Combining Theorem Proving with Natural Language Processing

Björn Pelzer[1] and Ingo Glöckner[2]

[1] Department of Computer Science, Artificial Intelligence Research Group
University of Koblenz-Landau, Universitätsstr. 1, 56070 Koblenz
bpelzer@uni-koblenz.de

[2] Intelligent Information and Communication Systems Group (IICS),
University of Hagen, 59084 Hagen, Germany
ingo.gloeckner@fernuni-hagen.de

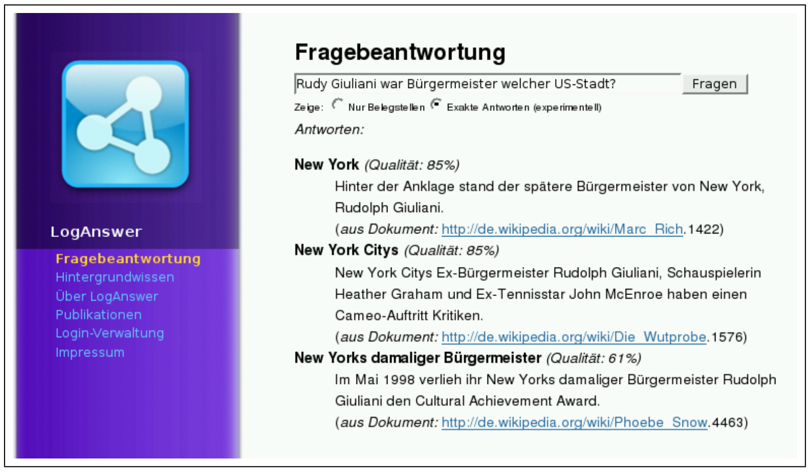
Abstract. The LogAnswer system is an application of automated reasoning to the field of open domain question answering, which aims at finding answers to natural language questions regarding arbitrary topics. In our system we have integrated an automated theorem prover in a framework of natural language processing tools to allow for deductive reasoning over an extensive knowledge base derived from textual sources. For this purpose we had to intertwine two opposing approaches: on the one hand formal logic with its precision but brittleness, and on the other hand, machine learning applied to shallow linguistic features, which are robust but less precise. In the paper we present implementation details and discuss obstacles and their proposed solutions.

1 Introduction

Question answering (QA) systems generate natural language (NL) answers in response to NL questions, using a large collection of textual documents.[3] Simple factual questions can be answered using only information retrieval and shallow linguistic methods like named entity recognition [1]. More advanced cases, like questions involving a temporal description, call for deduction based question answering which can provide support for temporal reasoning and other natural language related inferences [2]. Complete QA systems which integrate question answering and logical reasoning are [3,4,5]. The junctures of logic and answer validation are also addressed in research on recognizing textual entailment [6,7].

Our LogAnswer system uses first order logic to represent an extensive knowledge base, and a combination of NL processing tools and an automated theorem prover to derive answers within a few seconds, i.e. in a time frame appropriate for ad-hoc question answering on the web. The fields of automated reasoning and natural language processing (NLP) differ greatly in their methodologies. A

[3] A survey on the progress in question answering technology is provided by the QA track of the TREC conference series (see http://trec.nist.gov) and by the CLEF workshops (see http://www.clef-campaign.org/).



Fragebeantwortung

Rudy Giuliani war Bürgermeister welcher US-Stadt?

Zeige: Nur Belegstellen [✓ Exakte Antworten (experimentell)]

Antworten:

New York *(Qualität: 85%)*
Hinter der Anklage stand der spätere Bürgermeister von New York, Rudolph Giuliani.
(*aus Dokument:* http://de.wikipedia.org/wiki/Marc_Rich.1422)

New York Citys *(Qualität: 85%)*
New York Citys Ex-Bürgermeister Rudolph Giuliani, Schauspielerin Heather Graham und Ex-Tennisstar John McEnroe haben einen Cameo-Auftritt Kritiken.
(*aus Dokument:* http://de.wikipedia.org/wiki/Die_Wutprobe.1576)

