An Application of Automated Reasoning in Natural Language Question Answering

Ulrich Furbach [a], Ingo Glöckner [b] and Björn Pelzer [a]

[a] *Artificial Intelligence Research Group University of Koblenz-Landau, Universitätsstr. 1, 56070 Koblenz, Germany*
E-mail: {uli|bpelzer}@uni-koblenz.de

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

The LogAnswer system is an application of automated reasoning to the field of open domain question answering. In order to find answers to natural language questions regarding arbitrary topics, the system integrates an automated theorem prover in a framework of natural language processing tools. The latter serve to construct an extensive knowledge base automatically from given textual sources, while the automated theorem prover makes it possible to derive answers by deductive reasoning. In the paper, we discuss the requirements to the prover that arise in this application, especially concerning efficiency and robustness. The proposed solution rests on incremental reasoning, relaxation of the query (if no proof of the full query is found), and other techniques. In order to improve the robustness of the approach to gaps of the background knowledge, the results of deductive processing are combined with shallow linguistic features by machine learning.[*]

Keywords: question answering, theorem prover

1. Introduction

Question answering (QA) systems generate natural language (NL) answers in response to NL questions, using a large collection of textual documents. Question answering offers several benefits over traditional information retrieval (IR) systems that make it potentially useful for a diverse range of practical applications, with considerable differences regarding the acceptable response time, the quality of the answers produced and the way they are presented. As a QA system both accepts input and produces output in natural language, there is no special training required for the user. This means that QA systems can be employed for casual users, for example in public information systems, or as research tools in libraries. Compared to traditional web search engines, the use of natural language for querying makes it easier to express the information need. For example, if the user is interested in the date of the Mumbai terrorist attacks, then a QA system can directly answer the question *'When did the Mumbai attacks take place?'* or more succinctly, *'date of the Mumbai attacks'*. It would be difficult to express the same search task using a traditional web search engine which cannot take the expected answer type of the question into account (in this case, the user is only interested in the date of the specified event).

Due to its support for more purposeful ways of querying, a QA system can often determine a concise result that directly answers the given question – an advantage over internet search engines that usually only deliver long lists of document references, entailing further study by the user. A web-based QA system thus provides an alternative to existing search engines which is potentially more time-saving and space-saving. The latter aspect might be relevant for mobile internet devices with small displays. The LogAnswer system was designed with this web-based perspective in mind, though it works with a local document collection.

There are different methods for determining answers automatically. Simple factoid questions can be answered using only information retrieval and shallow linguistic methods like named entity recognition [45]. More advanced cases, like questions involving a temporal description, call for deduction-based question answering which can provide sup-

[*]Funding of this work by the DFG (Deutsche Forschungsgemeinschaft) under contracts FU 263/12-1 and HE 2847/10-1 (LogAnswer) is gratefully acknowledged.

AI Communications
ISSN 0921-7126, IOS Press.

port for temporal reasoning and other natural language related inferences [36]. Syntactical data alone is an insufficient basis for such inferences to operate upon. Instead a logical QA system also incorporates a representation of the semantics. An early example demonstrating the use of logic for semantic NL analysis is DORIS [8]. This system is aimed at discourse regarding a limited domain and unsuitable for answering arbitrary questions.

The FALCON system [24] is an open domain QA system for English language. It does employ logic for answer derivation, but for its background knowledge it relies on the information found in the WordNet lexical database [14], which is not designed as a knowledge base and lacks the semantic depth necessary to support complex inferences.

The commercial PowerAnswer [37] is another English open domain QA system. It is equipped with a logical background knowledge based on Wikipedia, and it employs the COGEX theorem prover for answer derivation. This system also does not use a deep semantic knowledge base, instead its reasoning operates mostly directly on the actual textual sources. Its components and methods are therefore geared towards English and not language-independent.

MySentient Answers [12] is an example for a QA system featuring an extensive knowledge base of considerable semantic depth, as it incorporates a subset of the Cyc ontology [31] as the background knowledge. The commercial system is not intended for open domain QA. Rather, the customers must provide formally encoded world knowledge for their specific application domains. The authors describe a test system using world knowledge derived from a text corpus, but no performance data is provided.

Other complete QA systems which integrate question answering and logical reasoning are Senso [49] and IRSAW [21]. However, these solutions are research prototypes developed for the TREC or CLEF evaluation campaigns, which ignore the issue of processing time.

The theorem prover SNARK [51] is used for deductive question answering in a QA component employed by the Amphion and BioDeducta systems [55]. While adaptable to different domains, the system described by the authors is intended as a domain-specific research tool for scientists, and response times are a lesser concern. Questions cannot be entered freely. Instead the user composes a query in cooperation with a query elicitation mechanism, which ensures that queries remain within the domain.

The junctures of logic and answer validation are also addressed in research on recognizing textual entailment [9,7].[1]

The LogAnswer system, by contrast, is designed for open-domain QA with an extensive knowledge base generated from a collection containing millions of sentences. The deep linguistic analysis of the question, which results in a representation of the question in first-order logic, and the subsequent logical processing must be able to generate answers within a few seconds, i.e. in a time frame appropriate for ad-hoc question answering on the web. The current version of LogAnswer handles German questions only, but as the automated reasoning is performed on a logical knowledge representation, the core components of the system are independent of any particular natural language, making a future adaptation to English and other (similar) languages relatively straightforward.

Reaching an efficiency to match the required response times is not the only challenge for logic-based QA, though. Utilizing logic for QA is also difficult since the fields of automated reasoning and natural language processing (NLP) differ greatly in their methodologies. A 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 approach to logic-based question answering must thus employ robustness-enhancing techniques in order to deliver useful results even if one of these problematic cases occurs.

To summarize, our objective is twofold: firstly, the LogAnswer system demonstrates that automated reasoning can be very useful and sufficiently

[1] 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 of the CLEF workshops (see `http://www.clef-campaign.org/`).

efficient in a natural language oriented application suitable for real-world usage as a web-based question answering system. For this purpose we embed an automated theorem prover within a NLP framework. The system features a large-scale knowledge base which taken as a whole exceeds the capacities of theorem provers and thus cannot feasibly be the object of deductive reasoning. Secondly, we are reporting our experience that in such a complex system automated reasoning has to be combined with other AI techniques. We demonstrate that an approach based on machine learning can reduce the extensive amount of formally represented knowledge to logic problems which are manageable by an automated theorem prover, provided the latter is optimized for knowledge-related theorems. We identify these logic problems and show that our theorem prover of choice meets the requirements of our application. As a knowledge base semi-automatically derived from imperfect textual sources is unlikely to meet the exacting demands of automated reasoning, we present methods which overcome this brittleness and make our system robust enough for its intended usage model. This demonstrates that the embedding of the prover in an AI system like LogAnswer is only possible by using subtle properties of the prover together with a tailored interface.

We start our paper by shortly commenting knowledge representation issues in the next section, followed by a high-level description of the LogAnswer system in Section 3. Our embedded theorem prover is presented in Section 4. We then detail several aspects of relevance to the use of logic for QA: passage retrieval (Section 5), which fetches the logical representations of the candidate passages; deduction based processing of the query (Section 6); and relaxation of the query based on partial prover results (Section 7) as a means of increasing robustness. After discussing the efficient use of an automated theorem prover as a reasoning server (Section 8), we consider the problem of selecting the best answers based on partial prover results (Section 9) and present a machine-learning approach suited for this reranking task (Section 10). In Section 11, we evaluate several aspects of LogAnswer. Limitations of our system are discussed in Section 12. The final section (Conclusions) summarizes the main results.

