

System Description: E-KRHyper 1.4

Extensions for Unique Names and Description Logic

Markus Bender, Björn Pelzer, and Claudia Schon

Universität Koblenz-Landau, Institut für Informatik, 56070 Koblenz, Germany
{mbender, bpelzer, schon}@uni-koblenz.de

Abstract. Formal ontologies may go beyond first-order logic (FOL) in their expressivity, hindering the usage of common automated theorem provers (ATP) for ontology reasoning. The Unique Name Assumption (UNA) is an extension to FOL that is valuable for ontology specification, allowing the definition of distinct objects. Likewise, the Description Logic *SHIQ* is a popular language for knowledge representation (KR). This system description provides details on the extension of the prover E-KRHyper by the ability to handle both the UNA and *SHIQ*. This ATP was developed for embedding in KR applications and hence already equipped with special features and extensions to FOL, making it natural to add the new capabilities in E-KRHyper version 1.4. We report on the theory, the implementation and also the evaluation results of the new features.

1 Introduction

The expressivity of first-order logic (FOL) is often insufficient when specifying ontologies. This has led to logic extensions like the *Unique Name Assumption* (UNA), which allows expressing that two constants are semantically distinct. Going further than merely extending FOL, Description Logics provide entire logical languages intended especially for knowledge representation (KR). Unfortunately automated theorem provers (ATP) are commonly restricted to FOL, which limits their usefulness for ontology reasoning. We have therefore modified our prover *E-KRHyper*, giving it the capability to handle both the UNA and the popular Description Logic *SHIQ*.

E-KRHyper (*Knowledge Representation Hyper Tableaux with Equality*) [31] is an ATP and model generation system for FOL with equality. It is an implementation of the *E-hyper tableau calculus* [12], which integrates a superposition-based handling of equality [4] into the hyper tableau calculus [11]. Like its predecessor, the original KRHyper [39] without equality, E-KRHyper has been designed for embedding in knowledge representation applications, see Section 2 for details.

E-KRHyper forms the reasoning component of the deduction-based question answering system *LogAnswer* [17,18]. The prover is also used in the controlled language processor *PENG Light* [35], and it is part of the *HETS* framework [26] for formal methods integration and proof management. (E-)KRHyper has also been used as an embedded knowledge processing engine in applications like content composition for e-learning [6,9], document management [8], database schema processing [7], semantic information retrieval [5], ontology reasoning [14], and planning [13]. An excerpt has been ported to mobile devices for user profile matching [36].

Given this background we wish to keep improving E-KRHyper’s abilities in ontology reasoning, so it was natural to soundly integrate the UNA [15] and *SHIQ* [16] both on the formal level and in the implementation. In this system description we provide details on the current version 1.4 of E-KRHyper, focussing on the new extensions, and we evaluate the system on suitable benchmarks.

2 General Overview of E-KRHyper

E-KRHyper 1.4 is written in OCaml [24] and compatible with POSIX-compliant operating systems and environments. It is available under the GNU GPL at [29]. As E-KRHyper is geared towards embedding, it has been designed with operational flexibility: like most ATP systems it can be invoked with file-based input, but it also features an interactive mode in which it can communicate with applications or the user over STDIN and STDOUT or other user-specified channels. This way E-KRHyper can remain operational over multiple reasoning tasks, with the logical input being modified in-between, for example in order to test different conjectures on a common knowledge base without having to reload the latter each time [30,19].

The prover accepts input in *PROTEIN* [10] and *TPTP* syntax [38] for FOL. A built-in clausifier automatically converts formula expressions into CNF. To deal with very large input theories E-KRHyper includes axiom selection methods like the SINE algorithm [21]. Input may go beyond the expressivity of FOL with equality: E-KRHyper supports stratified negation as failure [1], arithmetic evaluation as well as operations on list and set constructs. Special interface literals allow the prover to access external knowledge sources like web services or databases.