New Yorks damaliger Bürgermeister *(Qualität: 61%)*
Im Mai 1998 verlieh ihr New Yorks damaliger Bürgermeister Rudolph Giuliani den Cultural Achievement Award.
(*aus Dokument:* http://de.wikipedia.org/wiki/Phoebe_Snow.4463)

Fig. 1. Screenshot of the LogAnswer prototype. The system is available online at `www.loganswer.de`.

theorem prover generally implements a sound and complete deduction calculus which can produce complex proofs using hundreds of inference steps, and it operates on a set of clauses or formulas where consistency or lack thereof is critical for the result. By contrast, natural language is ambiguous, textual sources may be imperfect, and as a consequence a knowledge base derived from such sources cannot be expected to be consistent. Moreover, it is not feasible to provide a complete formalization of the background knowledge used by persons in understanding natural language. A practical NLP approach must take this into account and employ robustness-enhancing methods to overcome the flaws and deliver useful results.

Merging the reasoning depth of an automated theorem prover with the robustness of NL processing presents a number of difficulties. We will provide a short overview of our LogAnswer system and then address the obstacles and solutions in detail.

2 Description of the LogAnswer System

LogAnswer is a QA-system supporting the German language. A full system description is presented in [8]. LogAnswer operates on a knowledge base consisting of general semantic background rules as well as factual knowledge derived from the German Wikipedia and a corpus of newspaper articles. This knowledge is represented by semantic networks in the MultiNet formalism [9]. The user interacts with LogAnswer via the user interface, where a NL question can be entered into a search box, as shown in Figure 1. The query is then processed in several stages. The WOCADI parser module [10] translates the query into a MultiNet representation. The formalized text passages from the MultiNet knowledge base are then searched for the query terms, and the matching network fragments are returned as the basis for generating answer candidates. Each retrieved fragment corresponds to a text passage which may contain an answer to the question. Currently, 200 MultiNet passage representations are retrieved for each question. Shallow linguistic methods provide additional filtering, further cutting down the number of candidate passages.

At this stage the deduction-based processing begins. The automated reasoning component of LogAnswer is E-KRHyper [11], an automated theorem prover for first order logic with equality, based on the hyper tableaux calculus [12,13]. E-KRHyper is an in-house system, developed for embedding in knowledge representation applications. It has been equipped with several features, commands and modes of operation for this purpose. Our familiarity with the system allows us to perform deep modifications when required. Because of our experience in adapting this prover to numerous systems in the past, it was a natural choice for a reasoner in LogAnswer. Its performance is roughly comparable to Otter [14], a system which generally serves as a benchmark among automated theorem provers.

The query and the MultiNet candidate passages are converted into first order logic representations. For each answer candidate the theorem prover E-KRHyper attempts to refute the negated query representation in conjunction with the background knowledge rules and the respective candidate. A successful refutation indicates that an answer has been found, and the queried information is extracted from the refutational proof.

If answers are found for multiple candidate passages, then they are ranked according to features extracted from logic processing and from shallow linguistic analysis (e.g. lexical overlap). The five best answers are presented to the user. Depending on user choice, the answers are given only in the form of snippets from the textual sources or as precise answers together with the snippets providing context. The quality score shown with each result is an estimate of the correctness probability of the answer determined by a machine learning approach.

3 Knowledge Representation

MultiNet (**Multi**layered Extended Semantic **Net**works) [9] is a formalism for knowledge representation via semantic networks, which is particularly suited for the meaning representation of natural language. Characteristic of MultiNet is its stable inventory of pre-defined relations (edge labels), which made possible the long-term development of a computational lexicon based on MultiNet [15], and the introduction of so-called layer attributes. These attributes are used in multi-dimensional node descriptions which serve to capture quantification and other aspects of meaning that cannot be expressed using the relational means of a semantic network.

The formalism is generally independent of any particular natural language, although the tools for translating NL into MultiNet are only available for German (and in a rudimentary form for English) at this time.[4] The knowledge base of LogAnswer was therefore derived from textual sources in German, namely the German Wikipedia and a corpus of newspaper articles. All in all about 12 million sentences have been translated into semantic networks, which are stored using Scheme syntax.

