Towards Mobile Reasoning

Thomas Kleemann

http://www.kleemann-online.com
tomkl@kleemann-online.com

Abstract. Highly optimized reasoning support for Description Logics (DLs) has been developed during the past years. This paper presents our efforts to develop a reasoner suitable for mobile devices, namely mobile phones.
This might become important upon the convergence of the mobile industry and the web, especially the semantic web. Details of the implementation and current limitations provide explanations to the differences in performance in comparison to one of the well-known desktop reasoners.

1 Motivation

Mobile phones have become an ubiquitous companion for many users. There are western countries with more mobile contracts than inhabitants. The mobile industry is pushing users to adopt modern 3G services, i.e. to use the mobile phone as a client to messaging and news services. At the dawn of the semantic web we expect these messages to be semantically annotated. The device should be able to sort out unwanted messages and reduce the penetration of the user to messages of interest. We suggested a user profile and matchmaking in [6, 8] that copes with SPAM annotations. More DL based approaches to matchmaking can be found in [9, 10].

Unlike [15] we consider privacy a major issue. Due to the nature of user profiles, we would like to encourage users to keep their private data on their personal device. This requires the mobile device to decide the matchmaking without support of external reasoners. As a consequence we developed the reasoner Pocket KRHyper [7] that runs on a mobile phone or similar devices.

The remainder of this paper presents some details of the software and hardware environment, explains how the DL reasoning is performed, finally gives some figures on performance compared to RacerPro 1.9[1] that is one of the highly developed desktop reasoners.

2 Environment

The environment of mobile phones poses several restrictions on the development of the reasoner. Resources like computing power, memory and energy are tightly limited. Although phones offer hundreds of MB for the storage of pictures, sounds, and ringtones the memory offered to applications is restricted to a few hundreds kB. This is a hard limitation imposed by the manufacturer, that cannot be configured. Even

[1] http://www.racer-systems.com

worse this small memory is used for the application code, the application data like the ontology and the working set of the current computation. Consequently whenever we had the choice between memory consumption or additional computation, we went for the additional computation. Computing power is reduced by a factor of 50 to 100 compared to an average desktop PC.

The widest spread language on phones is JAVA 2 micro edition (J2ME)[2]. J2ME is tailored for devices with limited resources. It offers a language comparable to standard JAVA 1.1.8. Unfortunately all modern container classes of JAVA2, that would ease development and speed up execution, are missing.

3 Reasoning

The reasoning is based on a first-order model generating reasoner implementing the hyper-tableaux calculus [1]. The use of a first-order tableaux may be a little surprise, but turns out to be viable, as can be seen in section 4. The reasoning task offers a timelimit and throws *Exceptions* upon timing out or running into memory shortages. Thus the application may react appropriately. The reasoner comes as a library to be integrated into J2ME applications (midlets).

3.1 Translation

To use the FO reasoner for DL purposes the knowledge base is transformed into a set of possibly disjunctive clauses. The approach is relational and preserves the open world assumption. Although the result is a logic program, the restrictions of DLP [3] do not apply.

The knowledge base is considered to be a finite set of axioms $C \sqsubseteq D$ and $C \equiv D$, where C, D are concepts of the DL ALC extended by inverse and transitive roles and role hierarchies.

The transformation into sets of clauses introduces subconcepts to reduce the complexity of the concept expression or axiom. As an example the axiom $\exists R.C \sqsubseteq \forall S.\forall T.D$ is decomposed into $\exists R.C \sqsubseteq sub_i$ and $sub_i \sqsubseteq \forall S.sub_j$ and $sub_j \sqsubseteq \forall T.D$ to comply with the transformation primitives. Table 1 gives the transformation primitives in abstract DL syntax, a corresponding first order formula, and the generated clauses.

