

Proof. Assume ϑ is satisfiable and let $\mathcal{G} = (G, \prec, l)$ be a completion of ϑ.

An application of Lemma C.2 supplies a sequence $\vec{g} = (g_n \mid n \in \mathbb{N})$ satisfying the conditions listed in Lemma C.2. In particular, $g^r = g_0$. So it suffices to show that no g_n from $\vec{g}$ is eliminated. Let

$$G = G_0 \supseteq G_1 \supseteq \cdots$$

be the sequence produced by the elimination procedure. We show by induction that

- $\{g_n \mid n \in \mathbb{N}\} \subseteq G_i$, for all $i \in \mathbb{N}$.

The induction base $i = 0$ is clear. Suppose the claim has been proved for $j = i$. So $\{g_n \mid n \in \mathbb{N}\} \subseteq G_j$. We have to show that none of the rules e_1 to e_3 can be applied to any g_n. (e_1) is clear and $(e_2$, (e_3) follow immediately from the induction hypothesis. $\dashv$

We will now prove a lemma which enables us to prove Lemma C.2. Consider a constraint systems S for ϑ and a model $\mathfrak{M} = (\mathbb{N}, <, I)$ with

$$I(n) = (\Delta, R_0^{I,n}, \ldots, C_0^{I,n}, \ldots, a_0^{I,n}, \ldots),$$

for $n \in \mathbb{N}$. A mapping ρ from the set of unmarked terms in S into Δ is called *n-satisfying*, in symbols $S \subseteq_n^\rho \mathfrak{M}$, if

- $\rho(a) = a^{I,n}$, for all object names a in S,
- $\{C \mid (x : C) \in S\} \subseteq \{C \mid \rho(x) \in C^{I,n}\}$, for all terms x in S,
- if $xRy \in S$, then $\rho(x)R^{I,n}\rho(y)$,
- if $\psi \in S$, then $\mathfrak{M}, n \models \psi$, for all $\mathsf{PTL}_{\mathcal{ALC}}$-formulas ψ.

A mapping σ from Δ *onto* the set of marked variables in S is called *n-exhaustive*, in symbols $S \subseteq_n^\sigma \mathfrak{M}$, if

- $\{C \mid (\sigma(d) : C) \in S\} \subseteq t^{I,n}(d) = \{C \mid d \in C^{I,n}\}$, for all $d \in \Delta$.

A pair (ρ, σ) consisting of an n-satisfying mapping ρ and an n-exhaustive mapping σ is called *fitting*, in symbols $S \subseteq_n^{\rho,\sigma} \mathfrak{M}$.

Lemma C.3 *Suppose $\mathfrak{M} = (\mathbb{N}, <, I)$ with $I(n) = (\Delta, R_0^{I,n}, \ldots, C_0^{I,n}, \ldots, a_0^{I,n}, \ldots)$, for all $n \in \mathbb{N}$, is a model for ϑ. Let $n \in \mathbb{N}$ and suppose $\mathcal{G} = (G, \prec, l)$ is a completion of ϑ and $g \in G$.*

(1) Suppose $l(g)$ is a state which is not blocked, $l(g) \subseteq_n^{\rho,\sigma} \mathfrak{M}$, and v^0 is an unmarked variable in $l(g)$. Then there exist sequences

- *$g_0 \prec g_1 \prec \cdots \prec g_m$ of nodes in $\mathcal{G}$, $g = g_0$, such that $l(g_i)$, $i \in [1, m-1]$, are non-states and $l(g_m)$ is a state,*
- *$(\rho_0, \sigma_0), \ldots, (\rho_m, \sigma_m)$ of pairs of mappings with $(\rho, \sigma) = (\rho_0, \sigma_0)$, and $\rho_1 \subseteq \cdots \subseteq \rho_m$ such that $l(g_i) \subseteq_{n+1}^{\rho_i,\sigma_i} \mathfrak{M}$, for all $i \in [1, m]$,*

and an unmarked variable v^1 in $l(g_1)$ such that

- *(a) for all $\varphi \mathcal{U} \psi \in l(g_m)$, if $\mathfrak{M}, n+1 \models \psi$ then $\psi \in l(g_m)$,*
- *(b) for all unmarked terms x with $(x : C \mathcal{U} D) \in l(g_m)$, if $\rho_m(x) \in D^{I,n+1}$ then $(x : D) \in l(g_m)$,*
- *(c) $\rho_1(v^1) = \rho(v^0)$ and $\{C \mid (v^0 : \bigcirc C) \in l(g)\} \subseteq \{C \mid (v^1 : C) \in l(g_1)\}$.*

(2) Suppose $l(g)$ is not a state, $l(g) \subseteq_n^{\rho,\sigma} \mathfrak{M}$. Then there exist sequences