For the deduction-based processing in LogAnswer this knowledge base is further translated into first-order logic, stored in the TPTP format [16], a standard among automated theorem provers. The MultiNet nodes and attributed arcs can be translated in a fairly straightforward manner into relations and constants, and the same holds for the logical MultiNet rules which express the semantics of words and the logical properties of the MultiNet relations. However, it should be noted that MultiNet goes beyond the expressivity of pure first-order logic and TPTP. For example, MultiNet networks can contain generalized quantifiers like 'most' or 'almost all'. These aspects of the MultiNet representation are lost in the translation to logical expressions. In fact, our current translation from MultiNet to logic results in Horn logic (plus arithmetic expressions). It is planned to translate into a more powerful logic including equality in order to better capture the actual meaning of the natural language sentences. In order to allow such an extension, the E-KRHyper prover for full first-order logic with equality was chosen, which also supports arithmetic expressions.

4 Query Representation

For our purposes of logic-based question answering, the NL question must be translated into a conjunctive list of query literals. Synonyms are normalized by replacing all lexical concepts with canonical synset representatives. If the question asks for specific information, then this is represented by a special $FOCUS$ variable. In a successful proof this variable will be instantiated with the desired information from the knowledge base.

For example, *Rudy Giuliani war Bürgermeister welcher US-Stadt?*[5] translates into the following logical query:

$$\begin{aligned}&\mathrm{attr}(X_1, X_2), \mathrm{attr}(X_1, X_3), \mathrm{val}(X_2, \mathit{rudy.0}), \mathrm{sub}(X_2, \mathit{vorname.1.1}),\\ &\mathrm{val}(X_3, \mathit{giuliani.0}), \mathrm{sub}(X_3, \mathit{nachname.1.1}), \mathrm{sub}(FOCUS, \mathit{usstadt.1.1}),\\ &\mathrm{attch}(FOCUS, X_1), \mathrm{sub}(X_1, \mathit{bürgermeister.1.1})\end{aligned}$$

5 Deduction-based Query Processing

Each of the candidate passages retrieved from the knowledge base may contain an answer to the query, so a separate proof attempt is made for each candidate.

[4] In order to adapt the system to another language, a computational lexicon and a parser for generating MultiNet representations from expressions in that language must be provided. While the logical core of LogAnswer can remain the same, the lexical-semantic relations (synonyms, nominalizations etc.) used by LogAnswer must also be adapted to the language of interest.

[5] *Rudy Giuliani was the mayor of which city in the USA?*

The proof is done by refutation, with the logical query representation being treated as a negated conjecture. E-KRHyper operates on a clause normal form (CNF) representation and converts its first order input accordingly. To continue our example, E-KRHyper uses the following representation of the query:

$$\neg\,\mathrm{attr}(X_1, X_2) \vee \neg\,\mathrm{attr}(X_1, X_3) \vee \neg\,\mathrm{val}(X_2, \mathit{rudy.0}) \vee \neg\,\mathrm{sub}(X_2, \mathit{vorname.1.1}) \vee$$
$$\neg\,\mathrm{val}(X_3, \mathit{giuliani.0}) \vee \neg\,\mathrm{sub}(X_3, \mathit{nachname.1.1}) \vee \neg\,\mathrm{sub}(\mathit{FOCUS}, \mathit{usstadt.1.1}) \vee$$
$$\neg\,\mathrm{attch}(\mathit{FOCUS}, X_1) \vee \neg\,\mathrm{sub}(X_1, \mathit{bürgermeister.1.1})\,.$$

The prover keeps track of the $FOCUS$ variable throughout all clause transformations and inference steps, so that its binding can be extracted from a proof even if it has been renamed.