The clauses marked with * share variables in a disjunctive head, they are not range-restricted. As an extension to previous work [12] Pocket KRHyper now handles these clauses by generating all ground instances up to the given term weight. Doing so often causes timeouts of the prover due to the limited resources. Clauses marked with ** are suitable for reasoning tasks in acyclic terminologies. Decidability commonly requires the tableau procedure to engage a blocking technique. The blocking techniques found in [4] may be adapted to the transformation as shown in [2].

[2] http://java.sun.com/j2me/

description logic	first order formula	clauses
$C \sqcap D \sqsubseteq E$	$\forall x.C(x) \wedge D(x) \rightarrow E(x)$	e(x) :- c(x), d(x).
$C \sqcup D \sqsubseteq E$	$\forall x.C(x) \vee D(x) \rightarrow E(x)$	e(x) :- c(x). e(x) :- d(x).
$C \sqsubseteq \neg D$	$\forall x.C(x) \rightarrow \neg D(x)$	false :- c(x), d(x).
$\exists R.C \sqsubseteq D$	$\forall x \forall y.R(x,y) \wedge C(y) \rightarrow D(x)$	d(x) :- c(y), r(x,y).
$\forall R.C \sqsubseteq D$	$\forall x.(\forall y.R(x,y) \rightarrow C(y)) \rightarrow D(x)$	d(x); r(x,f_{R-C}(x)). * d(x) :- c(f_{R-C}(x)).
$C \sqsubseteq D \sqcap E$	$\forall x.C(x) \rightarrow D(x) \wedge E(x)$	e(x) :- c(x). d(x) :- c(x).
$C \sqsubseteq D \sqcup E$	$\forall x.C(x) \rightarrow D(x) \vee E(x)$	e(x); d(x) :- c(x).
$\neg C \sqsubseteq D$	$\forall x.\neg C(x) \rightarrow D(x)$	c(x); d(x). *
$C \sqsubseteq \exists R.D$	$\forall x.C(x) \rightarrow (\exists y.R(x,y) \wedge D(y))$	d($f_{R-D}(x)$) :- c(x). r(x,$f_{R-D}(x)$) :- c(x). **
$C \sqsubseteq \forall R.D$	$\forall x.C(x) \rightarrow (\forall y.R(x,y) \rightarrow D(y))$	d(y) :- c(x), r(x,y).
$R \sqsubseteq S$	$\forall x \forall y.R(x,y) \rightarrow S(x,y)$	s(x,y) :- r(x,y)
$R^{-} \equiv S$	$\forall x \forall y.R(x,y) \leftrightarrow S(y,x)$	s(y,x) :- r(x,y). r(x,y) :- s(y,x).
R^{+}	$\forall x \forall y \forall z.R(x,y) \wedge R(y,z) \rightarrow R(x,z)$	r(x,z) :- r(x,y), r(y,z).

Table 1. Translation Primitives

The effective test for satisfiability of a concept C inserts a single fact $C(a)$ into the knowledge base. C is satisfiable if the Pocket KRHyper finds a model for the knowledge base. A refutation indicates the unsatisfiability of the concept. Subsumption is reduced to satisfiability.

3.2 Optimizations

Common to DL reasoners are many optimizations [5]. In the following section some counterparts within the FO reasoner are presented.

Backjumping. Backjumping is a technique to reduce the unnecessary evaluation of branches, that do not contribute to the clash in a branch. In hyper-tableaux the 'hyper' eventually tracks back over multiple branching points to avoid the same.

Semantic Branching. This optimization is derived from DPLL. As hyper-tableaux is also a descendant from DPLL a corresponding technique known as complement splitting is in use. Semantic branching adds the complement of already inspected branches to subsequent branches. This avoids the redundant inspection of clashed terms. Different from semantic branching the complements of subsequent branches are added to the branches, that are inspected first. Because of the symmetry the effect is the same for cases where all branches are closed.

Boolean Constraint Propagation. The current mobile implementation of the hyper-tableaux tries to choose the disjunctive clause for the next expansion, that will immediately clash one of its branches. This is similar to BCP.

