Analysis of Blocking Mechanisms for Description Logics

Renate A. Schmidt*
*University of Manchester
renate.schmidt@manchester.ac.uk

Dmitry Tishkovsky†
†University of Manchester
dmitry.tishkovsky@manchester.ac.uk

Description Logic (DL) formalisms have gained success in many application areas such as knowledge representation, database reasoning, multi-agent systems, semantic web, and ontology reasoning. The most explored reasoning approach for DLs is the tableau approach. Existing tableau-based DL systems essentially exploit the various kinds of the tree-model properties of DLs. It is known that the (standard) loop checking procedure is avoidable only in tableau algorithms for inexpressive DLs. Loop checking mechanisms are vital for logics with universal modalities, TBox reasoning, and/or transitive roles. However, the standard loop checking rule has various disadvantages. From a practical perspective implementation in a DL prover usually requires quite considerable effort. The standard loop checking mechanism is very sensitive to a notion of a finite type in a particular DL. Since the notion of a type varies from one logic to another, small changes in the language of the underlying DL and/or in the tableau procedure can force considerable redesign and refactoring of the code related to the loop checking. In this paper we propose and study alternative loop checking mechanisms. Our facilitation of loop checking uses inference rules which while being simple are also more general than standard loop checking mechanisms implemented in DL systems and remedy the mentioned problems.

We focus on the description logic $\mathcal{ALCIO}(\mathsf{U})$. This is an extension of the basic DL $\mathcal{ALC}$ with inverse operator on roles, nominals (or objects/ABox elements) and the top role. As the language includes nominals and the universal modality, concept satisfiability with respect to (non-empty) TBox and (non-empty) ABox can be reduced to plain concept satisfiability in $\mathcal{ALCIO}(\mathsf{U})$, i.e. with respect to empty knowledge base (TBox and ABox). $\mathcal{ALCIO}(\mathsf{U})$ is equivalent to modal hybrid logic with nominals, converse modalities and universal modality, i.e. modal tense logic with nominals and universal modality.

In general, *standard loop checking* in the tableau algorithm for $\mathcal{ALCIO}(\mathsf{U})$ works as follows. Let Σ be a finite set of concepts closed under subconcepts. Roughly, a *type* (or Hintikka element) τ_i of a nominal i with respect to Σ is a maximal locally consistent set of concepts from Σ which hold at i. For every nominal i in the current branch a tableau algorithm fills a type τ_i by using local inference rules and box propagation inference rules. Once a type τ_i is completed the nominal i is marked ready for the loop checking. Two nominals i and j in the tableau branch are regarded equal if their types coincide. In this case one of the nominals is forbidden for further tableau expansion and can be discarded together with the set of associated concepts.

For any tableau calculus T and any concept C we denote by $T(C)$ a complete tableau tree built in the system T from the concept C as input, i.e. we assume that all branches are expanded and all rules from T are applied in $T(C)$ if they can be applied. A branch of a tableau is *closed* if a contradiction has been detected in this branch, otherwise the branch is called *open*. $T(C)$ is *closed* if all the branches of it are closed and is *open* otherwise. We say that T is *terminating* iff $T(C)$ is finite for every concept C. T is *sound* iff C is unsatisfiable whenever $T(C)$ is closed for all concepts C. T is *complete* iff for any concept C, C is satisfiable (has a model) whenever $T(C)$ is open.

Suppose T denotes a sound, complete and terminating tableau calculus with standard loop checking for some DL L. Suppose that loop checking cannot close the branch in a T-tableau, i.e. every detected loop in any T-tableau is a satisfiable ('good') loop. Let T_0 be the subcalculus of T without any provision for loop checking. T_0 is sound because it is a subcalculus of a sound calculus. If T_0 on an input concept expression C constructs an open (possibly infinite) branch then T terminates with an open branch as well and, hence, C has a model. Further, because the application of loop checking cannot close the current branch of a tableau, we obtain that if an input concept expression C is unsatisfiable then a contradiction can be detected after a finite number of steps already within the system T_0, i.e. without needing any loop checking. Thus, T_0 is a sound and complete calculus for L which possibly is not terminating but T_0 always terminates (produces a finite tableau tree) for unsatisfiable concepts.