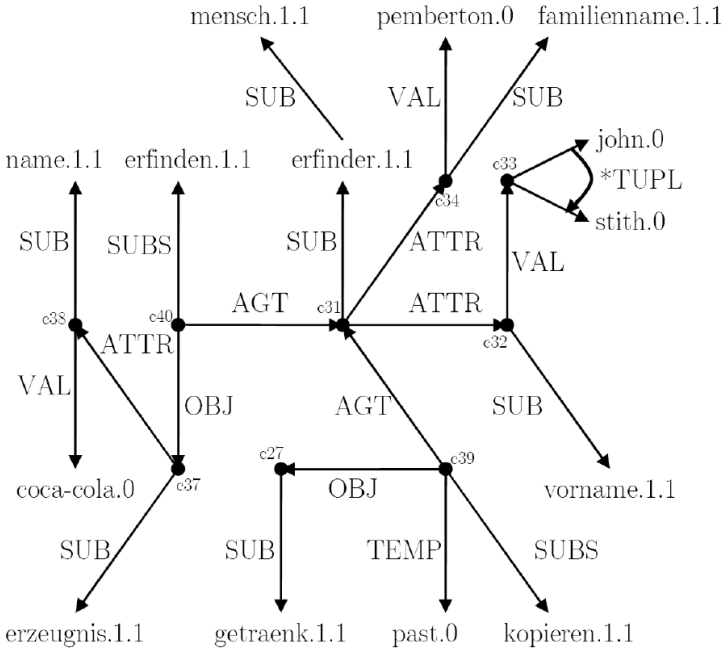
2. Knowledge Representation

Knowledge representation in natural language understanding is a traditional topic of AI research. Over the decades various formalisms of varying expressiveness have been introduced and used. In [28] the knowledge representation framework MultiNet (**Multi**layered Extended Semantic **Net**-works) is introduced as an extension of semantic networks specifically 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 [27], and the introduction of so-called layer attributes which serve to capture quantification and other aspects of meaning that cannot be expressed using the relational means of a semantic network. MultiNet comes with a set of axioms which allow inferences on the networks.

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.[2] 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 MultiNet semantic networks, which are stored using Scheme syntax. The MultiNet formalism and its applications have been the focus of our research for many years, and the knowledge base together with tools for acquisition and maintenance have been put to use in various projects.

For the deduction-based processing in LogAnswer this knowledge base is further translated into first-order logic, stored in the TPTP format [52], a standard among automated theorem provers. The MultiNet nodes and attributed arcs can be translated in a fairly straightforward manner into constants and predicates respectively, and the same holds for the logical MultiNet rules which express the semantics of words and the logical

[2] 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.



(a) MultiNet

$$
\begin{aligned}
&\text{sub}(\textit{erfinder.1.1}, \textit{mensch.1.1}) \\
&\wedge\ \text{subs}(\textit{c40}, \textit{erfinden.1.1}) \\
&\wedge\ \text{obj}(\textit{c40}, \textit{c37}) \\
&\wedge\ \text{agt}(\textit{c40}, \textit{c31}) \\
&\wedge\ \text{temp}(\textit{c39}, \textit{past.0}) \\
&\wedge\ \text{subs}(\textit{c39}, \textit{kopieren.1.1}) \\
&\wedge\ \text{agt}(\textit{c39}, \textit{c31}) \\
&\wedge\ \text{obj}(\textit{c39}, \textit{c27}) \\
&\wedge\ \text{sub}(\textit{c38}, \textit{name.1.1}) \\
&\wedge\ \text{val}(\textit{c38}, \textit{coca-cola.0}) \\
&\wedge\ \text{sub}(\textit{c37}, \textit{erzeugnis.1.1}) \\
&\wedge\ \text{attr}(\textit{c37}, \textit{c38}) \\
&\wedge\ \text{val}(\textit{c34}, \textit{pemberton.0}) \\
&\wedge\ \text{sub}(\textit{c34}, \textit{familienname.1.1}) \\
&\wedge\ *\text{tupl}(\textit{c33}, \textit{john.0}, \textit{stith.0}) \\
&\wedge\ \text{sub}(\textit{c32}, \textit{vorname.1.1}) \\
&\wedge\ \text{val}(\textit{c32}, \textit{c33}) \\
&\wedge\ \text{sub}(\textit{c31}, \textit{erfinder.1.1}) \\
&\wedge\ \text{attr}(\textit{c31}, \textit{c34}) \\
&\wedge\ \text{attr}(\textit{c31}, \textit{c32}) \\
&\wedge\ \text{sub}(\textit{c27}, \textit{getraenk.1.1})
\end{aligned}
$$

(b) First-order logic

Fig. 1. Example for the translation from MultiNet to first-order logic, for the NL-sentence '*Das Getränk wurde von John Stith Pemberton, dem Erfinder von Coca-Cola, kopiert.*' ('*The beverage was copied by John Stith Pemberton, the inventor of Coca-Cola.*') The figure shows only the fragment of the representation which models the relational structure. The integer suffixes of the word-based constants are used for disambiguation, they identify the particular word sense represented by the word within the current context. A subnet of the network may illustrate the MultiNet approach: the node labeled *c31* represents the person John Stith Pemberton. As such it is subordinate to the concept *erfinder* (*inventor*), expressed by a SUB arc to the node *erfinder.1.1*, which in turn is subordinate to *mensch* (*human being*). *c31* also has attributes represented by the nodes *c32* and *c34* (connected by ATTR arcs). The latter is subordinate to *familienname* (*surname*) and has the value *pemberton* - it represents the surname *Pemberton*. *c32* is the analogous for the given names *John* and *Stith*. The special *TUPL arc states that the two given names form the value of node *c32* in that particular order; i.e. it ensures that the person's name is *John Stith Pemberton*, not *Stith John Pemberton*.

$$\forall H \forall P \forall T : ((\text{hsit}(H, P) \wedge \text{temp}(H, T)) \rightarrow \text{temp}(P, T))$$

Fig. 2. Example for an axiom of the logical background knowledge. The axiom expresses the transitivity of the temporal relation: if the situation H takes place within the temporal frame T (temp(H, T)), and if H is a *hypersituation* with respect to situation P (hsit(H, P), i.e. P is a part of H), then P also takes place within T (temp(P, T)). For example, if a wedding (H) took place last week (T), then the exchange of the rings (P), which is a situational part of H, also took place last week.

properties of the MultiNet relations. See Figure 1 for an example.

It should be noted that some of the representational means of MultiNet go far beyond the expressivity of pure first-order logic and hence also beyond Description Logic. For example, MultiNet networks can contain generalized quantifiers like 'most' or 'almost all'.[3] These aspects of the Multi-

[3] A proof that 'most' in the sense of 'more than half' is not first-order definable can be found in [2], Theorem C12.

Net representation are lost in the translation to logical expressions. In fact, the clause normal form (CNF) of our first-order logic representation consists of Horn clauses (plus arithmetic expressions), as the more complex disjunctive inference rules of the MultiNet formalism have not been adapted yet. 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[4] [41] prover for full first-order logic with equality was chosen, which also supports arithmetic expressions.

3. Overview of the LogAnswer System

LogAnswer is a QA system supporting the German language. For a detailed overview we refer to the system description [16], as well as to elaborations on a number of different aspects of LogAnswer [22,40]. The system originates from the logic-based answer validator MAVE [18] (developed for the multi-stream QA system IRSAW [21,32]). The validator was re-designed for real-time question answering [19] and extended to a full QA system by adding a logic-based answer extractor, a user interface and other components. LogAnswer operates on a knowledge base consisting of general background knowledge (typically concerned with the entailments licensed by lexical-semantic relationships between word senses, but also containing a large number of reasoning axioms based on MultiNet inference rules - see Figure 2 for an example of a background knowledge axiom), as well as factual knowledge derived from the German Wikipedia and a corpus of newspaper articles. As described in the previous section, this knowledge is represented by semantic networks in the MultiNet formalism.

The user interacts with LogAnswer via the web-based user interface, where a NL question can be entered into a search box, as shown in Figure 3. The query is then processed in several stages. The WOCADI parser module [25] translates the query into a MultiNet representation. This process also includes a word sense disambiguation based on selectional restrictions expressed in the computational lexicon HaGenLex [27]. Based on this analysis, the category of the question (*factoid* vs. *definition*) is identified, as well as the *expected answer type* (e.g. PERSON). 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. A total of 200 MultiNet passage representations are retrieved for each question. The ranking of the retrieved passages is improved further by a machine learning approach trained on shallow linguistic features.

At this stage the deduction-based processing begins. The query and the MultiNet candidate passages are converted into their first-order logic representations. For each answer candidate the theorem prover E-KRHyper then attempts to prove the query representation in conjunction with the background knowledge rules and the respective candidate. A successful proof indicates that an answer has been found, and the queried information is extracted from the proof.

With the use of candidate passages we avoid having our deduction components work on the entire knowledge available to LogAnswer at once, as this would be prohibitive for the performance of the system. By processing the compact candidates one by one, the deduction can focus on the most promising fragments of the knowledge.