The proof procedure involves the construction of an E-hyper tableau, which is a tree labelled with clauses. A branch may be extended by applying an inference rule to its clauses and then appending the conclusion as a new leaf node. A branch is closed once it contains the empty clause. Positive disjunctions can split branches. Ground substitutions prevent sharing of variables between disjunct branches. This avoids the issue of rigid variables, and it allows E-KRHyper to build the tableau depth-first, working only on one branch at a time. Refutational completeness and a fair search control are ensured by an iterative deepening strategy bounded by the term weights of clauses. A comprehensive set of simplification and deletion rules helps the prover to eliminate redundant clauses and to avoid unnecessary inferences. If all branches are closed, E-KRHyper can output a proof consisting only of the essential inference steps. On the other hand, a branch that can be neither closed nor extended represents a model and proves the input satisfiable. E-KRHyper can then enumerate models by computing additional branches. The derivation process can also be refined to allow the computation of minimal models.

3 Extensions in Version 1.4

3.1 Support for the Unique Name Assumption

In automated theorem proving there are problems that need information on the inequality of certain constants [2,20,3]. In most cases such information about any two distinct constants c_1 and c_2 is provided by explicitly adding inequality facts of the form

$false \leftarrow c_1 \simeq c_2$ to the knowledge base. As the number of these facts grows quadratically in the number of constants, they clutter the knowledge base and distract human readers of the problem from its actual proposition. Additionally it is safe to assume that a larger knowledge base reduces the performance of a theorem prover in many cases, which is another drawback of explicit inequality facts.

To avoid such a blow-up, native handling of inequality of constants can be used, which removes the need for inequality facts. This can be done by using the *Unique Name Assumption* (UNA), which states that two constants are semantically equal if and only if they are syntactically equal. Instead of forcing the UNA onto all constants, we apply it on a subset of the constants, called the *distinct object identifiers* (DOI). This is a more flexible approach and mimics the TPTP's way of dealing with the UNA [38,37]. We also use the TPTP's syntax for defining DOI, i.e. a constant enclosed in double quotes is treated as a DOI by E-KRHyper.

E-KRHyper's underlying calculus was extended in a sound and correct way to allow native handling of DOI and those changes have been implemented into E-KRHyper accordingly [15]. The most significant change was the introduction of two rules to deal with two kinds of formulae that are important if DOI are used: *unit contradictions*, of the form $i \simeq j \leftarrow$, and *object tautology clauses*, of the form $\mathcal{A} \leftarrow i \simeq j, \mathcal{B}$, where i and j are two non-identical DOI, $\mathcal{A}[\mathcal{B}]$ is a possibly empty set of head [body] literals.

An unit contradiction is a clause that is false by definition, so the modified version of E-KRHyper closes a branch that contains a unit contradiction. On the opposite site, an object tautology clause is always true and thus can neither contribute to deriving a contradiction nor a model. Therefore object tautology clauses are deleted to prevent unneeded inference steps on the clause and to keep the set of clauses as small as possible.

This implicit handling of non-identical constants makes the problems easier to comprehend and reduces the execution time of reasoning.

3.2 Support for *SHIQ*

SHIQ is a description logic (DL) with widespread usage. It is decidable and features both inverse and transitive roles as well as qualified cardinality restrictions. Thanks to this it has become the formal foundation of OIL (Ontology Inference Layer), the ontology infrastructure of the Semantic Web [23]. Several dedicated DL-reasoners support *SHIQ*, such as *Pellet* [28] and *HermiT* [27]. The latter even employs a modified version of our own hyper tableau calculus. Adapting E-KRHyper to *SHIQ* was therefore an obvious way for us to enhance the ontology reasoning abilities of our prover.