Suppose T now denotes a sound, complete and terminating calculus for $\mathcal{ALCIO}(\mathsf{U})$, and suppose T_0 is the subcalculus of T with the only omission of loop checking from T. Let us consider the following *unrestricted blocking* tableau rule:

$$\frac{}{i = j \mid i \neq j} \qquad \text{(ub)}$$

where i and j are nominals in the current branch. Let T' extend the tableau calculus T_0 with the additional rule (ub).

Theorem 1 *T' is a sound, complete, and terminating tableau calculus for $\mathcal{ALCIO}(\mathsf{U})$.*

Proof. T' is sound because T_0 is sound and the rule (ub) is sound. Suppose C is a satisfiable concept. Then $T(C)$ is open. Let B be an open branch in $T(C)$. Replacing the application of standard loop checking to nominals i and j in $T(C)$ with the application of the unrestricted blocking rule (ub) to the same nominals (and completing the right branch of the two new branches created if necessary) we obtain an open T'-tableau $T'(C)$ for C. Consider the leftmost branch B' with respect to such an application of the rule (ub) in a part of $T'(C)$ corresponding to B under the above transformation. Obviously, this branch B' coincides with B and, hence, $T'(C)$ is open and terminating. Thus, T' terminates on any satisfiable input concept C. In the case when C is unsatisfiable it is clear that a closed tableau will be constructed and that the rule (ub) cannot corrupt the ensured termination of the T_0-tableau on C. □

Thus, the unrestricted blocking rule can be considered as a replacement of standard loop checking mechanisms. It is worth noting that this equivalence is not true if an implementation of the considered tableau calculus T' use an unfair strategy. Indeed, it is easy to find an unfair strategy such that T' does not terminate on the unsatisfiable concept $\forall \mathsf{U}.(\exists R.A) \sqcap \exists \mathsf{U}.(\forall R.\neg A)$, where U denotes the top role. Observe that the proof assumes only the existence of a sound and complete calculus with loop checking and can therefore be proved for numerous other DLs. It follows from the theorem that any fair tableau procedure based on T' is a sound and complete decision procedure for $\mathcal{ALCIO}(\mathsf{U})$.

Using the unrestricted blocking rule has the following advantages over standard loop checking as implemented in current tableau-based DL systems.

- It is conceptually very simple and easy to implement.
- It is universal and does not depend on the notion of a type.
- It is versatile and enables more controlled model construction in a tableau procedure. For instance, it can be used to construct smaller models for a satisfiable concept and, in particular, it can be used to generate domain minimal models.
- It generalises to other logics, including full first-order logic.
- It can be simulated in first-order logic provers.

A disadvantage of the (ub) rule is that as a variant of the cut rule it can cause a lot of branching in a tableau. A possible constraint on nominals i and j which can reduce the number of application of the rule is that the nominal j has to be a successor of the nominal i, i.e. in the common branch of i and j the nominal j is obtained from i as a result of some sequence of applications of structural (diamond) rules. With this constraint the rule corresponds exactly to the first-order blocking rule invented by Baumgartner and Schmidt (2006). This instance of the rule is called the *blocking* rule.

The unrestricted blocking rule is implemented as a plug-in addition in the METTEL tableau prover (see Hustadt et al., 2006), and has been tested on various DLs. METTEL is a flexible system implemented in Java. It is intended for easy and quick implementation of tableau algorithms for a large class of logics (originally, for logics of metrics and topology). It provides a core management for tableau algorithms which are attached to a system as collections of tableau rules. All tableau rules in METTEL are regarded as separate plug-ins which, in most cases, can be implemented very fast and with minor effort due to the very natural representation of rules via the core METTEL classes.

References

P. Baumgartner and R. A. Schmidt. Blocking and other enhancements for bottom-up model generation methods. In U. Furbach and N. Shankar, editors, *Automated Reasoning: Third International Joint Conference on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 125–139. Springer, 2006.

Ullrich Hustadt, Dmitry Tishkovsky, Frank Wolter, and Michael Zakharyaschev. Automated reasoning about metric and topology (System description). In Michael Fisher, Wiebe van der Hoek, Boris Konev, and Alexei Lisitsa, editors, *Proceedings of the 10th European Conference on Logics in Artificial Intelligence (JELIA'06)*, volume 4160 of *Lecture Notes in Artificial Intelligence*, pages 490–493. Springer-Verlag, 2006.