# Towards an Implementation of the Temporal Description Logic $\mathcal{TL}_{\mathcal{ALC}}$

Christian Günsel and Marco Wittmann
Institute for Computer Science, University of Leipzig, Germany
Email: guensel@informatik.uni-leipzig.de,
mai95ced@studserv.uni-leipzig.de

## 1 Introduction

This paper describes first steps towards an efficient implementation of a tableau calculus for the logic $\mathcal{TL}_{\mathcal{ALC}}$ of [14, 11], a temporal extension of the standard description logic $\mathcal{ALC}$. $\mathcal{TL}_{\mathcal{ALC}}$ extends $\mathcal{ALC}$ by means of the temporal operators $\bigcirc$ (next) and $\mathcal{U}$ (until) which are applied to concepts and formulas. The expression

$$\Box(\text{Living\_being} \sqsubseteq \text{Mortal}) \wedge \Diamond\Box(\neg\text{Living\_being} = \top)$$

for example formalizes the statement 'living beings are mortal and from some time on there will never be any living being again'. (Here the abbreviations $\Diamond C = \top\mathcal{U}C$ and $\Box C = \neg\Diamond\neg C$ are used). 'Mortal' could be defined by

$$\Box(\text{Mortal} = \text{Living\_being} \sqcap (\text{Living\_being}\ \mathcal{U}\ \Box\neg\text{Living\_being})).$$

$\mathcal{TL}_{\mathcal{ALC}}$ is interpreted in models that are based on the flow of time $\langle\mathbb{N}, <\rangle$ and in which for each $n \in \mathbb{N}$ an $\mathcal{ALC}$-model interpreting the $\mathcal{ALC}$-concepts at time $n$ in a domain $\Delta^n$ is specified. The resulting logic depends on the 'domain assumption' the intended models comply with: we can make the *expanding domain* assumption (i.e. $\Delta^n \subseteq \Delta^{n+1}$ for all $n \in \mathbb{N}$), or the *constant domain* assumption ($\Delta^n = \Delta^{n+1}$ for all $n \in \mathbb{N}$). While, on the one hand, applications of $\mathcal{TL}_{\mathcal{ALC}}$—say, for temporal conceptual modeling—more often seem to require constant domains, cf. [3, 1, 2, 4], there is, on the other hand, more hope for an efficient implementation under the expanding than under the constant domain assumption. Firstly, in [13] it is shown how expanding domain satisfiability can be reduced to constant domain satisfiability and, secondly, the tableau calculus presented in [11] for expanding domain satisfiability is considerably less complex than the tableau for constant domain models presented in [8]. In fact, the constant domain tableau extends the expanding domain tableau by means of extremely expensive rules. Note that the constant domain case is more general, because expanding domains can be simulated by using a special concept *Exists* which must apply to any object under consideration.

In this paper we report the first steps towards an efficient implementation of the *expanding domain* tableau for $\mathcal{TL}_{\mathcal{ALC}}$ presented in [11]. To the best of our knowledge it describes the first attempt implementing non-trivial combinations of modal and temporal logics with description logics. For this reason and the fact that even under the expanding domain assumption the computational complexity of the satisfiability problem is EXPSPACE-complete, cf. [13], it seemed reasonable not to start with the more complicated (although also EXPSPACE-complete) constant domain case but with expanding domains.

Besides the usefulness of temporal description logics, there are several aspects that make an implementation interesting: the tableau developed in [11] combines a standard tableau for $\mathcal{ALC}$, cf. [10, 6], with Wolper's tableau from [12] for propositional temporal logic. The implementation of the combined logic should be as modular as possible: firstly, modularity will allow us to apply optimization techniques developed for the component logics without any modification. Secondly, modularity could allow to use existing systems like DLP, FaCT, or RACER (see [7, 6, 5] and references therein) in a client-server architecture: while the new prover is responsible for the temporal part, satisfiability of $\mathcal{ALC}$-concepts is delegated to an existing description logic prover. Another interesting aspect is the type of tableau calculus which is implemented: it runs through two phases. In the first phase, the algorithm starts with a formula and exhaustively applies tableau rules. Finally a tableau $G$ is obtained to which no more rule is applicable. In the second phase, we recursively eliminate nodes from $G$ by means of elimination rules. The formula turns out to be satisfiable iff the root is not eliminated. Implementing the algorithm in this manner with two phases will be extremely unefficient, however. The main optimization problem is to find out, how the two phases interact so that, in contrast to the theoretical presentation, one can alternate between the first and the second phase in a systematic manner.