Lazy Unfolding. Beyond these optimizations a behavior similar to lazy unfolding is introduced by the transformation step. All subexpressions are named, and the transformation step tries to identify with already known subexpressions. The FO tableaux will expand these subexpressions only if no clash has been found prior to expansion.

Absorption. Absorption is left to the preprocessing of the knowledge base. It absorbs general concept inclusions (GCI) into the definition of concepts. For a DL tableaux this reduces the number of disjunctive branches. Here absorption reduces the number of cases, that require the enumeration of all ground instances, see section 4 for an example.

Reducing Memory Consumption. Currently the J2ME environment does not offer an preinstalled XML-parser. So we decided to use the lisp-like KRSS [11] syntax. The lisp-parser is significantly smaller than an XML-parser.

The clausal knowledge base is partitioned. The knowledge base keeps track of these parts by setting a mark at each such step, so it is possible to revert it to the state it had before an update or query. This allows for the addition of clauses generated for query processing and a subsequent removal of them. The repeated transformation of unchanged DL axioms to clauses is avoided. The query related clauses are added and retracted as required by the query processing.

Any form of indexing or caching requires additional memory. Because the memory constraints of the used platforms are tight, we dropped most of these enhancements. Consequently the current solution does not scale (wrt. the size of the model to search) as well as desktop reasoners.

3.3 Engineering

The development of a reasoning engine is a challenging and error-prone task. To cope with these difficulties we adopted the idea of unit testing. Unit testing automates the execution of (parts of) your software on predefined tests with known results. The ease of use and high degree of automation enables the early detection of software glitches and sideeffects of changes throughout the whole development cycle.

The tests are divided into a first-order part and a DL part. For the first-order reasoning kernel we used a subset of the TPTP [13]. These clausal problems are fed into the reasoner and evaluated by the FO tableaux. The result of the reasoning process is compared to the known result.

Starting with some ’benchmarks’ taken from the DIG page[3] many own tests, that focus on specific implementation details, were added to test the DL part. The results in the following section are derived from scaled tests. The automation of the tests includes the comparison of predefined results with the actual result of the reasoner.

[3] http://dl.kr.org/dig/

test	knowledge base	comments
1	$C \equiv C_1; C_i \sqsubseteq C_{i+1}$	test the propagation of individuals and the ability to handle long branches
2	$C \equiv C_1; C_i \sqsubseteq C_{i+1}; C_{m+1} \sqsubseteq \bot$	like test1 but not satisfiable; test refutations on long branches
3	$C \equiv \exists R.C_1; C_i \sqsubseteq \exists R.C_{i+1}$	test the handling of role successors
4	$C \equiv \exists R.C_1; C_i \sqsubseteq \exists R.C_{i+1}; C_{m+1} \sqsubseteq \bot$	like test 3, but with a refutation at the end of the branch
5	$C \equiv \exists R.C_1; \exists R.C_i \sqsubseteq \exists R.C_{i+1}$	test the handling of role successors and concept inclusions
6	$C \equiv \exists R.C_1; \exists R.C_i \sqsubseteq \exists R.C_{i+1}; \exists R.C_{m+1} \sqsubseteq \bot$	like test 5 with a closing inclusion
7	$C \equiv \exists R.C_1; \exists R.C_i \sqsubseteq \exists R.C_{i+1}; \top \sqsubseteq \forall R.\neg C_{m+1}$	like test 6 but with differently formulated closing, that derives from a range restriction
8	$C \equiv \exists R.C_1; \exists R.C_i \sqsubseteq \exists R.C_{i+1}; \top \sqsubseteq \forall R.\bot$	similar to test 7 but branches may be closed prior to the evaluation of the inclusions
9	$C \equiv \exists R.C_0; \exists R.C_i \sqsubseteq \exists R.C_{i+1}; \top \sqsubseteq \forall R.\bot$	this time the inclusions are not related to the definition of C. Still not satisfiable.
10	$C \equiv \exists R.C_0; C_{i+1} \sqcap \forall R.C_{i+2} \sqsubseteq \exists R.C_i; C_{i+1} \sqsubseteq \exists R.C_{i+2}$	this test demonstrates the effect of not range-restricted clauses for FO model generators
11	$C \equiv \exists R.C_0; C_{i+1} \sqsubseteq \exists R.C_{i+2} \sqcap (\exists R.\neg C_{i+2} \sqcup \exists R.C_i)$	Absorption in a preprocessing step solves this problem
12	$C \equiv \exists R_1.C_1; R_i \sqsubseteq R_{i+1}$	test handling of role hierarchies
13	$C \equiv \exists R_1.C_1; R_i \sqsubseteq R_{i+1}; \top \sqsubseteq \forall R_{m+1}.\bot$	test refutation at the end of role hierarchies