As its derivation E-KRHyper builds a hyper tableau in the form of a literal tree, using the hyper extension inference: if the negative literals of a clause unify with complementary literals from a tableau branch, then the positive literals of the clause are added as leaves. A branch is closed once it is found to contain a contradiction. In a derivation for LogAnswer this is the case when all the negative literals from the query unify with the branch; given that the query has no positive literals, the branch gets closed. The term bound to the $FOCUS$ variable in the unifying substitution used in the refutational proof then represents the queried information.

The current logical background knowledge consists of Horn formulas only, but with the ongoing translation of the MultiNet knowledge the logical rules will eventually contain non-Horn formulas as well. In a hyper tableaux derivation these can lead to tableau branching, with multiple closed branches and thus multiple closing unifiers. In such a case E-KRHyper extracts all the bindings for the $FOCUS$ variable from the proof and presents them as different answers to the main LogAnswer system.

6 Ensuring Robustness

A knowledge base derived from textual sources is bound to have imperfections. Furthermore, the logical background knowledge provided to the prover will never completely cover the actual background knowledge of a person who reads the text. Finally, the candidate passages are not guaranteed to contain an answer to the query, since they have only been selected by relatively simple filtering. For these reasons it is not certain that E-KRHyper can find a proof within an answer candidate, even if the corresponding NL source actually contains the queried information. Given that LogAnswer is supposed to provide answers in a short amount of time, the maximum time slot dedicated to a single proof attempt is constrained severely to ensure that all answer candidates can be tested. However, while a time limit ensures that the prover will not work indefinitely on one futile candidate when answers would be readily available in others, it does not help against missing an answer due to minor mismatches. When the formulas being reasoned upon have all been derived from imperfect textual sources and by imperfect tools for linguistic analysis, then formal logic may be too rigorous in demanding a perfect proof for an answer, and small compromises may actually be acceptable.[6]

For this reason E-KRHyper is embedded in a *relaxation loop*: if no proof is found within a time limit, then the query is relaxed by dropping a query literal and restarting the prover with the shortened query.[7] In theory this can be continued until a proof of the simplified query succeeds, but since LogAnswer aims to produce useful answers the loop will be stopped before all literals are skipped.

Also, rather than skipping a random query literal, the derivation progress during the failed refutation attempt can be used to guide the choice of which literal to drop. Given a negated query clause $\neg Q_1 \vee \cdots \vee \neg Q_n$, E-KRHyper tries to unify the query literals with fresh variants $B_1, \ldots, B_n$ of complementary branch literals from the current tableau branch b by testing the query literals from left to right. The unifying substitution is extended in every step such that starting with the empty substitution σ_0, σ_k is a substitution with $Q_i \sigma_k = B_i, \forall i \in \{1 \ldots k\}$, with $k \in \{1 \ldots n\}$. If one query literal $\neg Q_l$ fails to find a matching partner in the branch which would allow an extension of the current substitution σ_{l-1} to σ_l, then the remaining literals $\neg Q_{l+1}, \ldots, \neg Q_n$ are not tested under σ_{l-1}.

Instead E-KRHyper generates a *partial result.* A partial result is represented by a triple $(\{Q_1, \ldots, Q_{l-1}\}, \sigma_{l-1}, \{Q_l, \ldots, Q_n\})$, consisting of the list of successfully unified query literals, the unifying substitution, and the list of query literals that were not unified, the first of which being the one that failed, whereas the remaining were not tested at all. If E-KRHyper fails to find a proof within the time limit, then all partial results generated so far are returned to the main LogAnswer system. One of the best partial results is selected (i.e. one of the partial results with the highest number of refuted query literals), the failed literal $\neg Q_l$ is removed from the negated query clause and E-KRHyper restarts its derivation with the new query clause $\neg Q_1 \vee \cdots \vee \neg Q_{l-1} \vee \neg Q_{l+1} \vee \cdots \vee \neg Q_n$. When two partial results have the same number of failed literals, then additional criteria are used for selecting the best partial result: (a) partial results which provide a binding to the $FOCUS$ variable are considered better than partial results which do not bind the queried variable (this criterion is important since the system can only generate answers when the $FOCUS$ variable has been bound); and (b) a binding of the $FOCUS$ variable to a constant is preferable to a binding of the $FOCUS$ variable to a complex term (at the moment, the system is unable to generate answers for skolem terms, so answer extraction will only succeed when the queried variable is bound to a constant which directly represents a discourse entity).