We should like to emphasize that the implementation we have so far is just a prototype: the possible optimizations mentioned above are not yet realized.

The paper also describes a visualization tool for tableaux which is useful for teaching the tableau calculus and for debugging.

*Acknowledgements* Work on this paper was supported by the DFG project Wo582/3-1.

## 2 The Logic

Table 2 defines the implemented temporal description logic $\mathcal{TL}_{\mathcal{ALC}}$. Notice that $a_i$ are object names, $C_i$ concept names, $R_i$ role names, and $C$ and $D$ arbitrary concepts. $C = D$ and $a : C$ are atomic formulas, $\varphi$ and $\psi$ complex formulas.

| | **Syntax** | **Semantics** |
|---|---|---|
| Objects | $a_0, a_1, \ldots$ | $a_0^{I,n} \in \Delta^{I,n}, \ldots$ |
| Roles | $R_0, R_1, \ldots$ | $R_0^{I,n} \subseteq \Delta^n \times \Delta^n, \ldots$ |
| Concepts | $C_0, C_1, \ldots$ | $C_0^{I,n} \subseteq \Delta^n, \ldots$ |
| | $\top$ | $\top^{I,n} = \Delta^n$ |
| | $C \sqcap D$ | $(C \sqcap D)^{I,n} = C^{I,n} \cap D^{I,n}$ |
| | $\neg C$ | $(\neg C)^{I,n} = \Delta^n \setminus C^{I,n}$ |
| | $\exists R_i.C$ | $(\exists R_i.C)^{I,n} = \{d \in \Delta^n \mid \exists d' \in C^{I,n} : dR_i^{I,n}d'\}$ |
| | $\bigcirc C$ | $(\bigcirc C)^{I,n} = \{d \in \Delta^n \mid d \in C^{I,n+1}\}$ |
| | $C\mathcal{U}D$ | $(C\mathcal{U}D)^{I,n} = \{d \in \Delta^n \mid \exists m \geq n(d \in D^{I,m}$ & $\forall k(n \leq k < m \Rightarrow d \in C^{I,k}))\}$ |
| Formulas | $C = D$ | $\mathfrak{M}, n \models C = D$ iff $C^{I,n} = D^{I,n}$ |
| | $a : C$ | $\mathfrak{M}, n \models a : C$ iff $a^{I,n} \in C^{I,n}$ |
| | $\top$ | $\mathfrak{M}, n \models \top$ |
| | $\neg\varphi$ | $\mathfrak{M}, n \models \neg\varphi$ iff $\mathfrak{M}, n \not\models \varphi$ |
| | $\varphi \wedge \psi$ | $\mathfrak{M}, n \models \varphi \wedge \psi$ iff $\mathfrak{M}, n \models \varphi$ and $\mathfrak{M}, n \models \psi$ |
| | $\bigcirc\varphi$ | $\mathfrak{M}, n \models \bigcirc\varphi$ iff $\mathfrak{M}, n+1 \models \varphi$ |
| | $\varphi\mathcal{U}\psi$ | $\mathfrak{M}, n \models \varphi\mathcal{U}\psi$ iff there is some $m \geq n$ with $\mathfrak{M}, m \models \psi$ such that for all $k$ with $n \leq k < m$, $\mathfrak{M}, k \models \varphi$ |

Table 1: $\mathcal{TL}_{\mathcal{ALC}}$ Syntax and Semantics

A $\mathcal{TL}_{\mathcal{ALC}}$-model is a triple $\mathfrak{M} = (\mathbb{N}, <, I)$, where $(\mathbb{N}, <)$ is the set of natural numbers equipped with the usual strict order and $I$ is a function associating with each $n \in \mathbb{N}$ some $\mathcal{ALC}$-model