E-KRHyper accepts *SHIQ* converted into *DL-clauses* [27], which are basically FOL-clauses extended by the possibility of having special positive *at-least* literals which represent qualified number restrictions. For example, we can express that a car has at least three wheels with the *SHIQ*-axiom $car \sqsubseteq (\geq 3 \text{ has.wheel})$, which translates into the DL-clause $(\geq 3 \text{ has.wheel})(x) \leftarrow car(x)$. Hence an *at-least* literal has the form $(\geq n R.C)(x)$, with $n \geq 1$ and where R is a binary predicate symbol (representing the DL-role R), C is a unary (and possibly negated) predicate (representing the DL-concept C), and x is a variable (to be instantiated by ground terms that represent DL-individuals). E-KRHyper has been modified to handle such DL-clauses, and a corresponding *at-least* inference rule has been added to the calculus to evaluate the new literals:

$$at\text{-}least \frac{(\geq n \ R.C)(a) \leftarrow}{\bigcup_{1 \leq i \leq n} \{R(a, b_i) \leftarrow\} \cup \bigcup_{1 \leq i \leq n} \{C(b_i) \leftarrow\} \cup \bigcup_{1 \leq i < j \leq n} \{\leftarrow b_i \simeq b_j\}}$$

Thus given as a premise a positive *at-least* unit instantiated by a ground term a , the rule creates n fresh ground terms $b_1, \dots, b_n$ (the *R*-successors of a) and derives unit clauses for these new individuals, ensuring that they satisfy the predicates R and C and that they are pairwise distinct. The derived set is considered a conjunction, so all its clauses are attached to the current branch as a segment of $2n + \binom{n}{2}$ nodes. To ensure termination it may be necessary to block an application of the *at-least* rule, to prevent it from generating useless successors for a given individual. For this we use a modified version [16] of *pairwise anywhere blocking* [27].

As an optional optimization E-KRHyper can use its UNA functionality (see Section 3.1): instead of ensuring the distinctness of successors via explicit inequalities, the fresh $b_1, \dots, b_n$ created by one application of the *at-least* rule can be treated as a special type of DOI which only count as distinct with respect to each other, but which are treated as normal terms otherwise. For example, $b \simeq b' \leftarrow$ for some generated successor terms b and b' is only a unit contradiction if both b and b' have been created in the same *at-least* inference step.

4 Related Systems and Performance Evaluation

We have evaluated several aspects of E-KRHyper: the performance on FOL-problems in general, and specifically the performance of the two new extensions, namely the UNA implementation and the *SHIQ* capability. All tests were carried out on a computer featuring an Intel Core 2 Quad (Q9550) @ 2.83GHz and 4GB PC2-6400 RAM.

For the general evaluation we tested E-KRHyper on the 15,560 FOL problems of the TPTP v5.4.0, the most current release at the time of this writing. The prover was limited to one CPU core, 1GB of memory and 300 seconds for each problem. E-KRHyper solved 5,970 problems, corresponding to 38.4% of the test set.¹ The TPTP rates the difficulty of problems from 0.0 for the easiest to 1.0 for the hardest, the latter reserved for problems not solved by any state-of-the-art prover. The hardest problems solved by E-KRHyper were *PUZ050-1* (rated at 0.94), *SWW284+1* (0.93) and *LCL650+1.020* (0.92). Overall E-KRHyper has a general performance that is slightly above average among the provers listed at the TPTP website [37]. In comparison, the prover *Otter* [25], which usually serves as a benchmark in the ATP community, solves 27% under similar conditions, whereas top-ranking systems like *Vampire* [32] and *E* [33] have a significant lead with 66% and 61% respectively.

4.1 Unique Name Assumption

In order to evaluate the UNA implementation we used a variation of the synthetic benchmarks STORECOMM (SC) and STORECOMM-INVALID (SCI) [2], situated in the

¹ The TPTP v5.4.0 contains merely two FOL problems with DOIs and none with DL-features, so the prover modifications described in this paper lead only to a negligible improvement in the TPTP performance compared to earlier versions of E-KRHyper.

theory of arrays. A test case from SC is the task to show that two given permutations of unique store operations on an array result in the same array. A test case from SCI is the task to show that two given sequences of unique store operations that differ in at least one stored element at an index do not result in the same array. This definition of SCI differs from the definition in [2] and was chosen for technical reasons. We continue to call it STORECOMM-INVALID or SCI in the context of this work. The term *unique store operation* states that each index of an array is written to exactly one time.