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), again utilizing machine learning methods. The five best answers are presented to the user. Depending on the available information (in particular on the success of extracting an answer phrase), each answer is either given only in the form of a snippet from the textual source, or as a precise answer highlighted in the snippet that provides a context. The quality score of each result is displayed graphically as a horizontal bar. This score is an estimate of the correctness probability of the answer determined by the machine learning approach.

Van Benthem [6], Theorem 6.2, presents a general characterization of those quantifiers that are first-order definable.

[4] **K**nowledge **R**epresentation **Hyper** Tableau with **E**quality; the prover is freely available under the GPL from `http://www.uni-koblenz.de/~bpelzer/ekrhyper`

Fig. 3. Screenshot of the LogAnswer prototype. A conserved version of the system that reproduces the results shown in this paper is available online at `http://www.loganswer.de/rev-0.8/`.

4. The Theorem Prover in LogAnswer

As described, the LogAnswer QA system is deployed on the web, with an interface analogous to that of a typical search engine. This usage places certain demands on the performance of LogAnswer. The system must be able to respond to numerous queries from multiple users within short amounts of time. Answers are to be delivered with at most a few seconds delay, and ideally instantly. Automated theorem provers are generally not designed with this usage model in mind. Instead they usually operate on a single-problem basis: the prover is started with all formulas or clauses relevant for the reasoning task at hand, begins its derivation, and after a while it terminates, either with a successful result or failure due to time or hardware limitations.

While the reasoning speed has a high priority in the development of automated theorem provers, the time constraints commonly imposed on systems are still relatively generous compared to the time slots available for the reasoning in LogAnswer. For example, in 2008 the time limit in the annual CADE ATP System Competition (CASC) [39,53] for automated theorem provers was set at 240 seconds for each problem. The complexity of proof derivations for typical theorem prover problems like those forming the majority of the TPTP collection usually arises less from the size of the input set of clauses, rather it is a result of the proof depth which requires a large number of consecutive inference steps, combined with the collateral derivation of clauses which are ultimately unnecessary for the proof itself. Theorem provers build extensive indexing structures to store the clauses of a problem and to facilitate the fast searching required for inference computation and for redundancy checks. Furthermore a preprocessing stage is possible before the actual deduction process begins. Here simple optimizations can be performed like the elimination of tautologies, duplicate literals and variant clauses. The input clauses can be normalized, sorted and transformed to better fit the particular calculus employed by the prover, axioms can be added, and strategies of the prover can be adjusted to specific features of the current problem.

In LogAnswer the input set of clauses is considerably larger than in most TPTP problems.[5] The

[5] Out of the 6346 clause normal form problems in the TPTP v3.5.0 (the latest stable release at the time of this writing), 65% have less than 100 clauses and 95% have less than 1000. The average number of clauses per problem is 180, and 82% of the problems are below this average.

clause input for a single LogAnswer 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.

The problem set used for determining these numbers consists of 1,805 query/candidate passage combinations for the QA@CLEF 2007 questions for German [17].

To summarize, a theorem prover for LogAnswer must be able to quickly solve problems characterized by a large number of clauses, of which only a few are necessary for any given proof.

The system we have chosen for LogAnswer is E-KRHyper, an automated theorem prover for first-order logic with equality, based on the hyper tableaux calculus [3,4]. E-KRHyper is an in-house system, developed as a general theorem prover and model generator for first-order logic, with special emphasis on embedding in knowledge representation applications. It can handle problems in TPTP syntax (both as first-order formulas or in clause normal form), and in the Prolog-like Protein format. According to the ATP performance listings on the TPTP-website[6], E-KRHyper solves 24% of the entire TPTP. This is slightly above the 21% rate of the Otter [35] theorem prover which serves as a benchmark among ATP systems due to its long history and stability, but the performance of E-KRHyper on the common logic problems which form the bulk of the TPTP is moderate compared to leading systems. Nevertheless E-KRHyper is ranked as a SOTA (state-of-the-art) contributor, which means that it can solve some TPTP problems no other prover solves. An automated theorem prover usually performs better on some classes of problems than in others, depending on the calculus it implements as well as on reasoning strategy and implementation details. E-KRHyper has shown to be very efficient in solving the logic problems occurring in the operation of LogAnswer, outmatching otherwise leading theorem provers. A detailed performance evaluation of E-KRHyper in comparison to other ATP systems is given in Section 11.3.

As the expressivity of MultiNet exceeds that of pure first-order logic, a theorem prover for LogAnswer should also feature extensions to first-order logic so that our logical representations of the NL-text derived MultiNet knowledge base can capture as much of the knowledge as possible. Since one of our main research interests is the combination of automated reasoning and natural language processing, the prover must allow a tight integration of its reasoning within the LogAnswer framework in order to support the methods we have developed. E-KRHyper was built to interact with knowledge representation applications, and accordingly the prover is equipped with a multitude of extensions to first-order logic. This includes equality handling and negation as failure. Arithmetic evaluation of float and integer numbers is also supported. Certain features have been adapted from the Prolog programming language, among them the list structure and special predicates like `findall/3` and `sort/2`. These extensions enable E-KRHyper to work with the different kinds of data occurring in the knowledge representation of LogAnswer, and they can replicate or at least approximate some aspects of MultiNet which go beyond first-order logic.

On the operational level E-KRHyper allows a high degree of control over its inference process. In the prover's interactive mode a user or another application can communicate with E-KRHyper, retrieve information about its status and enter commands. Proving can be halted and resumed, logic data can be entered and retracted, proofs are provided both for refutations and for models.

While such features are useful for the integration of E-KRHyper in LogAnswer, taken by themselves they are insufficient to support an implementation of our methods for combining automated reasoning and natural language processing. The operations described in Sections 7 and 8 require access to information and means of control which are not necessary in the common usage of an automated theorem prover, and hence they are not supported by most systems. As E-KRHyper is our own development and we are experienced with its implementation, making the required modifications is much simpler for us here than with any other ATP.

In summary, the performance of E-KRHyper gives us confidence that our prover is a highly appropriate choice among ATP systems for the logic problems encountered in LogAnswer. Together with its logic extensions and the ease of modifications, we consider E-KRHyper to be a suitable reasoner for LogAnswer.

[6] `http://www.tptp.org`

5. Passage Retrieval

The retrieval stage of LogAnswer rests on a bag-of-words information retrieval (IR) approach. Two alternative retrieval modules can be used for the selection of text passages from the knowledge base, either the retrieval module of IRSAW [32], or a recent Lucene-based solution [23]. For the moment, we assume a segmentation into sentence-sized passages. However, the Lucene-based IR system enriches the descriptors of the considered sentence by descriptors from other sentences that result from a coreference solution of pronouns. For example, if an occurrence of the pronoun 'er' (he) in the considered sentence refers to a mention of 'John Pemberton' in a previous sentence, then the person name is also added to the index. In order to improve the ranking of the retrieved passages, a decision-tree based machine-learning approach is applied. It uses shallow linguistic features that can be computed from the known analysis of the passages and from the question classification. The following features are used for the 'shallow' passage reranking step (see [19,23] for more details):

- *failedMatch*: The number of lexical concepts and numerals in the question which cannot be matched with the candidate passage.
- *matchRatio*: Relative proportion of lexical concepts and numerals in the question which find a match in the candidate passage.
- *failedNames*: Proper names mentioned in the question, but not in the passage.
- *containsBrackets*: Indicates that the passage contains a pair of parentheses.[7]
- *eatFound*: An occurrence of the expected answer type has been found in the passage.
- *defLevel*: This feature is useful for definition questions. A value of $defLevel = 2$ indicates that the snippet contains a defining verb or apposition, and $defLevel = 1$ indicates a relative clause.
- *irScore*: The original passage score determined by the retrieval component.

Improving the quality of the ranking is important since the logical processing considers the retrieved passages in decreasing order of relevance until a timeout is reached. Therefore a candidate with a low rank might not be subjected to logical processing at all.

The machine learning method used for the shallow-feature based reranking is also applied later on in a second reranking step, where additional features from the deduction process can be used (see Section 9). Section 10 details the machine learning method itself.

6. Logical Representation and Deduction-based Query Processing

The theorem prover E-KRHyper operates on first-order logic, and all MultiNet data involved in a proof attempt must be translated accordingly. Each of the candidate passages retrieved from the knowledge base may contain an answer to the query, so the prover makes a separate proof attempt for each candidate. The input of any single prover run thus consists of the logical representations of the MultiNet background knowledge, one of the 200 answer candidates and the query. Only the query is translated on the fly; the background knowledge and the MultiNet text passage representations underlying the answer candidates have been translated in advance and are stored in first-order form.