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

in which $R_i^{I,n}$ are binary relations on $\Delta^n$, $C_i^{I,n}$ are subsets of $\Delta^n$, and $a_i^{I,n} \in \Delta^{I,n}$ such that $a_i^{I,n} = a_i^{I,m}$ for every $n, m \in \mathbb{N}$, and $\Delta^n \subseteq \Delta^m$ for every $n, m \in \mathbb{N}$ with $n < m$.

A formula $\varphi$ is called *satisfiable* iff there exists a model $\mathfrak{M}$ and a moment $n \in \mathbb{N}$ such that $\mathfrak{M}, n \models \varphi$.

| |
|---|
| $S \longrightarrow_{\mathcal{U}} X \cup \{(\varphi \mathcal{U} \psi)^*\} \cup S$,<br>if $(\varphi \mathcal{U} \psi)$ occurs without $^*$ in $S$, $X = \{\psi\}$ or $X = \{\varphi, \bigcirc(\varphi \mathcal{U} \psi)\}$<br>$S \longrightarrow_{\neg\mathcal{U}} X \cup S$,<br>if $\neg(\varphi \mathcal{U} \psi) \in S$, $\{\neg\psi, \neg\varphi\} \not\subseteq S$ and $\{\neg\psi, \bigcirc\neg(\varphi \mathcal{U} \psi)\} \not\subseteq S$,<br>$X = \{\neg\psi, \neg\varphi\}$ or $X = \{\neg\psi, \bigcirc\neg(\varphi \mathcal{U} \psi)\}$<br>$S \longrightarrow_{\neg\bigcirc} \{\bigcirc\neg\varphi\} \cup S$,<br>if $\neg \bigcirc \varphi \in S$, $\bigcirc\neg\varphi \notin S$<br>$S \longrightarrow_{\mathcal{U}c} X \cup \{(x : (C \mathcal{U} D))^*\} \cup S$,<br>if $x : (C \mathcal{U} D)$ occurs without $^*$ in $S$,<br>$X = \{x : D\}$ or $X = \{x : C, x : \bigcirc(C \mathcal{U} D)\}$<br>$S \longrightarrow_{\neg\mathcal{U}c} X \cup S$,<br>if $x : \neg(C \mathcal{U} D) \in S$, $\{x : \neg D, x : \neg C\} \not\subseteq S$<br>and $\{x : \neg D, x : \bigcirc\neg(C \mathcal{U} D)\} \not\subseteq S$,<br>$X = \{x : \neg D, x : \neg C\}$ or $X = \{x : \neg D, x : \bigcirc\neg(C \mathcal{U} D)\}$<br>$S \longrightarrow_{\neg\bigcirc c} \{x : \bigcirc\neg C\} \cup S$,<br>if $x : \neg \bigcirc C \in S$, $x : \bigcirc\neg C \notin S$ |

Table 2: Local temporal rules

# 3 The Tableau

We briefly describe the implemented tableau calculus. For details we refer the reader to [11]. The rules given in tables 2, 3, 4, and 5 operate on constraint systems – finite sets of extended $\mathcal{TL}_{\mathcal{ALC}}$-formulas in which *variables* can occur in the place of object names.

The algorithm starts with the constraint system $\{\varphi, x_0 : \top\}$. The inclusion of $x_0 : \top$ forces the domain to be nonempty. A $\mathcal{TL}_{\mathcal{ALC}}$-tableau $(G, \prec, l)$ consists of a finite tree $(G, \prec)$ and a function $l$ which assigns a constraint system to each node. An edge represents a rule application, the children of a node contain the constraint systems generated by that rule. If no *local rule* is applicable to a node, then it is called a state (and represents a moment of time). The global rule $\longrightarrow_{\bigcirc}$ can be applied to states only and creates a new world (pre-state). A branch of the tableau is closed if and only if the constraint system at its leaf contains a clash or, if after the application of the $\longrightarrow_{\bigcirc}$-rule a pre-state is created which already occurs in the tree expanded so far (pre-state blocking).