As the theory of arrays is not natively supported by E-KRHyper, we axiomatize the theory of arrays by using the following axioms:

$$sel(sto(A, I, E), I) = E \quad (1)$$

$$sel(sto(A, I, E), J) \Leftarrow I \neq J \quad (2)$$

$$A = B \Leftarrow sel(A, I) = sel(B, I) \quad (3)$$

The function $sel : ARRAY \times INDEX \rightarrow ELEMENT$ returns the element that is stored at the given index of the given array, and the function $sto : ARRAY \times INDEX \times ELEMENT \rightarrow ARRAY$ returns an array that is constructed by storing a given element at the given index of a given array.

To create a test case, four parameters are needed: a list $p = 0, \dots, n-1$, a permutation of this list, called q , a flag v that indicates if we want to generate a test case for SC or SCI and a flag d that indicates if this test case uses distinct object identifiers or not.

Independent of the chosen parameters every test case contains the three axioms that describe the theory of arrays. Additionally every test contains n unique predicates of the form $index(i_x)$ with $0 \leq x < n$ introducing the constants that represent the arrays' indices, which is needed for technical reasons. If distinct object identifiers are used, these predicates look like $index("i_x")$, as constants in quotes denote DOI.

If no distinct object identifiers are used, we need to express that all indices are distinct, which is done by introducing $\binom{n}{2}$ unique predicates of the form $false :- i_x = i_y$ with $(x, y) \in C_2^n$, where C_2^n is the set of 2-combinations over $\{0, \dots, n-1\}$. Additionally we need to express that all elements are distinct, which is done by introducing $\binom{n}{2}$ unique predicates of the form $false :- e_x = e_y$ with $(x, y) \in C_2^n$.

The actual property to be proven is then added by the equality predicate $T_{n,v,d}(q) = T_{n,v,d}(p)$, where $T_{k,v,d}(l)$ is defined as follows:

$$T_{k,v,d}(l) = \begin{cases} a & \text{if } k = 0 \\ sto(T_{k-1,v,d}(l), i_{l(k)}, e_0) & \text{if } k = 1 \text{ and } v = 0 \text{ and } d = 0 \\ sto(T_{k-1,v,d}(l), "i_{l(k)}", "e_0") & \text{if } k = 1 \text{ and } v = 0 \text{ and } d = 1 \\ sto(T_{k-1,v,d}(l), i_{l(k)}, e_{l(k)}) & \text{if } 0 \leq k < n \text{ and } v = 1 \text{ and } d = 0 \\ sto(T_{k-1,v,d}(l), "i_{l(k)}", "e_{l(k)}") & \text{if } 0 \leq k < n \text{ and } v = 1 \text{ and } d = 1 \end{cases}$$

We generated four different types of test cases: SC[I] without DOI, and SC[I] with DOI and considered arrays with 5, 15, $\dots$, 95 elements. For each of the 40 type-size-combinations, 20 random instances were generated and then flattened.

For an easier comparison of the results with and without DOI, they were put into relation by dividing the runtimes with DOI by the according runtimes without DOI. The