- *$g_0 \prec g_1 \prec \cdots \prec g_m$ of nodes in $\mathcal{G}$, $g = g_0$, such that $l(g_i)$, $i \in [1, m-1]$, are non-states, and $l(g_m)$ is a state,*
- *pairs $(\rho_0, \sigma_0), \ldots, (\rho_m, \sigma_m)$ of mappings with $\rho = \rho_0$ and $\rho_0 \subseteq \rho_1 \subseteq \cdots \subseteq \rho_m$ such that $l(g_i) \subseteq_n^{\rho_i,\sigma_i} \mathfrak{M}$, for all $i \in [1, m]$*

such that

- *(a) for all $\varphi \mathcal{U} \psi \in l(g_m)$, if $\mathfrak{M}, n \models \psi$ then $\psi \in l(g_m)$,*
- *(b) for all unmarked terms x with $(x : C \mathcal{U} D) \in l(g_m)$, if $\rho(x) \in D^{I,n}$ then $(x : D) \in l(g_m)$.*

Proof. (1) Let $l(g) \subseteq_n^{\rho,\sigma} \mathfrak{M}$ and $\rho(v^0) = d \in \Delta$. Take the node $g_1 \in G$ with $g \prec g_1$. g is not blocked and so $l(g_1)$ is the union of the following sets:

- $\{a : \top\} \cup \{a : C \mid (a : \bigcirc C) \in l(g)\}$, for $a \in ob(l(g))$,
- $\{\psi \mid \bigcirc \psi \in l(g)\}$,
- $\{(v_i : \top)\} \cup \{v_i : C \mid (v_i : \bigcirc C) \in l(g)\}$, for $0 < i \leq n$,
- $\{(v' : \top)\}$,

where $\{v_1, \ldots, v_n\} = \{min([w]_{l(g)}) \mid [w]_{l(g)} \in [l(g)]_\sim\}$ and v' is the only marked variable in $l(g')$. Recall that $\{v_1, \ldots, v_n\}$ is a set of variables from $l(g)$ which contains exactly one representative for each equivalence class $[w]_{l(g)}$. Take the variable $v^1 \in \{v_1, \ldots, v_n\}$ with $v^0 \in [v^1]_{l(g)}$. Define ρ_1 by putting $\rho_1(x) = \rho(x)$ for every term x in $l(g)$. Obviously we have $l(g_1) \subseteq^{\rho_1}_{n+1} \mathfrak{M}$. Note also that $\rho(v^0) = \rho_1(v^1)$. Put $\sigma_1(d) = v'$, for all $d \in \Delta$. Then $l(g_1) \subseteq^{\rho_1,\sigma_1}_{n+1} \mathfrak{M}$ should be clear.

We now construct the sequences $g_2, \ldots, g_m$ and $(\rho_2, \sigma_2), \ldots, (\rho_m, \sigma_m)$ with $g_1 \prec \cdots \prec g_m$ and $\rho_1 \subseteq \cdots \subseteq \rho_m$. This is done by induction. Suppose $g_1, \ldots, g_i$ and $(\rho_1, \sigma_1) \ldots, (\rho_i, \sigma_i)$ with $l(g_j) \subseteq^{\rho_j,\sigma_j}_{n+1} \mathfrak{M}$, $1 \leq j \leq i$, have already been constructed. The choice of g_{i+1} depends on which rule is applied to g_i in the construction of the tableau $\mathcal{G}$:

Case A. One of the rules $\Longrightarrow_s$ is applied to g_i:

Case 1. $\longrightarrow_{\mathcal{U}}$ is applied to $\varphi\mathcal{U}\psi \in l(g_i)$. Then $\mathfrak{M}, n \models \psi$ or $\mathfrak{M}, n \models \varphi, \bigcirc(\varphi\mathcal{U}\psi)$. If the first appears, take g_{i+1} with $l(g_{i+1}) = l(g_i) \cup \{\psi\}$ and $\mathcal{U}$-mark $\varphi\mathcal{U}\psi$ in $l(g_{i+1})$. Otherwise take g_{i+1} with $l(g_{i+1}) = l(g_i) \cup \{\varphi, \bigcirc(\varphi\mathcal{U}\psi)\}$ and $\mathcal{U}$-mark $\varphi\mathcal{U}\psi$. Let $(\rho_{i+1}, \sigma_{i+1}) = (\rho_i, \sigma_i)$.

Case 2. $\longrightarrow_{\neg\mathcal{U}}$ is applied to $\neg(\varphi\mathcal{U}\psi) \in l(g_i)$. Then $\mathfrak{M}, n \models \neg\psi, \neg\varphi$ or $\mathfrak{M}, n \models \neg\psi, \neg\bigcirc(\varphi\mathcal{U}\psi)$. If the first appears, take g_{i+1} with $l(g_{i+1}) = l(g_i) \cup \{\neg\psi, \neg\varphi\}$. Otherwise take g_{i+1} with $l(g_{i+1}) = l(g_i) \cup \{\neg\psi, \neg\bigcirc(\varphi\mathcal{U}\psi)\}$. Put $(\rho_{i+1}, \sigma_{i+1}) = (\rho_i, \sigma_i)$.