The algorithm requires to build the complete tableau; i.e., the rules having more than one possible outcome are – in contrast to (implementations of) standard tableaux – not interpreted as nondeterministic but branching. That is to say, both subtrees are expanded. We call those rules *branching rules*. We mention that the rules $S \longrightarrow_{\mathcal{U}c}$ and $S \longrightarrow_{\mathcal{U}}$ generate formulas which are marked by $^*$ to prevent further application of these rules, for details see [11] again.

| |
|---|
| $S \longrightarrow_{\neg\neg} \{\varphi\} \cup S$,<br>        if $\neg\neg\varphi \in S$,$\varphi \notin S$<br>$S \longrightarrow_{\wedge} \{\varphi, \psi\} \cup S$,<br>        if $\varphi \wedge \psi \in S$, $\varphi \notin S$ or $\psi \notin S$<br>$S \longrightarrow_{\neg\wedge} \{\neg\theta\} \cup S$,<br>        if $\neg(\varphi \wedge \psi) \in S$, $\neg\varphi \notin S$ and $\neg\psi \notin S$, $\theta \doteq \varphi$ or $\theta \doteq \psi$<br>$S \longrightarrow_{\neg\neg c} \{x : C\} \cup S$,<br>        if $x : \neg\neg C \in S$, $x : C \notin S$<br>$S \longrightarrow_{\sqcap} \{x : C, x : D\} \cup S$,<br>        if $x : C \sqcap D \in S$, $x : C \notin S$ or $x : D \notin S$<br>$S \longrightarrow_{\neg\sqcap} \{x : \neg E\} \cup S$,<br>        if $x : \neg(C \sqcap D) \in S$, $x : \neg C \notin S$ and $x : \neg D \notin S$<br>        $E \doteq C$ or $E \doteq D$<br>$S \longrightarrow_{=} \{x : C\} \cup S$<br>        if $C = \top \in S$, $x$ occurs in $S$, $x : C \notin S$ |

Table 3: Local $\mathcal{ALC}$-rules

| |
|---|
| $S \longrightarrow_{\neq} \{v : \neg C\} \cup S$,<br>        if $C \neq \top \in S$, there exists no $y$ with $y : \neg C \in S$,<br>        $v$ is the first new variable<br>$S \longrightarrow_{\exists} X \cup S$,<br>        if $x : \exists R.C \in S$,<br>        $X = \{v : C\} \cup \{v : \neg D \mid x : \neg\exists R.D \in S\} \cup \{v : E \mid E = \top \in S\}$,<br>        $v$ is the first new variable,<br>        $v$ is not blocked in $S$, that is to say that there is no $z$ in $S$<br>        such that $\{F \mid v : F \in X\} \subseteq \{F \mid z : F \in S\}$ |

Table 4: The locally generating rules

| |
|---|
| $S \longrightarrow_{\bigcirc} S'$<br>        if $S'$ is the union of the following sets:<br>        $\{a : \top \mid a \in ob(S)\}$,<br>        $\{a : C \mid (a : \bigcirc C) \in S\}$, for $a \in ob(S)$,<br>        $\{\psi \mid \bigcirc \psi \in S\}$,<br>        $\{(v_i : \top)\}$<br>        $\{v_i : C \mid (v_i : \bigcirc C) \in S\}$, for $0 < i \leq n$, where the $v_i$ are representatives from the equivalence classes obtained from the equivalence relation which identifies two variables $v_i$ and $v_j$ iff the same $\bigcirc$-concepts apply to them in $S$ (see [11] for a precise definition). |