The logical translation of a query consists of a conjunctive list of query literals. These literals represent a MultiNet fragment where variables take the place of node identifiers. To answer a query, this fragment must be matched against the MultiNet knowledge base, thereby instantiating the variables with concrete nodes. A proof attempt by the prover can then be regarded as the logical representation of this matching process: the proof is done by refutation, the query is treated as a negated conjecture, and a successful refutation means that the variables of the logical query have been instantiated by constants representing nodes in the MultiNet knowledge base. If the question asks for specific information, then this is represented by a special *FOCUS* variable. For example, *'Wer erfand Coca-Cola?*[8] translates into the logical query in Figure 4, with the FOCUS variable representing the person of interest.

In a successful proof, the *FOCUS* variable is instantiated to the MultiNet node which represents

[7] The queried information is often found in parentheses, for example *Mount Everest (8,848 m)*.

[8] *'Who invented Coca-Cola?'*

$$\text{obj}(X_1, X_2) \wedge \text{subs}(X_1, \textit{erfinden.1.1}) \wedge \text{agt}(X_1, FOCUS) \wedge \text{attr}(X_2, X_3) \wedge \text{sub}(X_3, \textit{name.1.1}) \wedge \text{val}(X_3, \textit{coca\_cola.0})$$

Fig. 4. Logical query representation

the requested information, the so-called *answer kernel*. The answer generation module will then translate this node into a natural language answer (at the moment, based on a direct extraction of the answer string from the passage).

E-KRHyper operates on a CNF representation and converts any first-order input given in formula form accordingly. To continue our example, for its refutational proof attempt E-KRHyper uses the negated query representation given in Figure 5.

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 during the process.

An exception to the creation of purely conjunctive logical query representations occurs during the handling of disjunctive questions. The logical representation of a disjunctive question will contain disjunctive literals. An example for this is the following modification of the previous query: *'Wer erfand Coca-Cola oder Pepsi-Cola?*[9] This results in the logical query shown in Figure 6. After negation of this conjecture and the subsequent transformation into CNF, the logical query representation for E-KRHyper consists of multiple clauses, see Figure 7. Such clauses are handled simultaneously in a single proof attempt, and disproving one of them is sufficient for an answer.

As its derivation E-KRHyper builds a hyper tableau in the form of a literal tree, using the hyper extension inference [3]: 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.

7. Relaxation and Partial Results

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. In addition, 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 any selected answer candidate, even if the text passages actually contain 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 which comprise the basis for the reasoning 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.[10]

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

[9] *'Who invented Coca-Cola or Pepsi-Cola?'*

[10] A training set is used to learn a useful interpretation of results for failed proofs; see section 9.

$$\neg\,\mathrm{obj}(X_1, X_2) \vee \neg\,\mathrm{subs}(X_1, \textit{erfinden.1.1}) \vee \neg\,\mathrm{agt}(X_1, FOCUS)$$
$$\vee \neg\,\mathrm{attr}(X_2, X_3) \vee \neg\,\mathrm{sub}(X_3, \textit{name.1.1}) \vee \neg\,\mathrm{val}(X_3, \textit{coca\_cola.0})\,.$$

Fig. 5. Negated logical query representation

$$\mathrm{obj}(X_1, X_2) \wedge \mathrm{subs}(X_1, \textit{erfinden.1.1}) \wedge \mathrm{agt}(X_1, FOCUS)$$
$$\wedge\,\mathrm{sub}(X_3, \textit{name.1.1}) \wedge \mathrm{attr}(X_2, X_3) \wedge (\mathrm{val}(X_3, \textit{coca\_cola.0}) \vee \mathrm{val}(X_3, \textit{pepsi\_cola.0}))$$

Fig. 6. Logical query representation of a disjunctive question

$$\neg\,\mathrm{obj}(X_1, X_2) \vee \neg\,\mathrm{subs}(X_1, \textit{erfinden.1.1}) \vee \neg\,\mathrm{agt}(X_1, FOCUS)$$
$$\vee \neg\,\mathrm{sub}(X_3, \textit{name.1.1}) \vee \neg\,\mathrm{attr}(X_2, X_3) \vee \neg\,\mathrm{val}(X_3, \textit{coca\_cola.0})\,.$$
$$\neg\,\mathrm{obj}(X_1, X_2) \vee \neg\,\mathrm{subs}(X_1, \textit{erfinden.1.1}) \vee \neg\,\mathrm{agt}(X_1, FOCUS)$$
$$\vee \neg\,\mathrm{sub}(X_3, \textit{name.1.1}) \vee \neg\,\mathrm{attr}(X_2, X_3) \vee \neg\,\mathrm{val}(X_3, \textit{pepsi\_cola.0})\,.$$

Fig. 7. Negated logical query representation of a disjunctive question

shortened query.[11] 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 that provide a binding to the *FOCUS* variable are considered better than partial results that 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).

[11] See Sect. 9 for an example. Another system which uses relaxation for achieving more robustness in logic-based QA is COGEX [37].

8. A Theorem Prover as a Reasoning Server

We have implemented a number of modifications to our theorem prover E-KRHyper in order to integrate it into LogAnswer; in particular to improve the combination of automated reasoning and natural language processing. More specifically we aim to preserve as much of the reasoning depth of the ATP as possible while also keeping its processing time down far enough that a large number of proof attempts can be carried out in a negligible time frame, thereby maintaining response times which do not bother a human user.

Indexing E-KRHyper maintains the sizable input clause set with the help of several discrimination-tree indexes [34]. Initializing a theorem prover with an input of the size typical for LogAnswer in the conventional manner for each single proof attempt, including the construction of the clause storage indexes and possibly a preprocessing stage before the calculus application, can easily exceed the time allotment for the reasoning phase before any actual reasoning has begun. 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. Cutting back on operations here is crucial for maintaining responsiveness. Fortunately there are several opportunities for such measures.

There are very few differences between the clause sets and indexing trees of 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, treating them as a single combined index, whereas writing operations only use the latest layer intended for new clauses. Once a proof attempt is completed, the newest index layers with the now unnecessary clauses derived in the previous reasoning task are simply discarded. This avoids a lengthy extraction of these clauses from a single, shared tree.

Reusing the background knowledge once loaded saves approximately 0.9 seconds per attempted proof on our test system (see Section 11.3 for specifications). While this amount may appear negligible, the savings quickly add up when considering that the processing of a single question may involve hundreds of proof attempts.

Incremental Reasoning 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 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. In this way, we combine relaxation with incremental reasoning [5].

Knowledge Base Partitioning Given the small time-slots for deduction, the possibilities for preprocessing of the input clauses are limited. While all proof tasks share the majority of clauses, the remaining differences would still require optimizations to be specific for any single derivation. Tests for subsumption and tautologies are performed on both the shared background knowledge and the individual query-related clauses. However, more complex preprocessing steps have to compromise in order to minimize the delay. A straightforward optimization is to partition the knowledge base so that each derivation is started with those clauses only which might actually help in reaching the goal.

The derivation within E-KRHyper proceeds in a bottom-up manner: following the inference rules

of the hyper tableaux calculus, the background knowledge clauses are applied repeatedly to any unit clauses, deriving new units in the process. Such a bottom-up strategy avoids solving the same subgoals multiple times, but a disadvantage is the lack of goal-direction. This becomes particularly apparent given the fact that the goal in a proof task for LogAnswer is clearly defined and restricted. Unlike in the common usage of automated theorem provers where any refutation of an input clause set is considered a success, for LogAnswer only those refutations are useful which involve the query clause and which include an instantiation of the *FOCUS* variable. Out of the large number of inferences possible on the extensive knowledge base, only a small subset is actually likely to be relevant for reaching a useful refutation. Naturally, an accurate identification of such inferences and the clauses involved is only possible after a refutation has been found. However, a query based filtering of the knowledge base is possible before the proof attempt, and this can remove such clauses from the derivation which are certain to be irrelevant.

For a refutation of a (negated) query clause, each of its negative literals must be refuted. A knowledge base clause is certain to be useless for such a refutation if it cannot lead to the derivation of a positive literal with the same predicate symbol as a query literal. We compute a dependency graph of the clauses where clause C depends directly on D if a negative literal in C shares the predicate symbol with a positive literal in D. Given a query clause, it is then trivial to determine all clauses the query depends on directly or indirectly, and the remaining clauses can be ignored during the deductive processing.