Case 3. The deterministic cases where one of the rules $\longrightarrow_{\neg\bigcirc}, \longrightarrow_{\neg\neg}$, or $\longrightarrow_{\wedge}$ is applied to $l(g_i)$ are considered in the obvious manner.

Case 4. $\longrightarrow_{\neg\wedge}$ is be applied to $\neg(\varphi \wedge \psi) \in l(g_i)$. Take g_{i+1} with $l(g_{i+1}) = l(g_i) \cup \{\neg\varphi\}$ if $\mathfrak{M}, n \models \neg\varphi$. Otherwise take g_{i+1} with $l(g_{i+1}) = l(g_i) \cup \{\neg\psi\}$. Put $(\rho_{i+1}, \sigma_{i+1}) = (\rho_i, \sigma_i)$.

Case 5. $\longrightarrow_{\mathcal{U}c}$ is applied to $(x : C\mathcal{U}D) \in l(g_i)$. Suppose first that x is not marked. We have $\rho_i(x) \in D^{I,n}$ or $\rho_i(x) \in C^{I,n}$ and $\rho_i(x) \in (\bigcirc(C\mathcal{U}D))^{I,n}$.

If the first appears, take g_{i+1} with $l(g_{i+1}) = l(g_i) \cup \{(x : D)\}$ and $\mathcal{U}$-mark $(x : C\mathcal{U}D)$. Otherwise take g_{i+1} with $l(g_{i+1}) = l(g_i) \cup \{x : C, x : \bigcirc(C\mathcal{U}D)\}$ and $\mathcal{U}$-mark $(x : C\mathcal{U}D)$. Let $(\rho_{i+1}, \sigma_{i+1}) = (\rho_i, \sigma_i)$.

Suppose now that x is marked in $l(g_i)$. Define

$$\Delta_D = \{d \in \Delta \mid \sigma_i(d) = x, d \in D^{I,n}\},$$

$$\Delta_{\neg D} = \{d \in \Delta \mid \sigma_i(d) = x, d \notin D^{I,n}\}.$$

If $\Delta_D = \emptyset$, then take g_{i+1} with $l(g_{i+1}) = l(g_i) \cup \{(x : C), (x : \bigcirc(C\mathcal{U}D))\}$ and $\mathcal{U}$-mark $(x : C\mathcal{U}D)$. Let $(\rho_{i+1}, \sigma_{i+1}) = (\rho_i, \sigma_i)$.

Suppose now that $\Delta_{\neg D} = \emptyset$. Then take g_{i+1} with $l(g_{i+1}) = l(g_i) \cup \{(x : D)\}$ and $\mathcal{U}$-mark $(x : C\mathcal{U}D)$. Let $(\rho_{i+1}, \sigma_{i+1}) = (\rho_i, \sigma_i)$.

Finally, suppose $\Delta_D \neq \emptyset$ and $\Delta_{\neg D} \neq \emptyset$. Then take g_{i+1} with $l(g_{i+1}) = l(g_i) \cup \text{copy}(S_i, x, v) \cup \{x : D, v : C, v : \bigcirc(C\mathcal{U}D)\}$, $\mathcal{U}$-mark $(x : C\mathcal{U}D)$ and $(v : C\mathcal{U}D)$ and mark x and v. Let $\rho_{i+1} = \rho_i$ and $\sigma_{i+1}(d) = \sigma_i(d)$, for $d \in \Delta - (\Delta_D \cup \Delta_{\neg D})$, $\sigma_{i+1}(d) = x$, for $d \in \Delta_D$, and $\sigma_{i+1}(d) = v$, for $d \in \Delta_{\neg D}$.

Case 6. The rule $\longrightarrow_{\neg\mathcal{U}c}$ is applied to $l(g_i)$. This case is treated analogously to Case 5.

Case 7. The deterministic rules $\longrightarrow_{\neg\neg c}$, $\longrightarrow_{\sqcap}$, $\longrightarrow_{=}$, and $\longrightarrow_{\neg\exists}$ are treated in the obvious manner.

Case 8. The rule $\longrightarrow_{\neg\sqcap}$ is applied to $(x : \neg(C \sqcap D)) \in l(g_i)$. Suppose first that x is unmarked. We have $\rho_i(x) \in (\neg C)^{I,n}$ or $\rho_i(x) \in (\neg D)^{I,n}$. If the first appears take g_{i+1} with $l(g_{i+1}) = l(g_i) \cup \{x : \neg C\}$. Otherwise take g_{i+1} with $l(g_{i+1}) = l(g_i) \cup \{x : \neg D\}$. Put $(\rho_{i+1}, \sigma_{i+1}) = (\rho_i, \sigma_i)$.

Suppose now that x is marked. Define

$$\Delta_{\neg C} = \{d \in \Delta \mid \sigma_i(d) = x, d \in (\neg C)^{I,n}\},$$

$$\Delta_C = \{d \in \Delta \mid \sigma_i(d) = x, d \in C^{I,n}\}.$$

If $\Delta_{\neg C} = \emptyset$, take g_{i+1} with $l(g_{i+1}) = l(g_i) \cup \{x : \neg D\}$ and $(\rho_{i+1}, \sigma_{i+1}) = (\rho_i, \sigma_i)$. If $\Delta_C = \emptyset$, then take g_{i+1} with $l(g_{i+1}) = l(g_i) \cup \{x : \neg C\}$ and $(\rho_{i+1}, \sigma_{i+1}) = (\rho_i, \sigma_i)$.