[6] A training set is used to learn a useful interpretation of results for failed proofs [17].
[7] See Sect. 8 for an example. Another system which uses relaxation for achieving more robustness in logic-based QA is COGEX [3].

7 A Theorem Prover as a Reasoning Server

At the time of this writing LogAnswer is deployed as a web-based QA-system with an interface analogous to that of a typical search engine. This usage places certain demands on the performance of LogAnswer which must be able to respond to numerous queries from multiple users in a short time. The reasoning stage within E-KRHyper can easily be the most time-consuming phase in the processing of a query, even with relaxation. Cutting back on operations here is crucial for maintaining responsiveness. Fortunately there are several opportunities for such measures.

To summarize, the clause input for a single proof attempt consists of:

– **background knowledge:** 10,200 CNF clauses,
– **query:** the negated query clause, on average 8 literals before relaxation,
– **answer candidate:** the logic representation of the candidate passage, circa 230 unit clauses.[8]

E-KRHyper maintains this clause set with the help of several dicrimination-tree indexes [18]. The time required for the construction of these extensive tree data structures can exceed the allotment for the reasoning phase. Repeating this for each proof attempt would be prohibitive, in particular when considering that there may be m answer candidates to check for a single query, with n relaxation steps for each candidate, resulting in $m \times n$ proof attempts for each query.

However, the clause sets and their indexing trees differ only very slightly between any two reasoning tasks. Only the current query and answer candidate can change between two proof attempts. The background knowledge, which comprises approximately 97% of every clause set, remains the same. Therefore LogAnswer does not restart E-KRHyper for each reasoning task. Instead the prover is started once and provided with the background knowledge, so that the construction of the bulk of the indexing structures is only done once during this initialization. The task-specific clauses are then added and retracted again as needed. This is done using the mechanism of *layered* discrimination-trees: the task-specific clauses are not actually added to the same tree structures as the background knowledge. Instead additional trees (the *layers*) will be created to store the new clauses. Index reading operations access all layers, whereas writing operations only use the latest layer intended for new clauses. Once a proof attempt is completed, the index layers with the now unnecessary clauses are simply discarded. This avoids a lengthy extraction of these clauses from a single, shared tree.

There is even less difference between the clause sets used in two consecutive relaxation steps. The only change here is the removal of one query literal. This allows us to reuse even more clauses. Given an (interrupted) hyper tableaux derivation, all derived literals which were added to the initial branch before the

[8] The problem set used for determining these numbers consists of 1806 query/candidate passage combinations for the CLEF 2007 questions for German.

first branch split can be treated as lemmas. Since the current logic representation of the knowledge base is Horn only, the set of lemmas actually corresponds to all derived branch literals, although this is bound to change in the future. The lemmas are then kept for the next relaxation step. Thus a repetition of inferences is avoided, and even if a relaxation step is not successful in finding an answer, it still makes use of its limited time slot to add new lemmas. This way we combine relaxation with incremental reasoning [19].

8 Prover Result Evaluation

The input for E-KRHyper is selected and updated by the LogAnswer main system, and the operation of the prover is also directed and controlled as described above. Given that many supporting text passages will be analysed for a query, the results of E-KRHyper must be subjected to further evaluation as well in order to select those answers that are most suitable for the user.

For this we apply a machine learning approach which combines both logic-based and shallow syntactic features [17]: whenever E-KRHyper terminates with a refutation, then the respective text passage is regarded as containing an answer to the query. All such passages are ranked by an aggregated score, computed from logic-based criteria regarding the respective proofs, like the number of relaxation steps required, as well as from shallow syntactic features, like the relative proportion of lexical concepts and numerals in the question which find a match in the candidate passage. If the user has opted to receive answers in the form of text snippets, then the five best passages are selected for presentation.