Table 5: The globally generating rule

**Algorithm** Since the tableau calculus is a combination of the algorithms for $\mathcal{ALC}$ [6] and Wolper's tableau for temporal logic [12], a second procedure must be carried out once the tableau has been completely constructed (which means that all its branches are closed). This procedure eliminates nodes that contain a clash or an unrealized eventuality. Such unrealized eventualities can occur, when a branch is generated by always applying the until-rule to $\varphi\mathcal{U}\psi$ in a way that does never introduce the witness $\psi$. Here is a (skeleton of) a description of the full algorithm: firstly, a labelled graph $(G, \prec', l)$ is constructed according to the tableau-rules, where $\prec'$ is the union of the rule-application-relation $\prec$ and the pre-state-blocking-relation between nodes. Then all clash containing nodes are eliminated. After that the two functions *eliminitate_rec* and *eliminate_eventuality* are applied iteratively to the resulting subset of the tableau: *eliminate_rec* eliminates all nodes without a $\prec'$-child and *eliminate_eventuality* eliminates all nodes containing an eventuality of the form $\varphi\mathcal{U}\psi$ or $x : C\mathcal{U}D$ which cannot be realized in a $\prec'$-path. This procedure terminates after finitely many steps and the formula $\varphi$ is satisfiable iff the iteration terminates with a set containing the root of $G$.

# 4 Implementation

We have a non-optimized prototype implementation of the tableau calculus described above in the programming language Lisp. Currently we work on the integration of standard optimization techniques for tableaux and on testing possible optimizations concerned with the interaction of the two phases of the algorithm:

(1) Standard optimizations: An optimization strategy from [6] which can be used for the $\mathcal{TL}_{\mathcal{ALC}}$-tableau without modification is *dependency-directed backtracking* or *backjumping*. This technique is used to jump after a clash to the most recent branching point inevitably causing the clash. Since according to the tableau algorithm, generating rules are carried out *after* branching rules, a considerable subtree of the tableau can be omitted in many cases [6]. The interpretation of nondeterministic choices as true *branchings* does not interfere with backjumping because a node having only paths leading to clashing leaf nodes would be eliminated by the final eventuality-checking procedure anyway and hence does not have to be expanded at all. *Local simplification* and *boolean constraint* propagation [6] are applicable to the tableau algorithm as well. Their usefulness will be examined empirically.

(2) Interaction between the two phases: However, the main goal of our investigation is to find out optimizations which allow us not to expand the complete tableau tree for most formulas. A possible heuristic is to apply the elimination procedure after every pre-state blocking step. That is to say, the elimination procedure is applied to every closed branch of the (possibly not completed) tableau.