Suppose now that $\Delta_{\neg C} \neq \emptyset$ and $\Delta_C \neq \emptyset$. Then take g_{i+1} with $l(g_{i+1}) = l(g_i) \cup \mathrm{copy}(S_i, x, v) \cup \{x : \neg C, v : \neg D\}$. Put $\rho_{i+1} = \rho_i$ and $\sigma_{i+1}(d) = \sigma_i(d)$ for $d \in \Delta - (\Delta_C \cup \Delta_{\neg C})$, $\sigma_{i+1}(d) = x$, for $d \in \Delta_{\neg C}$, and $\sigma_{i+1}(d) = v$, for $d \in \Delta_C$.

Case 9. The rule $\longrightarrow_{\neq}$ is applied to $C \neq \top \in l(g_i)$. Then we have $\mathfrak{M}, n \models C \neq \top$ and so we can take $e \in \Delta$ with $e \in (\neg C)^{I,n}$. We know that the outcome is $S_{i+1} = l(g_i) \cup \{w : \neg C\}$, for some new and unmarked variable w. Put $\rho_{i+1} = \rho_i \cup \{(w, e)\}$ and $\sigma_{i+1} = \sigma_i$.

Case 10. The rule $\longrightarrow_{\exists}$ is applied to $(x : \exists R.C) \in l(g_i)$. If x is unmarked we have $\rho_i(x) \in (\exists R.C)^{I,n}$. Hence we find $e \in \Delta$ with $\rho_i(x) R^{I,n} e$ and $e \in C^{I,n}$. If x is marked we find a $d \in \Delta$ with $\sigma_i(d) = x$. So again we find $e \in \Delta$ with $d R^{I,n} e$ and $e \in C^{I,n}$. In both cases we know that the outcome is $S_{i+1} = l(g_i) \cup \{w : C, xRw\}$ for a new and unmarked variable w. Put $\rho_{i+1} = \rho_i \cup \{(w, e)\}$ and $\sigma_{i+1} = \sigma_i$.

Case B. The rule $\Longrightarrow_{\downarrow}$ is applied to g_i and the variable v. Then v is unmarked in $l(g_i)$. Suppose $\rho_i(v) = d$. Take the marked variable $w = \sigma_0(d)$ in $l(g_0)$ and let g_{i+1} be the node with $l(g_{i+1}) = l(g_i) \cup \{(v : C) \mid (w : \bigcirc C) \in l(g_0)\}$. Let $\rho_{i+1} = \rho_i$ and $\sigma_{i+1} = \sigma_i$.

Case C. The rule $\Longrightarrow_{\downarrow'}$ is applied to g_i and the variable v. Then v is marked in $l(g_i)$. Let D be the set of $d \in \Delta$ such that $\sigma_i(d) = v$. $D \neq \emptyset$ since σ_i is onto. Let

$$Y = \{w \mid \exists d \in D\, \sigma_0(d) = w\}$$

and assume $Y^* = \{w_1, \ldots, w_n\} = \{\min([w]_{l(g_0)}) \mid w \in Y\}$. Choose the node g_{i+1} in such a way that $l(g_{i+1})$ consists of $l(g_i)$ and the union of

- $\{v : C \mid (w_1 : \bigcirc C) \in l(g_0)\}$,
- $\mathrm{copy}(l(g_i), v, v'_i)$, $1 < i \leq n$,
- $\{(v'_i : C) \mid (w_i : \bigcirc C) \in l(g_0)\}$, for $1 < i \leq n$.

Define $\rho_{i+1} = \rho_i$ and $\sigma_{i+1}(d) = \sigma_i(d)$, for $d \in \Delta - D$, $\sigma_{i+1}(d) = v$, for $\sigma_0(d) = w_1$, and $\sigma_{i+1}(d) = v'_i$ for $\sigma_0(d) = w_i$, $1 < i \leq n$.

Suppose now that we have reached some m such that g_m is a state. Theorem 5.3 guarantees the existence of m. We show that the constructed sequences $g_1, \ldots, g_m$ and $(\rho_1, \sigma_1), \ldots, (\rho_m, \sigma_m)$ are as required. Firstly, it is clear from the construction that $g_i \prec g_{i+1}$, for all $i < m$, and that $l(g_m)$ is the only state in this list. $\rho_i \subseteq \rho_{i+1}$ is also clear and $l(g_i) \subseteq_{n+1}^{\rho_i, \sigma_i} \mathfrak{M}$ is easily proved by induction on i. It remains to verify the conditions (a)-(c).