On the other hand, if precise answers are desired, then further processing is necessary. The queried information is extracted directly from a refutational proof as the binding of the $FOCUS$ variable. The context of this instantiating answer value within the semantic network underlying the candidate passage is used to phrase the actual answer that can be presented to the user.

Continuing our running example we take a look at one of the candidate passages for which E-KRHyper will terminate: *Hinter der Anklage stand der spätere Bürgermeister von New York, Rudolph Giuliani.*[9] The logical representation of the passage is shown here (actually, only the fragment of the representation which models the relational structure):

$$\begin{aligned}
&\mathrm{hinter}(c221, c210) \wedge \mathrm{sub}(c220, \mathit{nachname.1.1}) \wedge \mathrm{val}(c220, \mathit{giuliani.0}) \wedge \\
&\mathrm{sub}(c219, \mathit{vorname.1.1}) \wedge \mathrm{val}(c219, \mathit{rudolph.0}) \wedge \mathrm{prop}(c218, \mathit{spät.1.1}) \wedge \mathrm{attr}(c218, c220) \wedge \\
&\mathrm{attr}(c218, c219) \wedge \mathrm{sub}(c218, \mathit{bürgermeister.1.1}) \wedge \mathrm{val}(c216, \mathit{new\_york.0}) \wedge \\
&\mathrm{sub}(c216, \mathit{name.1.1}) \wedge \mathrm{sub}(c215, \mathit{stadt.1.1}) \wedge \mathrm{attch}(c215, c218) \wedge \mathrm{attr}(c215, c216) \wedge \\
&\mathrm{subs}(c211, \mathit{stehen.1.1}) \wedge \mathrm{loc}(c211, c221) \wedge \mathrm{scar}(c211, c218) \wedge \mathrm{sub}(c210, \mathit{anklage.1.1})
\end{aligned}$$

A full proof of the query fails since the system is lacking knowledge that *Rudy* is a short form of *Rudolph*. Moreover, the fact that New York is a US city is not known to the system. Therefore two query literals cannot be proved, viz $\mathrm{val}(X_2, \mathit{rudy.0})$

[9] *Responsible for the charges was the future mayor of New York, Rudolph Giuliani.*

and sub(*FOCUS*, *usstadt.1.1*). After two relaxation steps, which skip these two literals, a proof is found with a constant *c215* bound to the *FOCUS* variable, which corresponds to an entity mentioned in the text. The information about the position of the entity in the text string is then used to extract the answer *New York*.

Some answers generated this way are discarded immediately. This includes trivial answers which return the term from the query (*Who is Virginia Kelley? - Virginia Kelley*) and non-informative answers (*the mother* instead of *the mother of Bill Clinton*). Of those answers passing these sanity checks the five best in the aforementioned ranking are selected and then displayed together with the supporting passages.

9 Conclusions

In this paper we have explored an application of automated reasoning to open domain question answering. With the ever growing amounts and availability of digitally stored information the utilization of this data is becoming an important but difficult task. Question answering systems strive to find specific information in a significantly more goal-directed manner than search engines. This requires deep reasoning over the semantics of stored data. Our approaches bridge the gaps between the precision yet brittleness of deduction and the robustness but limited accuracy of shallow linguistic techniques. This is achieved by combining the results of both levels using machine learning. We have shown how the difficulties of a theorem prover dealing with imperfect NL-derived data can be solved by embedding the deduction component in a robust knowledge processing framework, which uses as feedback loop to gradually relax the logical query.

The proposed approach to ensuring robustness has been evaluated in [17]. The task was that of identifying the correct answer passages in a set of 12,337 retrieved candidate passages. A baseline system using shallow features but no logical reasoning achieved an F-score of 42.7% in this experiment (other criteria for filtering quality were also studied). When combining shallow features and strict proofs, the F-score increased by 7.3%. Adding relaxation achieved another 7.9% improvement of the F-score. The best results were obtained in a configuration which allowed three relaxation steps. This experiment demonstrates that the performance of our QA system profits from the automated reasoning capabilities of the logic prover and from the proposed relaxation technique. Another benefit of the logic-based approach is that the answer bindings determined by the prover provide the basis for answer extraction.