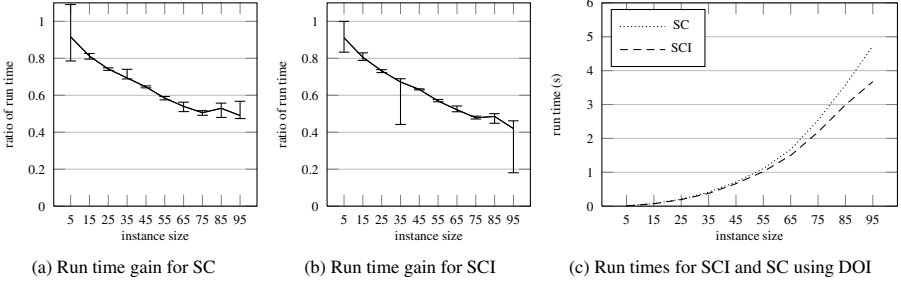


Fig. 1. Relative execution times for S-COMM and S-COMM-INV

Table 1. Comparison of benchmark results for E-KRHyper, HermiT and Pellet

Ontology Name	Nr. of Axioms	Runtimes (s)				
		Transformation	E-KRHyper (w/o DOI)	E-KRHyper (with DOI)	HermiT	Pellet
galen	5134	7.44	2.761	2.755	4.685	25.075
galen-ians-full-doctored	5668	8.70	2.987	2.992	4.887	28.102
galen-ians-full-undoctored	5911	8.27	3.152	3.148	7.596	29.441
not-galen	6825	13.61	3.570	3.565	11.057	28.665
emap	13730	19.09	4.270	4.259	10.894	64.533
vicodi_0	53876	11.79	2.699	2.675	10.522	106.831
vicodi_1	107529	16.26	6.941	6.893	12.632	233.324
vicodi_2	161182	18.50	13.246	13.221	15.074	387.200
vicodi_3	214835	20.05	19.092	19.201	17.719	512.019
vicodi_4	268488	24.95	24.411	24.590	18.819	ERROR

result of this operation can be seen in Fig. 1a [Fig. 1b] for SC [SCI], where the graph shows the average, minimal and maximal runtime, for each size. It is easy to see that the version with DOI outperforms the one without DOI and that the difference grows with the size of the arrays. We assume that the specific structure of the problems influences the runtime, which might lead to the visible fluctuations in the graphs. This has not yet been investigated. We compared the results of the DOI version of E-KRHyper with the results of the prover E, which is also able to handle DOI [34]. E solved the 200 problems for SC [SCI] with an average runtime of 0.09 [0.08] seconds where E-KRHyper needed 1.5 [1.27] seconds. Fig. 1c shows that for E-KRHyper with DOI the runtime in relation to the array size is quadratical. [15] provides details on the setup, benchmarking process and the evaluation.

4.2 Description Logic

For an evaluation E-KRHyper's performance on DL-ontologies we have chosen several ontologies from [22] and checked them for consistency. Systems for comparison are HermiT and Pellet. Tab. 1 shows the runtimes for those ontologies and provers.

The column *Transformation* lists the times needed to transform the ontologies into DL-clauses, i.e. a suitable input for E-KRHyper. Even with adding the transformation and inference times, E-KRHyper outperforms Pellet significantly. Adding the transformation and inference time, HermiT outperforms E-KRHyper in all of the presented

benchmarks, but the difference is not that big. Taking into account that HermiT is a highly optimised reasoner for OWL ontologies and E-KRHyper is a versatile prover for FOL with certain extensions, this result is impressive.

As it was our intention to compare E-KRHyper with DL-provers, the typical ontologies we used tend to feature only *at-least* literals with low cardinalities. An unfortunate result of this is that the experimental optimisation to use DOI with the *at-least* rule has hardly any effect, as seen in the table. A more thorough evaluation of this particular optimisation would require tailor-made ontologies and must be regarded as future work.

5 Conclusion

E-KRHyper is a versatile automated theorem prover and model generator for first-order logic with equality. In version 1.4 it remains above average in the TPTP-benchmark and can compete with highly specialised and optimised DL provers. With the introduced extensions it gains access to new application domains. For problems that rely on the inequality of certain constants, we have shown that using DOI has significant benefits over explicit inequality-facts. The possibility to reason in *SHIQ* with overall good performance increases E-KRHyper's field of use and makes it even more flexible.