(a) Suppose $\varphi \mathcal{U} \psi \in l(g_m)$ and $\mathfrak{M}, n+1 \models \psi$. None of the rules $\Longrightarrow_s$ is applicable to g_m, so $\varphi \mathcal{U} \psi$ is $\mathcal{U}$-marked in g_m. But $\varphi \mathcal{U} \psi$ is not $\mathcal{U}$-marked in g_1 and so there exists $i < m$ such that $l(g_i) \Longrightarrow_{\mathcal{U}} l(g_{i+1})$ and $\varphi \mathcal{U} \psi$ occurs $\mathcal{U}$-unmarked in $l(g_i)$. But then, by the above procedure, $\psi \in l(g_{i+1})$ and so $\psi \in l(g_m)$.

(b) Suppose $(x : C \mathcal{U} D) \in l(g_m)$, x is unmarked in $l(g_m)$ and $\rho_m(x) \in D^{I,n+1}$. None of the rules $\Longrightarrow_s$ is applicable to g_m, so $(x : C \mathcal{U} D)$ is $\mathcal{U}$-marked in g_m. But $(x : C \mathcal{U} D)$ is not $\mathcal{U}$-marked in g_1 and so there exists $i < m$ such that $l(g_i) \Longrightarrow_{\mathcal{U}c} l(g_{i+1})$ and $(x : C \mathcal{U} D)$ occurs $\mathcal{U}$-unmarked in $l(g_i)$. But then, by the above procedure, $(x : D) \in l(g_{i+1})$ and so $(x : D) \in l(g_m)$.

(c) follows from the definition.

This completes the proof of the first claim of the Lemma.

The proof of the second claim can be conducted in precisely the same manner as the proof above starting from g_1. It is left to the reader. $\dashv$

Notice that we did not exclude the possibility that g_m is blocked. In this case we find a state g' which is not blocked such that $g_{m-1} \overline{\prec} g'$ and $l(g')$ is a variant of $l(g_m)$. It is straightforward (but tedious) to reformulate the lemma above using the unblocked state $l(g')$ instead of $l(g)$.

Now we are ready for proving Lemma C.2.

Proof. Suppose ϑ is satisfiable. We find $\mathfrak{M} = (\mathbb{N}, <, I)$ with

$$I(n) = (\Delta, R_0^{I,n}, \ldots, C_0^{I,n}, \ldots, a_0^{I,n}, \ldots),$$

for all $n \in \mathbb{N}$, and $\mathfrak{M}, 0 \models \vartheta$. We show the following

Claim. There exists a sequence of intervals $\{[n(i), m(i)] \mid i \in \mathbb{N}\}$ of natural numbers with $n(0) = 0$ and $m(i) + 1 = n(i+1)$, for all $i \in \mathbb{N}$, such that there exist sequences

- $\vec{g} = (g_n \mid n \in \mathbb{N})$ of nodes in G with $g_0 = g^r$, and
- $\vec{\rho} = (\rho_n \mid n \in \mathbb{N})$ and $(\sigma_n \mid n \in \mathbb{N})$ of mappings

such that the following hold:

(c1) for all $n \in \mathbb{N}$: $g_n \overline{\prec} g_{n+1}$,

(c2) for all $n \in \mathbb{N}$: $l(g_n)$ is a state iff there exists $i \in \mathbb{N}$ with $n = m(i)$,

(c3) for all $i \in \mathbb{N}$ and $n \in [n(i), m(i)]$: $l(g_n) \subseteq_i^{\rho_n, \sigma_n} \mathfrak{M}$,

(c4) for all $i \in \mathbb{N}$ and all $\varphi \mathcal{U} \psi \in l(g_{m(i)})$, if $\mathfrak{M}, i \models \psi$ then $\psi \in l(g_{m(i)})$, (c5) for all $i \in \mathbb{N}$ and all $(x : C\mathcal{U}D) \in l(g_{m(i)})$ with x unmarked, if $\rho_{m(i)}(x) \in D^{I,i}$ then $(x : D) \in l(g_{m(i)})$,

(c6) for all $n \in \mathbb{N}$ every unmarked eventuality for g_n is realized by $\vec{g}$.

By (c6), the sequence $(g_n \mid n \in \mathbb{N})$ is as required for proving Lemma C.2.

The construction of the sequences is by induction. We start with g_0, ρ_0, and σ_0: let $g_0 = g^r$, $\rho_0(a) = a^{I,0}$, for all $a \in ob(\vartheta)$, and $\sigma_0(d) = min(V)$ for all $d \in \Delta$. Obviously $l(g_0) \subseteq_0^{\rho_0,\sigma_0} \mathfrak{M}$.

Suppose now that we have constructed a sequence of intervals $\{[n(i), m(i)] \mid i \leq k\}$ and sequences

- $(g_n \mid n \leq m(k))$ and $(\rho_n, \sigma_n \mid n \leq m(k))$

satisfying the conditions (c1)-(c5) stated in the claim above until $m(k)$ — save that $l(g_{m(k)})$ is possibly not a state. In the latter case, using Lemma C.3 (2), it is straightforward to extend these sequences by means of nodes $g_{m(k)+1}, \dots, g_{m(k)+l}$ and mappings $\rho_{m(k)+1}, \dots, \rho_{m(k)+l}$, $\sigma_{m(k)+1}, \dots, \sigma_{m(k)+1}$ such that $l(g_{m(k)+l})$ is a state and the extended sequence still has all properties (c1)-(c5). So, we can assume without loss of generality that $l(g_{m(k)})$ is a state. We now distinguish two cases.