The MultiNet formalism provides a fixed vocabulary of predicate symbols, and E-KRHyper computes the subsets of potentially relevant clauses for each predicate once after the initial reading of the knowledge bases.

For each subsequent reasoning task, the computed dependency information is used in two ways to prune the clause set: (a) E-KRHyper first deactivates all clauses which are not in one of the subsets of clauses potentially relevant for the predicates found in the current query clause. (b) Then all those clauses are deactivated which depend on the existence of unit clauses with predicates for which no active units exist. For example, given a set of clauses:

$$p(FOCUS) \wedge q(X) \rightarrow . \quad (1)$$

$$r(X) \rightarrow p(X). \quad (2)$$

$$p(X) \wedge u(Y) \rightarrow r(Y). \quad (3)$$

$$s(X) \rightarrow q(X). \quad (4)$$

$$p(X) \rightarrow t(X). \quad (5)$$

$$\rightarrow p(a). \quad (6)$$

$$\rightarrow s(b). \quad (7)$$

In step (a) E-KRHyper would deactivate clause (5), as it is not relevant for the conjecture (1). Then in step (b) the clauses (2) and (3) would be deactivated, as both depend on unit clauses with the predicates $r/1$ or $u/1$, but no units exist for either predicate. Activating and deactivating requires a negligible amount of time, and the clause indexing structures do not have to be changed. The deactivated clauses remain in the indexes, but they are ignored during the reasoning.

On average this allows the removal of 20% of the inference driving non-unit clauses, resulting in an average reduction of the proof time by 10%. As such the savings are relatively minor, but since the computational cost after the initial dependency computation is low, there is hardly any trade-off. Also, the current background knowledge represents the core of MultiNet, so it is to be expected that few of these essential reasoning rules are omissible. As the background knowledge base grows, the proportion of clauses irrelevant to a query will increase.

There are similarities between our partitioning and the graph-based hypothesis selection of Couchot and Hubert [11], in particular their notion of a *static dependency graph*. This graph represents dependencies between predicates, whereas our method uses the dependencies between clauses. However, both kinds of dependencies are closely related, and essentially they imply each other. The static dependency graph is more complex in that its edges are labeled by weights based on the sizes of the clauses involved in its computation. These weights are then used to sort the predicates which label the nodes, effectively ordering the predicates by their likelihood to be relevant. This allows tuning the hypothesis selection by only selecting clauses containing predicates deemed sufficiently relevant. Introducing such an ordering appears un-

suitable for our application, as it would add another uncertainty, so we prefer omitting only such clauses which are guaranteed to be irrelevant. Also, the method of Couchot and Hubert is intended for clause sets where any proof is acceptable. Initially only the few most relevant clauses are selected, and if no proof is found, then increasingly less relevant clauses are selected for subsequent proof attempts. In LogAnswer we know that the query clause must be involved, so while we have no degrees of relevance, our selection is applied in a more goal-directed manner based on the obligatory query.

The meta-prover SInE 0.3[12] is another example for a prover which selects axioms. It uses more complex relevance criteria than E-KRHyper to select a subset of the logical input; the selected axioms are then passed on to the theorem prover E [50] for the actual inference processing. SInE first sorts all symbols of the first-order logic input by the number of axioms in which the respective symbol occurs. After that, SInE iteratively selects clauses, starting with the conjecture, and then adding clauses whose rarest symbol occurs in an already selected clause. The main difference between our approach and SInE's axiom selection is that the latter is based both on predicate and on function symbols, whereas E-KRHyper only considers the predicates. The method of SInE is highly effective: in 2008 the system won the CASC division for large theories (LTB), and on average it required the shortest time for a proof in our theorem prover evaluation, detailed in Section 11.3. Nevertheless it is impractical to adopt the SInE approach in LogAnswer, for several reasons. We aim to minimize all preprocessing, by making as much use of the background knowledge shared between proof attempts, and hence by distributing the computationally intensive aspects of preprocessing into the startup phase of LogAnswer. Here the duration of these operations is practically irrelevant, as it has no effect upon the response times for the handling of user questions. If we were to order the symbols by SInE's method, we would have to recompute this ordering for each proof attempt, as the logical representation of the candidate text passage can change the frequency of each symbol. With our own method, the only changes in the clause set between proof attempts involve the query conjecture, which is always relevant, and the units from the candidate text passage. As the finite set of MultiNet predicates is known beforehand, E-KRHyper can compute the solely predicate-based dependencies in the startup phase of LogAnswer without having to know which specific units the candidate passages will add. Once these units are added, E-KRHyper recognizes their precomputed dependencies by their predicate symbols. This keeps the individual dependency-related computation for each query to a minimum. Furthermore, dependencies based on function symbols are problematic once logic extensions are introduced. For example, using a superposition-based equality treatment like that of E-KRHyper, a unit clause $X = a$ can introduce the symbol a into virtually any axiom by term rewriting, making it difficult to state any meaningful function-based dependency which could allow useful axiom selection. A purely predicate-based partitioning mechanism appears more suitable for our purposes.

A noteworthy benefit of our partitioning mechanism at this time is that it allows us to load two variants of the background knowledge base into E-KRHyper. This is useful for dealing with the two question categories mentioned in Section 3, *factoid* vs. *definition*. The standard treatment of a question in LogAnswer during its translation into first-order logic is to normalize synonyms by selecting a canonical representative for each group of synonymous words. The occurrences of lexical constants in reasoning rules which constitute the logical background knowledge are normalized the same way. This lowers the amount of unique symbols in this knowledge base and minimizes the number of clauses, since no variants for different synonyms are required. However, this approach can lead to unwanted results when handling definition questions, where the original word from the question should be preferred over any synonym. Consider for example the question *'What is the Morning Star?'* With synonym normalization it would be possible to find an answer *'The Evening Star is the Venus.'* The loss of the original term in the answer could be confusing for the reader. Therefore there is no normalization for definition questions, and an alternative logical representation of the background knowledge provides variant clauses which account for synonyms. The two sets use distinct predicate symbol variants differentiated by markers, i.e. a predicate P is notated as P^f

[12] **S**UMO **In**ference **E**ngine; there are no publications about SInE yet, but the system and its description are available at `http://www.cs.man.ac.uk/~hoderk/sine`.

in the background knowledge for factoid question handling and as P^d in the set for definition questions. The query literals and the predicates from the answer candidate are marked according to the question category, and the partitioning mechanism then ensures that only the correct subset of clauses participates in the deduction process. This way the variant knowledge bases can share the same indexing structures within the prover. It is not necessary to load and unload the correct background knowledge as needed, or to have two prover instances running within LogAnswer, one for each question category.

9. Prover Result Evaluation

Given that many supporting text passages will be analyzed for a query, the results of E-KRHyper must be subjected to further evaluation in order to select the best answers that are eventually presented to the user.

If a proof of the logical representation of the question from the representation of the supporting passage and the background knowledge succeeds, then the passage is known to contain the requested information. However, if a proof of the full query fails and the system must fall back to relaxation, then the computed answer substitution is only logically valid for the reduced query fragment and not necessarily correct for the full query. The question thus arises how reliable and useful the results obtained by relaxation will be in practice, since they are not justified from a logical perspective. The answer to this question depends on the application. If the system is to find proofs of mathematical theorems, then applying relaxation does not make sense, since one needs results that are guaranteed to be correct on formal grounds. On the other hand, in a concrete application like question answering, it is sufficient and still very useful if the answers have a high probability of being correct. The effect of the number of relaxation steps on the correctness probability of the computed answer binding must be learned from a training corpus (which encodes statistical characteristics of the domain not covered by the logical model).

To this end, the shallow features mentioned in Section 5 are combined with the following logic-based features that depend on the relaxation result determined by the prover (see [22,23] for details):

- *skippedLitsLb*: Number of query literals skipped in the relaxation proof.
- *skippedLitsUb*: Number of skipped literals, plus the number of literals with unknown status (neither skipped nor proved).
- *litRatioLb*: Relative proportion of actually proved literals compared to the total number of query literals, i.e. $1 - skippedLitsUb/allLits$.
- *litRatioUb*: Relative proportion of potentially provable literals vs. all query literals, i.e. $1 - skippedLitsLb/allLits$);
- *npFocus*: The queried *FOCUS* variable was bound to a constant which corresponds to a nominal phrase in the text.
- *focusEatMatch*: The answer type of the answer binding found by the prover matches the expected answer type.
- *focusDefLevel*: This feature is relevant for definition questions. A value of $focusDefLevel = 2$ indicates that the answer binding found by the prover corresponds to an apposition, and $focusDefLevel = 1$ occurs if the answer binding corresponds to a noun phrase involving a relative clause.