Acknowledgements. We would like to thank Sonja Polster, who provided us with the results of the DL-benchmarks.

References

1. Apt, K.R., Blair, H.A., Walker, A.: Towards a theory of declarative knowledge. In: Minker, J. (ed.) Foundations of Deductive Databases and Logic Programming. pp. 89–148. Morgan Kaufmann Publishers Inc., San Francisco (1988)
2. Armando, A., Bonacina, M.P., Ranise, S., Schulz, S.: New results on rewrite-based satisfiability procedures. *ACM Trans. Comput. Log.* (2009)
3. Baaz, M., Fermüller, C.G.: Resolution-based theorem proving for manyvalued logics. *J. Symb. Comput.* (1995)
4. Bachmair, L., Ganzinger, H.: Equational reasoning in saturation-based theorem proving. In: Bibel, W., Schmitt, P.H. (eds.) Automated Deduction — A Basis for Applications. Kluwer (1998)
5. Baumgartner, P., Burchardt, A.: Logic programming infrastructure for inferences on frame-Net. In: Alferes, J.J., Leite, J. (eds.) JELIA 2004. LNCS (LNAI), vol. 3229, pp. 591–603. Springer, Heidelberg (2004)
6. Baumgartner, P., Furbach, U.: Living books, automated deduction and other strange things. In: Hutter, D., Stephan, W. (eds.) Mechanizing Mathematical Reasoning. LNCS (LNAI), vol. 2605, pp. 249–267. Springer, Heidelberg (2005)
7. Baumgartner, P., Furbach, U., Gross-Hardt, M., Kleemann, T.: Model based deduction for database schema reasoning. In: Biundo, S., Frühwirth, T., Palm, G. (eds.) KI 2004. LNCS (LNAI), vol. 3238, pp. 168–182. Springer, Heidelberg (2004)
8. Baumgartner, P., Furbach, U., Gross-Hardt, M., Kleemann, T., Wernhard, C.: KRHyper inside - model based deduction in applications. *Fachberichte informatik, Universität Koblenz-Landau* (2003)