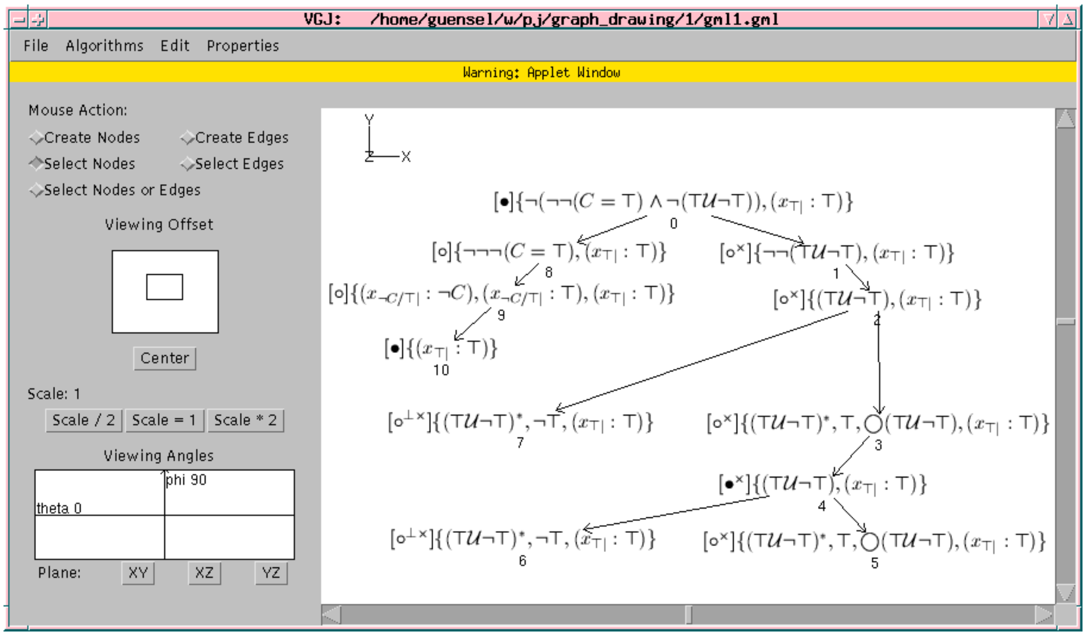


Figure 1: Example tableau visualization

If the root node is not eliminated a model has already been found. Otherwise, expanding must be continued. The choice of a good strategy for continuing the expansion as well as the decision whether the additional cost of the repeated elimination procedure is justified will depend on empirical tests we are carrying out at present.

# 5 Visualization

Parallel to the implementation work, a tool has been developed which allows to visualize tableaux as trees of formula sets. Besides, additional data like blocking-relations, clashes, eliminated nodes, and in principle any implemented property of constraint systems can be displayed.

It is possible to stop the execution of the algorithm at any time and to store a snapshot of the current data structures in a graph modelling language format. The graph's nodes are images of the attached constraint systems. A free graph drawing tool (see [9]) allows to display the generated tableau-representation.

A stepwise visualization of a tableau algorithm can be useful for didactic purposes – at present it serves as a debugging tool.

Figure 1 shows the tableau for the formula $\neg(C = \top) \vee (\top\mathcal{U}\neg\top)$ after a complete run of the algorithm. The root node and pre-states are labelled with $[\bullet]$, a $^{\perp}$ symbol indicates that the constraint system contains a clash. Eliminated nodes are marked with $^{\times}$. The root node has not been eliminated.

# References


[1] A. Artale and E. Franconi. Temporal ER modeling with description logics. In *Proc. of the International Conference on Conceptual Modeling (ER'99)*. Springer Verlag, November 1999.

[2] A. Artale and E. Franconi. A Survey of Temporal Extensions of Description Logics. In *Annals of Mathematics and Artificial Intelligence*. 2000. To appear.

[3] A. Artale and E. Franconi. Temporal description logics. In D. Gabbay, M. Fisher, and L. Vila, editors, *Handbook of Time and Temporal Reasoning in Artificial Intelligence*. MIT Press, 2001. To appear.

[4] A. Artale, E. Franconi, M. Mosurović, Frank Wolter, and Michael Zakharyaschev. Reasoning over conceptual schemas and queries in temporal databases. 2001. Submitted.

[5] Volker Haarslev and Ralf Möller. High performance reasoning with very large knowledge bases. In *International Workshop in Description Logics 2000 (DL2000)*, Aachen,Germany, 2000.

[6] I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.

[7] Ian Horrocks and Peter F. Patel-Schneider. Optimizing description logic subsumption. *Journal of Logic and Computation*, 9(3):267–293, 1999.

[8] Carsten Lutz, Holger Sturm, Frank Wolter, and Michael Zakharyaschev. A tableau calculus for temporal description logic: the constant domain case. In *Proceedings of IJCAR'2001, Lecture Notes in Computer Science*, 2001. In print.

[9] Carolyn McCreary and Larry Barowski. *The VGJ Graph Drawing Tool*, 1998. http://www.eng.auburn.edu/department/cse/research/graph_drawing/graph_drawing.html.

[10] M. Schmidt-Schauss and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1–26, 1991.

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

[12] P. Wolper. The tableau method for temporal logic: an overview. *Logique et Analyse*, 110 -111:119 – 136, 1985.

[13] F. Wolter and M. Zakharyaschev. Satisfiability problem in description logics with modal operators. In *Proceedings 6th Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 512–523, Montreal, Canada, 1998. Morgan Kaufman.

[14] F. Wolter and M. Zakharyaschev. Temporalizing description logic. In D. Gabbay and M. de Rijke, editors, *Frontiers of Combining Systems*, pages 379 – 402. Studies Press/Wiley, 1999.