Case 1. There exists an unmarked eventuality for some $l(g_n)$, $n \leq m(k)$, which is not realized by $(g_n \mid n \leq m(k))$ until $m(k)$.

Take a minimal $l \leq m(k)$ such that there exists an eventuality α for $l(g_l)$ which is not realized until $m(k)$. To begin with suppose α is of the form $(v : C'\mathcal{U}D')$, for some unmarked variable v. We take variables $v_l, \dots, v_{m(k)}$ with $v_l = v$ such that, for all $i \in [l, m(k)]$,

- v_i occurs unmarked in $l(g_i)$,
- $\{C \mid (v_i : C) \in l(g_i)\} \subseteq \{C \mid (v_{i+1} : C) \in l(g_{i+1})\}$, whenever $i \leq m(k)$ and g_i is not a state,
- $\{C \mid (v_i : \bigcirc C) \in l(g_i\} \subseteq \{C \mid (v_{i+1} : C) \in l(g_{i+1})\}$, whenever $i < m(k)$ and $l(g_i)$ is a state.

Assume $\rho_{m(k)}(v_{m(k)}) = d \in \Delta^k$. We have $(v_{m(k)} : C'\mathcal{U}D') \in l(g_{m(k)})$ and $(v_{m(k)} : D') \notin l(g_{m(k)})$, since $(v : C'\mathcal{U}D')$ is not realized until $m(k)$. Hence $d \in (C'\mathcal{U}D')^{I,k}$, by (c3), and $d \notin (D')^{I,k}$, by (c5). There exists $n > k$ such that $d \in (D')^{I,n}$. Assume that n is minimal with this property.

Iterated application of Lemma C.3 (1) provides intervals $\{[n(j), m(j)] \mid k+1 \leq j \leq n\}$ with $n(k+1) = m(k) + 1$ and $m(j) + 1 = n(j+1)$ for all $j \in [k+1, n-1]$, and sequences

- $g_{n(k+1)} \overline{\prec} \cdots \overline{\prec} g_{m(n)}$, $\rho_{n(k+1)}, \dots, \rho_{m(n)}$, $\sigma_{n(k+1)}, \dots, \sigma_{m(n)}$

such that (c1)-(c5) above hold until $m(n)$, and there is a sequence of variables $v_{n(k+1)}, \dots, v_{m(n)}$, v_j occurs unmarked in $l(g_j)$ for all $j \in [n(k+1), m(n)]$, such that the following hold for all $j \in [n(k+1), m(n)]$:

1. $\rho_j(v_j) = d$,
2. $\{C \mid (v_j : C) \in l(g_j)\} \subseteq \{C \mid (v_{j+1} : C) \in l(g_j)\}$, whenever g_j is not a state,
3. $\{C \mid (v_{m(i)} : \bigcirc C) \in l(g_{m(i)}\} \subseteq \{C \mid (v_{n(i+1)} : C) \in l(g_{n(i+1)})\}$, for all $i \in [k, n-1]$.

We show that $(v : C'\mathcal{U}D')$ is realized for g_l by $(g_j \mid 0 \leq j \leq m(n))$ and the sequence

$$v_l, \dots, v_{m(k)}, v_{n(k+1)}, \dots, v_{m(n)}.$$

To this end it suffices to show that $(v_{m(n)} : D') \in l(g_{m(n)})$. But, by the minimality of n, we have $(v_j : C'\mathcal{U}D') \in l(g_j)$ for all $j \in [l, m(n)]$. Moreover, $\rho_{m(n)}(v_{m(n)}) \in (D')^{I,n}$. Using (c5) we infer $(v_{m(n)} : D') \in l(g_{m(n)})$. This concludes the case where α has the form $(v : (C'\mathcal{U}D'))$.

The cases where the eventuality α is of the form $(a : C'\mathcal{U}D')$ or $\varphi\mathcal{U}\psi$ are treated similarly and can be left to the reader.

Case 2. Every unmarked eventuality for every g_n, $n \leq m(k)$, is realized by $(g_n \mid n \leq m(k))$ until $m(k)$.

This case is easier than the first one since no eventuality has to be realized. We just take the unique g' with $l(g_{m(k)}) \prec l(g')$ and add it to the list $(g_n \mid n \leq m(k))$. The required mappings ρ' and σ' showing $l(g') \subseteq_{k+1}^{\rho',\sigma'} \mathfrak{M}$ are obtained by putting, for every object name a in $l(g')$, $\rho'(a) = \rho_{m(k)}(a)$, and for every unmarked variable w in $l(g')$, $\rho'(w) = \rho_{m(k)}(w)$. For $d \in \Delta$ we let $\sigma'(d) = v'$ for the unique marked variable v' in $l(g')$.