Table 2. Testsuite

4 Comparison

Based on the test cases for DL reasoning a generator of tests was developed. These tests are used in multiple sizes wrt. the number of axioms to offer a view into the scalability of the reasoning solution. In the primary application of Pocket KRHyper all models or refutations were computed in short branches of a tableaux (less than 25 proof steps) because the applications ontology was about 200 concepts large but not so deeply nested.

To the best of my knowledge there are no competing reasoners on mobile phones. So I decided to compare the performance with RacerPro 1.9. The JAVA-library is compiled for the desktop version of JAVA (J2SE) without changes in the code. Consequently all indexing and caching is still missing even though the desktop environment would allow for the additional space. The integration of RacerPro into the automated test routine uses its socket interface. The syntax of the test cases is compatible due to the use of KRSS.

Out of more than 50 tests the following section describes about a dozen tests to give an idea of the characteristics of Pocket KRHyper. The tests presented here are limited to satisfiability tests with satisfiable *and* unsatisfiable concepts. This is suitable to describe the behavior of subsumption tasks with positive and negative outcome. Checking for subsumption and non-subsumption is not a trivial task for all first-order reasoners, who are possibly better suited to find refutations than models [14].

4.1 Testsuite

All tests are scaled to the sizes $m \in \{2,5,10,20,50\}$. Table 2 lists the tests, where $i \in \{1,2,...m\}$ for the corresponding size. C possibly indexed are concepts, R possibly indexed are roles. All 65 tests check for the satisfiability of concept C. A JAVA application runs all tests with Pocket KRHyper and RacerPro, compares the results and collects the execution time in ms for the reasoning tasks. Some comments in Table 2 indicate the targeted feature of the test.

The automated execution of these tests led to quite stable results wrt. the time needed. Surprisingly only Racer had some glitches, where every now and then but reproducible single tests performed quite bad beyond the time-limit of 1500ms. In the calculation of the graphical representation (see Fig. 1) of the runtimes the median of five runs was taken to eliminate this phenomenon. The times for the FO reasoner include the transformation into clauses. Table 3 gives exact values, — indicates a timeout. All times are in milliseconds taken on a contemporary personal computer running WindowsXP with Java Runtime 1.5 (J2SE). Reasoners are Pocket KRHyper via the library and RacerPro 1.9 via TCP-Socket. Due to limitations in precision all runtimes below 10ms are presented as $\leq$10ms.

4.2 Evaluation

Some weaknesses of the first-order approach are quite obvious. Test 10 is never solved within the timelimit. This is a consequence of the necessary enumeration of ground instances. The only solution to this problem is an improved preprocessing as test 11 indicates. As one might expect both tests are performing well with Racer. The only surprise is the increased runtime for test 10 with size 5 compared to the same test in size 10. An additional weakness is the scalability for tests 3 and 4. Following the transformation steps, a lot of functions are used. Pocket KRHyper tries to reduce the amount of storage and compares all new facts with existing ones in the current branch. This comparison is carried out by a unification method. This method suffers from the deeply nested functions and a lack of caching of intermediate results. In