The same machine learning technique also used for shallow passage reranking is applied to this enhanced extended feature set in order to refine the score of each passage that has undergone logical processing; see the following Section 10 for details.

Due to time restrictions or due to a failure of linguistic analysis, it is possible that a passage will not be processed by the prover. In this case the passage is useless if the user is interested in exact answers only, since the substitution determined by the prover forms the basis for answer extraction. However, if the user has opted to receive answers in the form of text snippets, then a passage that lacks logical validation can also be useful. In this case, the passages that lack logical validation are also considered for answer selection, based on the score computed from the shallow features only. The five best passages from the total pool are then chosen 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 kernel within the semantic network underlying the candidate passage is used to phrase the actual answer which can be presented to the user. In the

case that the same answer is extracted from several snippets, an aggregation method is utilized in order to determine the joint evidence for the answer judging from all supporting snippets. The use of this method can significantly improve the results of answer selection [20].

We take a look at one of the candidate passages for which E-KRHyper will terminate by continuing our running example: *'Das Getränk wurde von John Stith Pemberton, dem Erfinder von Coca-Cola, kopiert.'*[13] The logical representation[14] of the passage is shown in Figure 8.

No relaxation is required for this query, as every query literal can be disproven. In the resulting proof the *FOCUS* variable is instantiated by the constant *c31*, 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 *John Stith Pemberton*.

Some answers generated in this way are discarded immediately. For example, there is a test for trivial answers which only repeat terms from the query (*'Who is Virginia Kelley?'* - *Virginia Kelley*) and also a test for non-informative answers (*'the mother'* instead of *'the mother of Bill Clinton'*). Of those answers passing the sanity checks, the five best in the aforementioned ranking are selected and then displayed together with the supporting passages.

10. Machine Learning Issues Related to Interpreting Relaxation Results

In the previous sections we have mentioned two situations where LogAnswer applies machine learning. The first is during the passage retrieval (see Section 5), in which machine learning evaluates textual passages according to shallow syntactic features and then ranks them by the computed aggregated scores. The other use is during the ranking of the prover results for the final answer generation (see Section 9). Here the score from the earlier ML evaluation is combined with logical features of the refutational proofs in order to find the most suitable answers. In both cases the ML technique applied is the same. This section details the method used in LogAnswer.

Decision trees were chosen as the basic models for learning prover result evaluation since they can handle irrelevant and dependent attributes and since they allow very fast evaluation in the running system once learned. We need class probabilities (instead of a binary classification) for reranking, and it is obvious how to use a decision tree as a probability estimation tree (PET) based on the relative frequency p_j of the YES class in all training instances at a given leaf e_j. Since all features in our application are numeric, we can restrict attention to binary trees with branching conditions of the form $x_\alpha \geq \theta$, where α is one of the attributes and $\theta \in \mathbb{R}$ is a threshold, see Fig. 9 for an example. Following Provost and Domingos [46], we use Laplace smoothing and bagging (bootstrap aggregation of ensembles of trees with averaging of individual results [10]) in order to improve probability estimates. The special characteristics of the passage reranking task necessitated some refinements of this basic approach.

Stratified Bagging There is a low frequency of positive cases (correct passages) compared to negative examples – the precision of passage retrieval is typically 3% or less. In order to get more stable results despite this imbalance, we use a stratified variant of bagging. That is, the proportion of positive and negative examples in the test set are maintained by a separate subsampling of the positive and negative cases.

Support for Monotonicity Specifications The qualitative effect of the chosen features on the probability estimates is often known in advance. For example, it is reasonable to assume that the correctness probability of a support passage can be reasonably assumed to increase monotonically with the number of proved literals.[15] Hence let MON $\subseteq \{1, \ldots, n\}$ be a subset of the given (numeric) attributes declared as showing a positive effect on the probability estimates, where n is the total number of features.[16] We say that a PET τ re-

[13] *'The beverage was copied by John Stith Pemberton, the inventor of Coca-Cola.'*

[14] Actually, only the fragment of the representation which models the relational structure.

[15] Potharst et al [44] also consider the problem of constructing monotone decision trees. However, they target only on monotone classifications, while our approach targets at monotone probability estimates.

[16] The implementation also allows declaring attributes with a negative effect on probability estimates. These attributes can be handled in complete analogy to the non-decreasing case (it is sufficient to reverse all inequalities).

$$sub(erfinder.1.1, mensch.1.1) \wedge subs(c40, erfinden.1.1) \wedge obj(c40, c37) \wedge agt(c40, c31) \wedge temp(c39, past.0)$$
$$\wedge subs(c39, kopieren.1.1) \wedge agt(c39, c31) \wedge obj(c39, c27) \wedge sub(c38, name.1.1) \wedge val(c38, coca\text{-}cola.0)$$
$$\wedge sub(c37, erzeugnis.1.1) \wedge attr(c37, c38) \wedge val(c34, pemberton.0) \wedge sub(c34, familienname.1.1)$$
$$\wedge *tupl(c33, john.0, stith.0) \wedge sub(c32, vorname.1.1) \wedge val(c32, c33) \wedge sub(c31, erfinder.1.1)$$
$$\wedge attr(c31, c34) \wedge attr(c31, c32) \wedge sub(c27, getraenk.1.1)$$

Fig. 8. Logical passage representation

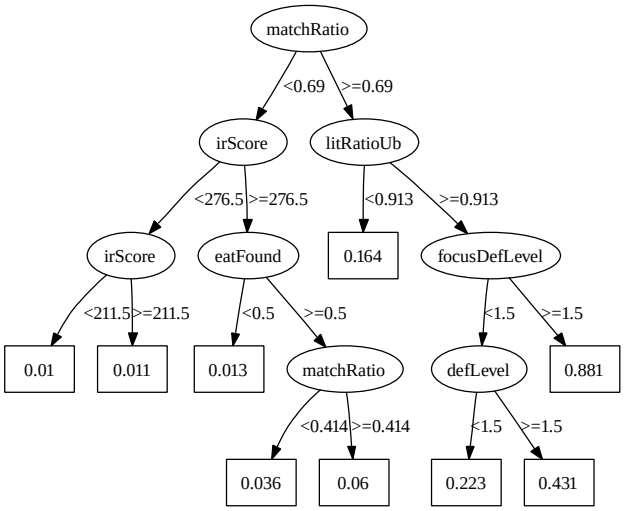


Fig. 9. Example of a probability estimation tree; see Sections 5 and 9 for documentation of the features. The tree was constructed by the PET induction method described here, declaring *matchRatio*, *irScore*, and *litRatioUb* as having a positive effect on probability estimates. In this case, this resulted in an increasing sequence of probability estimates at the leaves.

spects the monotonicity requirements expressed by MON if for all $\alpha \in$ MON, $(x_1, \ldots, x_n) \in \mathbb{R}^n$ and $x'_\alpha \geq x_\alpha$, it holds that $p^\tau(x_1, \ldots, x_n) \leq p^\tau(x_1, \ldots, x_{\alpha-1}, x'_\alpha, x_{\alpha+1}, \ldots, x_n)$, where $p^\tau : \mathbb{R}^n \to [0, 1]$ is the mapping from the inputs to probability estimates at the corresponding leaves of the tree. We use this criterion to constrain the admissible splits when inducing the decision tree.

Notice that splitting one leaf can now affect the allowable splits of all other leaves of the tree. While most approaches to decision tree induction work strictly locally, considering one leaf at a time, these global dependencies forced us to implement the tree induction in such a way that all leaf nodes of the tree constructed so far are considered in parallel when determining the next split.

Group-Sensitive Learning In the question answering context, the training data is naturally grouped by questions. There is often a strong variation in the number of correct answers available for these questions, which can be problematic for some learning techniques. In order to balance the effect of learning evenly among the questions from which the data was sampled, a group-sensitive splitting criterion is needed. The criterion should also reflect that the top-ranked results for each question are most important. The usual splitting criteria for learning decision trees treat all items equally – they do not directly maximize the quality of the few top-ranked results actually shown to the user. A more appropriate criterion should keep track of the k highest-ranked correct results for each question and encourage splits that further improve the rank of these results.