Now, in the limit we obtain sequences $(g_n \mid n \in \mathbb{N})$ and $(\rho_n, \sigma_n | n \in \mathbb{N})$ which obviously satisfy the conditions (c1)-(c5). (c6) can be shown as follows: suppose α is an unmarked eventuality in $l(g_m)$. Then, since the number of eventualities in each $l(g_n)$ is finite, eventually α will be realized in the construction of $(g_n \mid n \in \mathbb{N})$. $\dashv$

Now we prove the completeness part:

Theorem C.4 *Let ϑ be some PTL_{ALC}-formula. If there exists a completion of ϑ in which the root is not eliminated then ϑ is satisfiable.*

Proof. Assume $\mathcal{G} = (G, \prec, l)$ is some complete tableau for ϑ the root g^r of which is not eliminated. Let $B \subseteq G$ be the set of nodes which remain after the execution of the elimination procedure. We have $g^r \in B$. We must show that ϑ is satisfiable. By Theorem 4.4 it is sufficient to prove the existence of some quasimodel that satisfies ϑ. In order to do this we are going to show the following

Claim. There exists a sequence of intervals $\{[n(i), m(i)] \mid i \in \mathbb{N}\}$ of natural numbers with $n(0) = 0$ and $m(i) + 1 = n(i+1)$, for all $i \in \mathbb{N}$, such that there exists a sequence $\vec{g} = (g_n \mid n \in \mathbb{N})$ of nodes in B with $g_0 = g^r$ and the following holds for all $n \in \mathbb{N}$:

(d1) $g_n \overline{\prec} g_{n+1}$,

(d2) $l(g_n)$ is a state iff there exists $i \in \mathbb{N}$ with $n = m(i)$, (d3) every eventuality for $g_{m(i)}$, $i \in \mathbb{N}$ is realized by $\vec{g}$.

Suppose the sequence has been constructed. We then reason as follows: By assumption, (e1) is not applicable to B. Hence all $l(g_n)$ are clash-free. By utilizing (d1)-(d3) it is not difficult to show that $(l(g_{m(i)}) \mid i \in \mathbb{N})$ is a quasimodel satisfying ϑ. We leave the details to the reader.

The construction of the sequences is by induction. We start with $g_0 = g^r$. Suppose now that we have constructed a sequence of intervals $\{[n(i), m(i)] \mid i \leq k\}$ and sequences $(g_n \mid n \leq m(k))$ satisfying the conditions (d1)-(d2) stated in the claim above until $m(k)$ save that $l(g_{m(k)})$ is possibly not a state. Using the condition that (e2) cannot be applied to B it is straightforward to extend this sequence by means of nodes $g_{m(k)+1}, \dots, g_{m(k)+l}$ from B such that $l(g_{m(k)+l})$ is a state and the extended sequence still has properties (d1),(d2). So, we can assume without loss of generality that we have $\{[n(i), m(i)] \mid i \leq k\}$ and a sequence $(g_n \mid n \leq m(k))$ satisfying (d1) and (d2); in particular $g_{m(k)}$ is a state. We distinguish two cases.

Case 1. There exists an eventuality for some $l(g_{m(i)})$, $i \leq k$, which is not realized by $(g_n \mid n \leq m(k))$ until $m(k)$.

Choose $l = m(j) \leq m(k)$ minimal such that there exists an eventuality α for $l(g_l)$ which is not realized until $m(k)$. First suppose α is of the form $(v : C'\mathcal{U}D')$, for an unmarked variable v.

We take variables $v_l, \dots, v_{m(k)}$ with $v_l = v$ and $v_{m(k)}$ occurs unmarked in $l(g_{m(k)})$ such that, for all $i \in [l, m(k) - 1]$,

- v_i occurs unmarked in $l(g_i)$,
- $\{C \mid (v_i : C) \in l(g_i)\} \subseteq \{C \mid (v_{i+1} : C) \in l(g_{i+1})\}$, whenever v_i is not a state,
- $\{C \mid (v_i : \bigcirc C) \in l(g_i\} \subseteq \{C \mid (v_{i+1} : C) \in l(g_{i+1})\}$, whenever $l(g_i)$ is a state.

We have $(v_{m(k)} : C'\mathcal{U}D') \in l(g_{m(k)})$ and $(v_{m(k)} : D') \notin l(g_{m(k)})$, since $(v : C'\mathcal{U}D')$ is not realized until $m(k)$. (e3) is not applicable to any node in B, in particular not to $g_{m(k)}$. So we find $\{[n(j), m(j)] \mid k+1 \leq j \leq n\}$ with $m(j) + 1 = n(j+1)$ for all $j \in [k+1, n-1]$, and a sequence

- $g_{m(k)} \overline{\prec} g_{n(k+1)} \overline{\prec} \dots \overline{\prec} g_{m(n)}$ of nodes in B

such that (d1), (d2) above hold until $m(n)$, and there is a sequence of variables $v_{n(k+1)}, \dots, v_{m(n)}$ such that v_j occurs unmarked in $l(g_j)$ for all $j \in [n(k+1), m(n)]$ and $(g_j \mid l \leq j \leq m(n))$ and $(v_j | l \leq j \leq m(n))$ realize $C\mathcal{U}D'$ until $m(n)$.