m	test reasoner	1	2	3	4	5	6	7	8	9	10	11	12	13
m=2	racer	20	10	10	10	10	10	20	10	20	10	10	10	10
	pKrh	30	≤ 10	≤ 10	≤ 10	≤ 10	≤ 10	≤ 10	≤ 10	≤ 10	—	≤ 10	≤ 10	≤ 10
m=5	racer	20	10	20	20	20	30	20	10	20	90	20	10	20
	pKrh	≤10	≤ 10	20	30	10	11	≤ 10	≤ 10	≤ 10	—	≤ 10	≤ 10	≤ 10
m=10	racer	20	30	20	20	30	30	30	30	20	50	30	30	30
	pKrh	≤10	≤ 10	90	70	30	20	40	≤ 10	30	—	≤ 10	≤ 10	≤ 10
m=20	racer	50	40	40	40	50	50	50	71	40	90	60	40	41
	pKrh	30	≤ 10	701	681	100	100	130	≤ 10	≤ 10	—	≤ 10	20	20
m=50	racer	91	100	90	90	90	100	100	90	90	241	170	90	90
	pKrh	60	20	—	—	981	1001	1402	20	20	—	140	40	60

Table 3. Runtimes in comparision of Pocket KRHyper and RacerPro 1.9

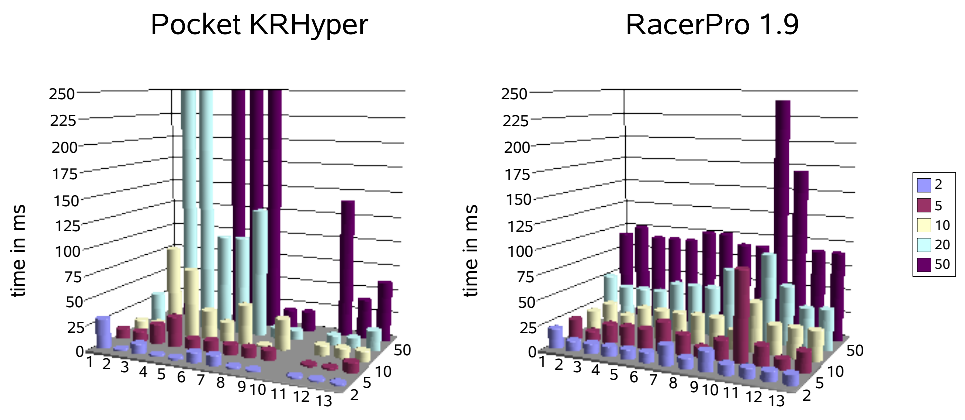

Fig. 1. Runtimes in milliseconds, max. 250ms, timelimit 1500ms

contrast Racer scales well for these two problems up to sizes between 100 and 200. Only above these limits Racer performs bad.

Besides these weaknesses the first-order model generation approach gives competitive runtimes compared to a well established DL reasoner. For small test cases (up to 10) and special cases (e.g. role hierarchies: test 12 and 13) the performance is usually better than Racer. This is not to speak of superior performance, because Racer is designed for a richer DL and has to maintain all indexes and caches, that will speed up subsequent queries into the same TBox. One of Pocket KRHyper s nice features is the symmetry of satisfiable and non-satisfiable problems or respectively subsumption testing. The lack of scalability is most of all a consequence of the dropped indexing. That was chosen deliberately because of the tightly limited memory in mobile phones.

Tests 8 and 9 demonstrate the *local* character of the FO tableaux and the transformation. The additional axioms in the knowledge base do not significantly influence the time for answering the satisfiability. This is an important property for the suitability into a resource scarce environment.

5 Conclusion

In this paper some details about the Pocket KRHyper approach to DL reasoning were presented. Although the mobile reasoner was successful in its original application a more thorough testing and comparison was due. Out of many tests 13 were described in detail. For many other features like inverse roles or explicit disjuncts the comparison revealed similar results. Overall the developed reasoner proved to be a useful tool. The need to look into very specific details of the implementation and not into the average capability or performance to solve problems motivated the disregard of existing benchmarks.