Hence let Q be the set of all questions from which the training items were sampled. For a question $q \in Q$, let $X(q)$ be the set of training items

associated with q and yes_q the number of positive training items in $X(q)$, i.e. the number of training items that correspond to a correct passage. Suppose we are interested in the k best results. If the number of correct results for the question is smaller than k, then it only makes sense to keep track of the correct answers that are actually available. Depending on the question, the following definition of the SC_{kmrr} splitting criterion therefore restricts attention to the $k_q^* = \min\{k, \text{yes}_q\}$ correct results with the highest ranks:

$$\text{SC}_{\text{kmrr}}(\tau) = \frac{1}{|Q|} \sum_{q \in Q} \sum_{i=1}^{k_q^*} \frac{w_{i,k_q^*}}{\text{rank}_{q,i} - i + 1} \qquad (8)$$

where $\text{rank}_{q,i}$ is the rank of the ith best correct result for question q in the PET τ, and w_{i,k_q^*} is the weight associated with this ith best correct result given a total of k_q^* considered positive results for question q. We use weights with a linear decay:

$$w_{i,k_q^*} = \frac{2(k_q^* - i + 1)}{k_q^* \, (k_q^* + 1)} \, . \qquad (9)$$

The ranks are obtained by sorting the training items $X(q)$ for each question q in decreasing order of estimated correctness probabilities. In the case that several items share the same probability score, we assume a pessimistic ordering (i.e. all incorrect results come first, followed by the correct ones).

Coupled Learning As explained in Section 9, the LogAnswer system must be able to merge passages with a logical validation (evaluated by a PET τ_{d} trained on both deduction-based and shallow features) and passages that lack logic-based features (e.g. due to a failed linguistic analysis), handled by a PET τ_{sh} for shallow features only. In order to ensure that the probability estimates determined by τ_{d} and τ_{sh} are commensurable, we adopt a *coupled learning* approach. That is, τ_{d} and τ_{sh} are induced in parallel with a global splitting criterion $\text{SC}_{\text{kmrr}}(\tau_{\text{d}}, \tau_{\text{sh}})$ defined across the trees. For a question q used for training, let us denote by $X_{\text{d}}(q)$ the training items generated from parseable candidate passages (in this case the logical representation of the passage is known, and logic-based features can be computed); and let $X_{\text{sh}}(q)$ be the set of training items for passages with a failed parse (in this case, the logical representation of the passage cannot be determined, so only shallow features are available). The combined splitting criterion $\text{SC}_{\text{kmrr}}(\tau_{\text{d}}, \tau_{\text{sh}})$ is defined like (8), but $\text{rank}_{q,i}$ is now determined from all correct results in the merged set $X(q) = X_{\text{d}}(q) \cup X_{\text{sh}}(q)$. Given the scores of all possible splits, the coupled learning technique then extends only one of the two PETs in each induction step, depending on whether the best split for τ_{d} or for τ_{sh} results in a larger increase of $\text{SC}_{\text{kmrr}}(\tau_{\text{d}}, \tau_{\text{sh}})$.

Controlling the Tree Size Decision tree induction is usually followed by a pruning stage that serves to avoid overfitting. The proposed method for inducing PETs by a global search for the best split permits a very simple approach to pruning: it creates a sequence of trees with increasing size such that the next tree in the sequence achieves the maximum increase of the quality score possible by splitting one leaf of the predecessor in the tree sequence.[17] Therefore one can simply stop the tree induction after a given number of splits. For coupled learning, we use a global limit S_G on the total number of splits, as well as a local limit S_L on the number of splits for each individual tree. The coupled induction ends once the number of induction steps reaches the global limit S_G, and the resulting PETs are then pruned back to the specified size by keeping only nodes constructed by the first S_L splits for each considered tree. For LogAnswer, suitable size limits of $S_L = 20$ splits per tree and $S_G = 40$ combined splits were determined experimentally.

11. Evaluation

In the following, we present an evaluation of the LogAnswer system from three perspectives. First, we present results based on question sets from the QA@CLEF evaluations which illustrate the overall performance of LogAnswer in typical QA problems and permit a comparison with other state-of-the-art systems. Then, we present a systematic study of using LogAnswer for passage reranking, using different system configurations and parameters of the proposed machine learning method. These experiments provide some data on the following questions: Do the obtained results profit from the use of logic? Does the use of relaxation

[17] Note that a sequence with these properties is not obtained if local split selection is used, as in regular C4.5.

further improve results or is it sufficient to try one proof for the full query? Can the quality of results be improved by combining logic-based results and results for passages with a failed parse? The latter aspects are especially important in an open-domain setting with possibly ungrammatical texts and missing background knowledge. Finally, we present a detailed evaluation of prover performance. Here, we are interested in the actual effect of the optimizations of the E-KRHyper prover for use in the LogAnswer system, as described in Section 8. We also present a performance comparison with other state-of-the-art ATP systems on typical problems encountered by LogAnswer.

11.1. Evaluation of the Overall System

The LogAnswer system is continually evaluated on different sets of test questions. The question answering track of the CLEF system evaluation campaign, QA@CLEF, is the main platform for evaluating and comparing QA systems for languages different from English. In the following, we briefly reconsider the QA@CLEF task and the types of questions considered in QA@CLEF 2007 [17] and 2008 [15].

The basic types of questions that are covered are factoid questions asking for a concrete person, organization, location, date, or similar entity with an expected answer type that can often be inferred from the question. Apart from the bulk of factoid questions, there is a smaller number of definition questions that ask for the main characteristic of an entity of interest. There is also a handful of so-called list questions that must be answered by an enumeration of items. Additional complexity is added by questions with a temporal restriction, such as *'Who was President of Russia in 1994?'* These types of questions reflect what is still challenging today, given the state of the art of QA technology.

In order to mimic a question answering dialogue, the questions are grouped by topics (like *'Wassily Kandinsky'*). Follow-up questions in a topic group can then contain pronouns and other anaphoric constructions that refer to the topic mentioned in one of the previous questions. All questions must be answered from a document collection that comprises a news corpus and a snapshot of the German Wikipedia.

The following results are obtained by the current LogAnswer system[18] on the 200 test questions of QA@CLEF 2008 with German as the target language:

- 61 right answers (precise phrase-sized answers fully justified by the chosen snippet);
- 9 inexact answers (typically answer phrases that contain the requested information but also some irrelevant material, or incomplete enumerations in the case of a question asking for a list of items);
- 2 unsupported answers (correct answers with a wrong snippet or with a snippet that does not fully justify the answer).

For comparison, the best system for German [48] that took part in the QA@CLEF 2008 system evaluation was able to find 74 right answers, the second best system for German [26] had 46 right answers. An early prototype of LogAnswer that took part in the QA@CLEF 2008 evaluation in May 2008 only found 29 right answers [23]. The performance leap of the current system (61 right answers) compared to the earlier prototype is mainly due to the new machine learning technique and due to refinements of the method used for extracting answer phrases given logical answer substitutions. With these improvements, LogAnswer is now competitive with other state-of-the-art systems in the QA@CLEF evaluations.

In general, one can state that only those systems achieve top scores in QA@CLEF that use at least some form of structural check of the question against the supporting text passage. In LogAnswer, this check is accomplished on the semantic level by trying a proof of the question representation from the representation of the snippet and the available background knowledge. However, other techniques for accomplishing such a test, like graph matching techniques applied to dependency trees, are also successfully used. For example, the best system for German in QA@CLEF 2008 relied on a subsequence kernel approach, applied to (simplified skeletons of) dependency trees [48].

Despite the quick successes achieved by such pure machine learning approaches, we believe that

[18] Revision 0.8, 2009-06-01. Note that for these experiments, only the QA@CLEF 2007 question set was used for obtaining training data for the machine learning method, while in normal operation, LogAnswer uses a model learned from both the QA@CLEF 2007 and 2008 question sets.

the logic-based framework has more potential in the long run since it offers a clear-cut interface for integrating background knowledge, be it simple semantic relations or elaborate ontologies. For graph matching techniques, by contrast, it is not clear how to integrate an existing ontology, for example.

