A tableau based system for deciding EXPSPACE-hard temporalized description logics

Christian Günsel
Department of Computer Science, University of Liverpool, UK
Email: cg@csc.liv.ac.uk

Introduction This abstract describes an implementation of a tableau calculus for the logic $\mathcal{TL}_{\mathcal{ALC}}$ of [3, 2], 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

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

The Logic $\mathcal{TL}_{\mathcal{ALC}}$ Defining the syntax we start with concept names $C_0, \ldots$, role names $R_0, \ldots$, and object names $a_0, \ldots$. Complex concepts are $\neg C$, $C \sqcap D$, $\exists R_i.C$, $\bigcirc C$, and $C\,\mathcal{U}\,D$ (C, D being concepts and R_i being some role name). Atomic *formulas* are either of the form $C = D$ or $a_i : C$ (C, D being concepts and a_i being some object name). Complex formulas now are made up using the operators $\neg$, $\wedge$, $\bigcirc$, and $\mathcal{U}$. Note that one can for instance assign a validity period to $\mathcal{ALC}$ axioms by means of temporal operators at the formula level: $\text{TBOX}_0\,\mathcal{U}\,\text{TBOX}_1$.

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 Δ^n, $C_i^{I,n}$ are subsets of Δ^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 φ is called *satisfiable* iff there exists a model $\mathfrak{M}$ and a moment $n \in \mathbb{N}$ such that $\mathfrak{M}, n \models \varphi$. Note that we are assuming expanding domains here. While, on the one hand, constant domains ($\Delta^n = \Delta^{n+1}$ for all $n \in \mathbb{N}$) are more general and have more applications – there is, on the other hand, more hope for an efficient implementation under the expanding domain assumption, as the tableau for constant domains requires extremely expensive rules. We have proven EXPSPACE-hardness for the expanding domain assumption by modifying the proof for constant domains.

The Algorithm We briefly describe the implemented tableau calculus. For details see [2]. 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 assigning 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. Consider for example the rule

$$\begin{aligned} &S \longrightarrow_{\neg\mathcal{U}c} X \cup S, \\ &\qquad \text{if } x : \neg(C\mathcal{U}D) \in S, \{x : \neg D, x : \neg C\} \not\subseteq S, \text{ and } \{x : \neg D, x : \bigcirc\neg(C\mathcal{U}D)\} \not\subseteq S; \\ &\qquad X = \{x : \neg D, x : \neg C\} \text{ or } X = \{x : \neg D, x : \bigcirc\neg(C\mathcal{U}D)\}. \end{aligned}$$

It splits up a negated until concept, creating two successor constraint systems, one immediately satisfying $\neg(C\,\mathcal{U}\,D)$ by asserting $\neg C$ and $\neg D$, the other postponing this to the next moment.

A branch of the tableau is closed if and only if the constraint system at its leaf contains a clash (a contradiction) or, if after the application of a rule a constraint system is created which already occurs in the tree expanded so far (blocking). Since there are only finitely many subformulas of φ, the procedure terminates.

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.

After building the complete tableau, the algorithm performs in a second phase a search for unrealized eventualities – formulas or concepts of the form $\varphi\,\mathcal{U}\,\psi$ or $C\,\mathcal{U}\,D$ that have not got successor nodes realizing ψ or D respectively.

The System Our system allows to perfom a standard encoding and normalizing procedure of the input formula which is independent of the syntax used, so it can be easily adapted to e.g. less expressive logics. Programming language is Lisp, data structures are implemented in an object oriented way. There are several optimizations:

(1) Standard optimizations: An optimization strategy from [1] 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. By this means a considerable subtree of the tableau can be omitted in many cases [1]. 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 [1] are applicable to the tableau algorithm as well.

(2) Interaction between the two phases: However, the main goal of our investigation is to find out and test optimizations which allow us not to expand the complete tableau tree for most formulas. To this end, we have implemented a combination of two optimization techniques: *subset blocking* and *eventuality checking on the fly*. Every application of subset-blocking instead of equality-blocking will potentially prune the search space (because more nodes may be blocked), but in order to preserve correctness of the algorithm it is necessary to have equality-blocking in the worst case before returning *unsatisfiable*. Eventualities are checked after each blocking step – if all of them are fulfilled, the tableau expansion stops and *satisfiable* is returned immediatly.

The decision whether the additional cost of the repeated elimination procedure is justified will depend on empirical tests we are carrying out at present.

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.

At the workshop we are going to present some results about the system's behaviour for standard benchmark formulas.

References

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

[2] H. Sturm and F. Wolter. A tableau calculus for temporal description logics: the expanding domain case. *Journal of Logic and Computation*, 12(5):809–838, 2002.

[3] 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.