

A Tableaux-based Mobile DL Reasoner

F. Müller and M. Hanselmann and T. Liebig and O. Noppens
Ulm University, D-89069 Ulm, Germany
{felix.mueller|michael.hanselmann}@uni-ulm.de

This is a preliminary experience report about designing and implementing a tableaux-based DL reasoner suitable to run on a mobile device. For now the reasoning system only offers pure TBox subsumption. Currently there is no pre-processing for efficient taxonomy computation nor ABox support. The reasoner supports the DIG standard for client communication.

Nowadays most mobile devices are capable of running Java programs using the Micro Edition of the Java 2 platform (J2ME). In order to gain valuable experience in the course of developing a mobile DL reasoner we took the most ambiguous challenge by choosing the greatest common environment, namely pure J2ME, on very limited devices, namely mobile phones.

Our approach is based on the well-known tableaux algorithm which is used in most modern DL reasoning systems. This has the advantage of starting with a relatively simple base algorithm which can be successively extended in order to support additional language constructs. The mobile reasoner currently can handle *ALCN* KBs with unique definitions.

Experience has shown that a wide range of optimization techniques typically improve the performance of a naive tableaux implementation by orders of magnitude [1]. The most important ones are lazy unfolding, absorption and dependency directed backtracking. Not all of them are relevant within the current implementation due to the restricted expressivity of the supported language (e. g. absorption can be dropped in absence of GCIs).

Within our mobile setting of very restricted memory and processing resources it is not only important to utilize those optimizations but to implement them in a resource-saving way. In this respect, the right choice of lean data structures is of exceptional importance. Our solution consists of an array based index model in order to implement an extended lazy unfolding technique called *normalizing* and *naming/tagging* (e.g. see [2]) very efficiently. Naming means that all concept expressions and sub-expressions are recursively indexed with an unambiguous identifier. In order to save memory we use integers as identifiers for concepts as well as expressions in contrast to strings or even Java objects. For example, an expression $D \sqcap \forall r.E$ may be successively named as follows: $3 \rightarrow (D \sqcap \forall r.E)$,

$4 \rightarrow D$, and $5 \rightarrow \forall r.E$ (E will then be named within a recursive call). Note that before introducing a new name the algorithm has to look-up whether this expression has already been named. A negated expression will receive a negative integer. All named TBox axioms, sub-axioms and elements have an associated integer matrix encoding their definition. Consider the TBox axiom $A \equiv D \sqcap \forall r.E$ and $6 \rightarrow E, 1 \rightarrow A$. As a result of the naming phase the following integer arrays (among others) are generated: for 1 (resp. A): $\boxed{k_n \mid 4 \mid 5}$ where k_n is an integer which encodes that the following entries are conjunctively connected. 5 ($\forall r.E$) in turn is represented with $\boxed{k_m \mid 6}$ where k_m encodes a universal quantification over a particular role, namely r (we use a special bit encoding in order to map language constructor, role, etc. into one integer). It can easily be seen that this reflects the original axiom by replacing the identifiers with their given expressions.

The encoding from above is of advantage for detection of obvious syntactical clashes even between complex expressions which is known to be one of the most effective optimizations [1]. For example, we can use a fast integer operation in order to detect A and $\neg A$ within a tableaux node simply by locally sorting it and checking whether there are two succeeding elements whose sum is 0.

Our implementation successfully runs using the freely available Sun Wireless Toolkit emulator software as well as on real mobile phones. In order to test our reasoner we added a DIG/1.1 interface which allows to use Protégé as a front-end within the emulator setting. For linking Protégé with the reasoner running on a real phone we developed a small desktop application which serves as a proxy. More precisely, the proxy receives HTTP DIG messages from a client and will pass those via Bluetooth to the mobile reasoner.

To our knowledge, there is no related work except Pocket KRHyper [3] a port of a subset of the hyper tableau calculus to a mobile phone. Pocket KRHyper covers unfoldable \mathcal{ALCI} TBoxes extended by role hierarchies using a variant of the KRSS syntax. As far as can be seen within the log files of the system, FaCT++ seems to utilize a similar internal naming approach.

References

- [1] Ian Horrocks. Applications of Description Logics: State of the Art and Research Challenges. In *Proc. of the 13th Int. Conf. on Conceptual Structures (ICCS'05)*, pages 78–90, 2005.
- [2] Ian Horrocks and Peter Patel-Schneider. Optimising Description Logic Subsumption. *Journal of Logic and Computation*, 9(3):267–293, June 1999.
- [3] Alex Sinner and Thomas Kleemann. KRHyper – In Your Pocket. In *Proc. of the International Conference on Automated Deduction (CADE-20)*, volume 3632 of *LNCS*, pages 452–458. Springer Verlag, 2005.