In the future we intend to further improve the translation of the MultiNet formalism into first-order logic, enabling us to fully exploit the expressivity of the semantic networks in combination with automated reasoning.

References

1. Prager, J., Brown, E., Coden, A., Radev, D.: Question-answering by predictive annotation. In: SIGIR '00: Proceedings of the 23rd Annual International ACM

SIGIR Conference on Research and Development in Information Retrieval, New York, NY, ACM Press (2000) 184–191

2. Moldovan, D., Bowden, M., Tatu, M.: A temporally-enhanced PowerAnswer in TREC 2006. In: Proc. of TREC-2006, Gaithersburg, MD (2006)
3. Moldovan, D., Clark, C., Harabagiu, S., Maiorano, S.: COGEX: A logic prover for question answering. In: Proc. of NAACL-HLT 2003. Volume 1., Morristown, NJ (2003) 87–93
4. Saias, J., Quaresma, P.: The Senso question answering approach to Portuguese QA@CLEF-2007. In: Working Notes for the CLEF 2007 Workshop, Budapest, Hungary (2007)
5. Glöckner, I., Hartrumpf, S., Leveling, J.: Logical validation, answer merging and witness selection: A study in multi-stream question answering. In: Proc. of RIAO-07, Pittsburgh (2007)
6. Bos, J., Markert, K.: When logical inference helps determining textual entailment (and when it doesn't). In: Proc. of 2nd PASCAL RTE Challenge Workshop. (2006)
7. Bobrow, D., Condoravdi, C., Crouch, R., de Paiva, V., Kaplan, R., Karttunen, L., King, T., Zaenen, A.: A basic logic for textual inference. In: Proceedings of the AAAI Workshop on Inference for Textual Question Answering, Pittsburgh, PA (Jul 2005)
8. Furbach, U., Glöckner, I., Helbig, H., Pelzer, B.: LogAnswer - A Deduction-Based Question Answering System. In: IJCAR 2008 - 4th International Joint Conference on Automated Reasoning, Sydney, Australia, 10th - 15th August, 2008, Proceedings, to appear. Lecture Notes in Computer Science, Springer (2008)
9. Helbig, H.: Knowledge Representation and the Semantics of Natural Language. Springer (2006)
10. Hartrumpf, S.: Hybrid Disambiguation in Natural Language Analysis. Der Andere Verlag, Osnabrück, Germany (2003)
11. Pelzer, B., Wernhard, C.: System Description: E-KRHyper. In: Automated Deduction - CADE-21, Proceedings. (2007) 508–513
12. Baumgartner, P., Furbach, U., Niemelä, I.: Hyper Tableaux. In: JELIA'96, Proceedings. (1996) 1–17
13. Baumgartner, P., Furbach, U., Pelzer, B.: Hyper Tableaux with Equality. In: Automated Deduction - CADE-21, Proceedings. (2007)
14. McCune, W.: OTTER 3.3 Reference Manual. Argonne National Laboratory, Argonne, Illinois (2003)
15. Hartrumpf, S., Helbig, H., Osswald, R.: The semantically based computer lexicon HaGenLex. Traitement automatique des langues **44**(2) (2003) 81–105
16. Sutcliffe, G., Suttner, C.: The TPTP Problem Library: CNF Release v1.2.1. Journal of Automated Reasoning **21**(2) (1998) 177–203
17. Glöckner, I., Pelzer, B.: Exploring robustness enhancements for logic-based passage filtering. In: KES2008, Proceedings, to appear. Lecture Notes in Computer Science, Springer (2008)
18. McCune, W.: Experiments with Discrimination-Tree Indexing and Path Indexing for Term Retrieval. Journal of Automated Reasoning **9**(2) (1992) 147–167
19. Beckert, B., Pape, C.: Incremental theory reasoning methods for semantic tableaux. In: Proceedings, 5th International Workshop on Theorem Proving with Analytic Tableaux and Related Methods. Lecture Notes in Computer Science, Springer (1996)