Suppose now that v is marked. Then $(v : \bigcirc(C'\mathcal{U}D')) \in l(g_l)$, since $(v : D') \notin l(g_l)$ and $l(g_l)$ is saturated. If $l = m(j) < m(k)$, then take a sequence of variables $v_{l+1}, \dots, v_{m(j+1)}$ such that

- $\{C \mid (v_l : \bigcirc C) \in l(g_l\} \subseteq \{C \mid (v_{l+1} : C) \in l(g_{l+1})\}$,
- v_i occurs unmarked in $l(g_i)$, for all $i \in [l+1, m(j+1)]$
- $\{C \mid (v_i : C) \in l(g_i)\} \subseteq \{C \mid (v_{i+1} : C) \in l(g_{i+1})\}$, for all $i \in [l+1, m(j+1) - 1]$.

We have $(v_{m(j+1)} : C'\mathcal{U}D') \in l(g_{m(j+1)})$. Now proceed with the variable $v_{m(j+1)}$ as above in the case of unmarked variables.

If $l = m(j) = m(k)$, then take a sequence of nodes $g_{n(k+1)} \overline{\prec} \cdots \overline{\prec} g_{m(k+1)}$ and of variables $v_{n(k+1)}, \ldots, v_{m(k+1)}$ such that $g_{m(k+1)}$ is the only state in this sequence, $g_{m(k)} \prec g_{n(k+1)}$, and

- $\{C \mid (v_l : \bigcirc C) \in l(g_l\} \subseteq \{C \mid (v_{l+1} : C) \in l(g_{l+1})\}$,
- v_i occurs unmarked in $l(g_i)$, for all $i \in [l+1, m(k+1)]$
- $\{C \mid (v_i : C) \in l(g_i)\} \subseteq \{C \mid (v_{i+1} : C) \in l(g_{i+1})\}$, for all $i \in [l+1, m(k+1)-1]$.

We have $(v_{m(k+1)} : C'\mathcal{U}D')) \in l(g_{m(k+1)})$. Now proceed with the variable $v_{m(k+1)}$ as above for unmarked variables. The cases where the eventuality α is of the form $(a : C'\mathcal{U}D')$ or $\varphi\mathcal{U}\psi$ are treated similarly and left to the reader.

Case 2. Every eventuality for some $l(g_{m(i)})$, $i \leq k$, is realized by $(g_n \mid n \leq m(k))$ until $m(k)$.

In this case no eventuality has to be realized. Thus, using that (e2) is not applicable to B, we can extend the sequence $(g_n \mid n \leq m(k))$ by a new node $g_{m(k)+1} \in B$ with $g_{m(k)} \overline{\prec} g_{m(k)+1}$.

In the limit we obtain a sequence $\vec{g}$ satisfying (d1)-(d3). $\dashv$

References

[1] A. Artale and E. Franconi. Temporal description logics. In L. Vila, Peter van Beek, M. Boddy, M. Fisher, Dov M. Gabbay, A. Galton, and R. Morris, editors, *Handbook of Time and Temporal Reasoning in Artificial Intelligence.* 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*, pages 808–814, 1995. Morgan Kaufmann.

[5] F. Baader and U. Sattler. Tableau algorithms for description logics. In R. Dyckhoff, editor, *Proceedings of Tableaux 2000*, vol. 1847 of LNAI, pages 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, North-Holland, 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-Kaufman, 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] 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.

[13] K. Schild. Combining terminological logics with tense logic. In *Proceedings of the 6th Portuguese Conference on Artificial Intelligence*, pages 105–120, Porto, 1993.

[14] A. Schmiedel. A temporal terminological logic. In *Proceedings of the 9th National Conference of the American Association for Artificial Intelligence*, pages 640–645, Boston, 1990.

[15] H. Sturm and F. Wolter. A tableau calculus for temporal description logic: The expanding domain case. Journal of Logic and Computation, 2001. In print.

[16] P. Wolper. The tableau method for temporal logic: An overview. *Logique et Analyse*, 28:119–152, 1985.

[17] F. Wolter and M. Zakharyaschev. Satisfiability problem in description logics with modal operators. In Anthony G. Cohn, Lenhart Schubert, and Stuart C. Shapiro, editors, *KR'98*, pages 512–523. Morgan Kaufmann, 1998.

[18] F. Wolter and M. Zakharyaschev. Multi-dimensional description logics. In Dean Thomas, editor, *Proceedings of IJCAI'99*, pages 104–109, Morgan Kaufmann, 1999.

[19] F. Wolter and M. Zakharyaschev. Temporalizing description logic. In D. Gabbay and M. de Rijke, editors, *Frontiers of Combining Systems 2*, pages 379–402. Studies Press/Wiley, 2000.