9. Baumgartner, P., Furbach, U., Groß-Hardt, M., Sinner, A.: Living book - deduction, slicing, and interaction. *J. Autom. Reason* (2004)
10. Baumgartner, P., Furbach, U., Koblenz, U.: PROTEIN: A PROver with a Theory Extension INterface. In: CADE-12, Proc. Springer, Heidelberg (1994)
11. Baumgartner, P., Furbach, U., Niemelä, I.: Hyper tableaux. In: Orłowska, E., Alferes, J.J., Moniz Pereira, L. (eds.) JELIA 1996. LNCS, vol. 1126, pp. 1–17. Springer, Heidelberg (1996)
12. Baumgartner, P., Furbach, U., Pelzer, B.: Hyper tableaux with equality. In: Pfenning, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 492–507. Springer, Heidelberg (2007)
13. Baumgartner, P., Mediratta, A.: Improving stable models-based planning by bidirectional search. In: International Conference on Knowledge Based Computer Systems (KBSC) (December 2004)
14. Baumgartner, P., Suchanek, F.M.: Automated Reasoning Support for First-Order Ontologies. In: Alferes, J.J., Bailey, J., May, W., Schwertel, U. (eds.) PPSWR 2006. LNCS, vol. 4187, pp. 18–32. Springer, Heidelberg (2006)
15. Bender, M.: E-hyper tableaux with distinct object identifiers. Arbeitsberichte aus dem Fachbereich Informatik 01/2013, Universität Koblenz-Landau (2013), <http://www.uni-koblenz.de/FB4/Publications/Reports>
16. Faßbender, D.: DLE-hyper tableau calculus. Diplomarbeit, University of Koblenz-Landau (February 2012)
17. Furbach, U., Glöckner, I., Helbig, H., Pelzer, B.: Loganswer - a deduction-based question answering system (System description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 139–146. Springer, Heidelberg (2008)
18. Furbach, U., Glöckner, I., Helbig, H., Pelzer, B.: Logic-based question answering. In: KI (2010)
19. Furbach, U., Glöckner, I., Pelzer, B.: An application of automated reasoning in natural language question answering. In: *AI Commun.* (2010) (PAAR Special Issue)
20. Ganzinger, H., Sofronie-Stokkermans, V.: Chaining techniques for automated theorem proving in many-valued logics. In: Proc. ISMVL 2000. IEEE Computer Society (2000)
21. Hoder, K., Voronkov, A.: Sine qua non for large theory reasoning. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE 2011. LNCS, vol. 6803, pp. 299–314. Springer, Heidelberg (2011)
22. Horrocks, I.: Header Benchmarks (2009–2013), http://hermit-reasoner.com/2009/JAIR_benchmarks/ (accessed January 9, 2013)
23. Horrocks, I., Fensel, D., Broekstra, J., Decker, S., Erdmann, M., Goble, C., van Harmelen, F., Klein, M., Staab, S., Studer, R., Motta, E.: Oil: The ontology inference layer. Technical report, Vrije Universiteit Amsterdam, Faculty of Sciences (September 2000), <http://www.ontoknowledge.org/oil/>
24. INRIA. Objective Caml (OCaml) programming language website, <http://ocaml.org/>
25. McCune, W.: OTTER 3.3 Reference Manual. Argonne National Laboratory (2008)
26. Mossakowski, T., Maeder, C., Lüttich, K.: The heterogeneous tool set (hets). In: Beckert, B. (ed.) VERIFY. *CEUR Workshop Proceedings*, vol. 259. CEUR-WS.org (2007)
27. Motik, B., Shearer, R., Horrocks, I.: Optimized reasoning in description logics using hyper-tableaux. In: Pfenning, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 67–83. Springer, Heidelberg (2007)
28. Parsia, B., Sirin, E.: Pellet: An owl dl reasoner. In: 3rd International Semantic Web Conference (ISWC 2004) (2004)
29. Pelzer, B.: Project website of E-KRHyper (2007–2013), <http://userp.uni-koblenz.de/~bpelzer/ekrhyper/> (accessed January 21, 2013)

30. Pelzer, B., Glöckner, I.: Combining theorem proving with natural language processing. In: Proceedings of the First International Workshop on Practical Aspects of Automated Reasoning (PAAR-2008/ESHOL-2008), Sydney, Australia, August 10-11. CEUR Workshop Proceedings (2008)
31. Pelzer, B., Wernhard, C.: System description: E- kRHyper. In: Pfenning, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 508–513. Springer, Heidelberg (2007)
32. Riazanov, A., Voronkov, A.: The design and implementation of vampire. *AI Commun.* (2002)
33. Schulz, S.: E - a brainiac theorem prover. *AI Commun.* (2002)
34. Schulz, S., Bonacina, M.P.: On handling distinct objects in the superposition calculus. In: Konev, B., Schulz, S. (eds.) *Proc. IWIL* (2005)
35. Schwitter, R.: Anaphora resolution involving interactive knowledge acquisition. In: Fuchs, N.E. (ed.) *CNL 2009*. LNCS, vol. 5972, pp. 36–55. Springer, Heidelberg (2010)
36. Sinner, A., Kleemann, T.: KRHyper - In Your Pocket. In: Nieuwenhuis, R. (ed.) *CADE 2005*. LNCS (LNAI), vol. 3632, pp. 452–457. Springer, Heidelberg (2005)
37. Sutcliffe, G.: The TPTP website (2004-2013), <http://www.tptp.org> (accessed January 21, 2013)
38. Sutcliffe, G.: The TPTP problem library and associated infrastructure: The FOF and CNF parts, v3.5.0. *J. Autom. Reason.* (2009)
39. Wernhard, C.: System description: KRHyper. *Fachberichte informatik*, Universität Koblenz-Landau (2003)