Of course Racer is not the only competing system[4]. To ease our purposes Racer supported the same syntax and I consider this a good basis for further developments. Especially because Racer is going to implement the SWRL rule language I will revisit the comparison. Rules are a natural extension for the first-order reasoner. In fact merely an interface has to be added. That was one of the reasons not to specialize the reasoning procedure for DL purposes. Furthermore the transformation is likely to include more features like normalizations and absorption in future versions. The current version of Pocket KRHyper is available for download at sourceforge[5].

Finally I would like to thank Racer-Systems for granting a license for the current version. Compared to version 1.7 many improvements make a difference in this comparison.

References

1. Peter Baumgartner. Hyper Tableaux — The Next Generation. Technical Report 32–97, Universität Koblenz-Landau, 1997.
2. Peter Baumgartner, Ulrich Furbach, Margret Gross-Hardt, and Thomas Kleemann. Model based deduction for database schema reasoning. In Susanne Biundo, Thom Frühwirth, and Günther Palm, editors, *KI 2004: Advances in Artificial Intelligence*, volume 3238, pages 168–182. Springer Verlag, 2004.
3. Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: Combining logic programs with description logic. In *Proc. of the Twelfth International World Wide Web Conference (WWW 2003)*, pages 48–57. ACM, 2003.
4. Volker Haarslev and Ralf Möller. Expressive ABox Reasoning with Number Restrictions, Role Hierarchies, and Transitively Closed Roles. In *KR2000: Principles of Knowledge Representation and Reasoning*, pages 273–284. Morgan Kaufmann, 2000.
5. I. Horrocks and P. F. Patel-Schneider. Optimising description logic subsumption. *Journal of Logic and Computation*, 9(3):267–293, 1999.
6. Thomas Kleemann and Alex Sinner. Description logic based matchmaking on mobile devices. In Joachim Baumeister and Dietmar Seipel, editors, *Proc. of 1st Workshop on Knowledge Engineering and Software Engineering - KESE2005, Koblenz*, ISSN 1860-4471, pages 37–48, 2005.
7. Thomas Kleemann and Alex Sinner. Krhyper - in your pocket, system description. In Robert Nieuwenhuis, editor, *Automated Deduction - CADE-20: 20th International Conference on Automated Deduction*, volume 3632 of *Lecture Notes in Computer Science*, pages 452–458. Springer Verlag, 2005.
8. Thomas Kleemann and Alex Sinner. User profiles and matchmaking on mobile phones. In Oscar Bartenstein, editor, *Proc. of 16th International Conference on Applications of Declarative Programming and Knowledge Management INAP2005, Fukuoka*, 2005.
9. Lei Li and Ian Horrocks. A software framework for matchmaking based on semantic web technology. In *Proc. of the Twelfth International World Wide Web Conference (WWW'2003)*, pages 331–339. ACM, 2003.
10. Tommaso Di Noia, Eugenio Di Sciascio, Francesco M. Donini, and Marina Mongiello. Abductive matchmaking using description logics. In *Proc. of IJCAI'03*, pages 337–342, 2003.
11. Peter F. Patel-Schneider and B. Swartout. Description-logic knowledge representation system specification, November 1993.
12. A. Sinner and T. Kleemann. Krhyper - in your pocket, system description. In *Proc. of Conference on Automated Deduction, CADE-20*. Springer Verlag, 2005.
13. G. Sutcliffe and C.B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.
14. Dmitry Tsarkov and Ian Horrocks. DL reasoner vs. first-order prover. In *Proc. of the 2003 Description Logic Workshop (DL 2003)*, volume 81 of *CEUR (http://ceur-ws.org/)*, pages 152–159, 2003.
15. Wolfgang Wahlster. Smartweb: Mobile applications of the semantic web. In *GI Jahrestagung (1)*, pages 26–27, 2004.

[4] see http://www.cs.man.ac.uk/˜ sattler/reasoners.html for a comprehensive list

[5] http://sourceforge.net/projects/mobilereasoner