A latent modeling of semantic relationships, as achieved by Latent Semantic Analysis (LSA) [30, 13], can capture some associations between word meanings without the need for manual knowledge acquisition. LSA has been applied for reranking answer passages in question answering [54] but it is insensitive to the structure of question and support passage. For example, assuming that 'was' and 'by' are eliminated as stop words, LSA cannot capture the very different meaning of the following two questions: *'Who shot Harvey Lee Oswald?'* (answer: *'Jack Ruby'*), and *'Who was shot by Harvey Lee Oswald?'* (answer: *John F. Kennedy*). Due to these limitations, LSA can not replace an explicit modeling of background knowledge and a technique that accomplishes a structural comparison between question and candidate passage.

The second best system for German in QA@CLEF 2008, IRSAW [26], uses specialized answer extractors (e.g. pattern-based techniques and named entity recognition) for extracting answer phrases that are then validated and merged by applying the logical core of LogAnswer. IRSAW outperformed the first LogAnswer prototype because the logic-based answer extraction of LogAnswer was not yet mature at the time of QA@CLEF 2008 participation. In the meantime, these systems achieve a similar result quality.

LogAnswer has also been tested on the development questions of the QA task of CLEF 2009, ResPubliQA.[19] This task requires the QA system to answer questions that refer to JRC-Acquis[20], a collection of documents related to EU administration. The answer must be given by specifying a paragraph of a document in the collection that contains the requested information. Only a single paragraph can be selected for each question. For evaluating LogAnswer, we have translated the development questions of ResPubliQA into German. Then, the system was run on the translated questions, in this case based on models trained on the QA@CLEF 2007 and 2008 questions. In this experiment, the current LogAnswer system successfully determines the requested paragraph (at top rank) for 57 of the 100 development questions. The ResPubliQA evaluation is not finished yet, so we cannot compare these results to those of other systems. The achieved success rate of 57% correct results at top rank looks promising, though, and it is consistent with the success rate of LogAnswer in other passage reranking tasks (see the following section).

[19] `http://celct.isti.cnr.it/ResPubliQA/`
[20] `http://langtech.jrc.it/JRC-Acquis.html`

11.2. Interpreting Relaxation Results

In the following, we present a more detailed evaluation of the quality of results achieved by the current LogAnswer prototype, and in particular on the ability of the proposed ML technique to determine a useful reranking of the candidate passages based on relaxation results. Since we are interested in the reranking quality here and not in the success rate of the subsequent extraction of answer phrases, we will focus on the ability of the system to find correct answer passages, which has not yet been assessed for the current setup. The questions of QA@CLEF 2007 [17] with German as the target language were used for generating the training set, and the corresponding questions for QA@CLEF 2008 [15] were used for testing.

Description of the Training Set As mentioned earlier, the QA@CLEF 2007 test set for German contains 200 questions organized by topic, with some of these questions containing anaphoric references (by pronouns or nominal anaphors) to mentions of the topic in previous questions in the topic group. In order to obtain high-quality training data, we manually exchanged each anaphoric question with a resolved question in which the anaphoric expression is replaced by the proper antecedent. In the current experiment, we restrict attention to factoid questions only (i.e., questions that ask for a single specific fact). Definition questions were not considered since the QA@CLEF 2007 question set for German does not contain enough definition questions to provide a good basis for applying ML techniques. Moreover, the decision if a description of something qualifies as a definition depends on additional criteria not covered by a logical validation.

There are 164 factoid questions in the QA@CLEF test set for German. Due to parsing errors, three of

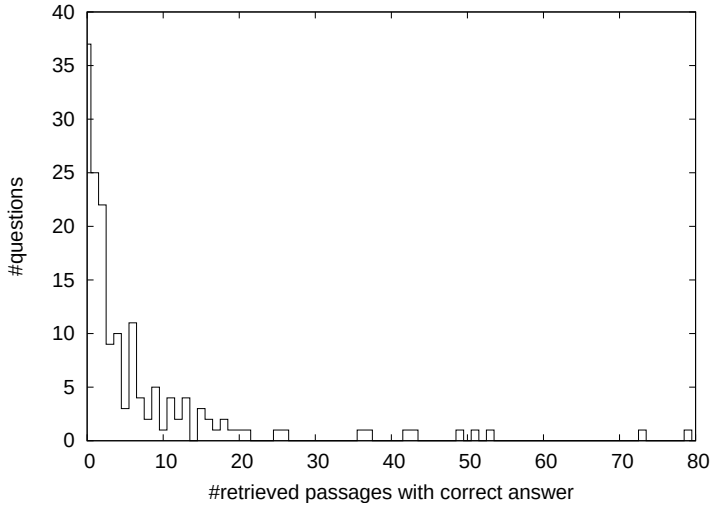


Fig. 10. Number of questions in the full training set with a given number of correct candidate passages

these questions had to be discarded. The remaining 161 factoid questions were used for generating the training set. To this end, the LogAnswer system was run on these questions, using the IR-SAW retrieval backend. This process resulted in a total of 31,091 support sentences to be retrieved (9,208 candidates with a successful parse). The retrieved passages were then annotated for containment of a correct answer. The annotation revealed that 1,140 (3.67%) of all retrieved candidates are correct (parseable passages: 4.44%; non-parseable passages: 3.34%). These averages are misleading, however, since there is a strong imbalance in the number of positive/negative cases for different questions, as shown by Fig. 10. There is one question with 79 correct candidate passages, and one question with 73 correct passages, but the median of the number of correct passages is 2 (i.e. two or less passages with a correct answer are available for half of the question). It is this imbalance that makes it necessary to use a group-sensitive ML technique. Considering only the 9,208 parseable passages that permit a logical validation by applying the prover, there are even less correct candidate passages to select from. In this case, the median of the number of parseable passages with a correct answer drops to 1 (i.e. for half of the questions, at most one of the passages that can be processed by the prover contains a correct answer), and there are 64 questions without any correct parseable passage at all (compared to 37 considering all retrieved candidates). This demonstrates the need to merge passages with a deep validation and (non-parsed) passages with a shallow-feature based validation in order to avoid a loss in the number of questions that can potentially be answered.

Description of the Test Set The questions of QA@CLEF 2008 track for German were used for testing; see [15]. In this case, there were 160 factoid questions of interest, and 40 questions of other types that were not considered in the current experiment. We discarded 10 so-called NIL questions in the test set (i.e., questions that deliberately find no answer in the document collection) since the treatment of these questions is not of interest here. Anaphoric questions were resolved automatically using the coreference module of the WOCADI parser [25], and 7 questions with wrong results of coreference resolution or spelling errors were discarded since the treatment of these phenomena is outside the scope of our project. The remaining 143 questions were used for generating test data. In this case, the retrieval resulted in a pool of 27,712 candidate passages total, including 832 correct candidates (3.0%). A full parse is available for 8,428 of the retrieved passages, and 335 of these parseable passages contain an answer (4.0%). The median of the number of correct candidates per question was 3 in this case (for the full pool of candidates); the median was 1 considering parseable candidates only. There are 127 questions with at least one correct candidate passage retrieved, but only 95 questions that can potentially be answered from parseable passages only.

Description of Tested Models A total of 9 passage reranking models was tested. The model normally used by LogAnswer, labeled CMT, was trained by stratified bagging of 10 monotone PETs. In order to assess the effect of monotonicity specifications, we also trained a model CNT with no monotonicity checks. The $\mathrm{SC}_{\mathrm{kmrr}}$ criterion was generally used for selecting the best splits, based on linear weighting and a default window width of $k = 3$. However, variants of the basic model with $k = 1$ (CMOT) and $k = 5$ (CMFT) were also tested. The maximum relaxation depth was normally limited to 3 relaxation cycles (indicated by the letter 'T'), but for training the CMZ model, a zero relaxation limit was used. In this case, no relaxation takes place, but the prover can still return partial prover results. Finally, a shallow-only model SH involving no logical processing at all was generated. In order to make optimal use of both parseable and non-parseable candidates, the coupled learning approach was normally used (indicated by the letter 'C'), but we also tried independent training of two PETs, with the logic-based model trained on the parseable passages using both logic-based and shallow features, while the model used for shallow feature based reranking was either trained on non-parseable passages (BMT model), or on all retrieved passages (AMT model). The baseline for all these models is provided by the original ranking of the passages determined by the passage retrieval system. The corresponding baseline model that always returns the original retrieval score is denoted by IR.

Description of Quality Metrics The test set consists of factual questions that can be fully answered by a *single* text passage containing the queried fact.[21] Therefore the position of the first correct answer in the result list for a question q, i.e. $\mathrm{rank}_{q,1}$, is of special importance, since it corresponds to the effort of a user who searches sequentially for a correct answer in the result list. The mean reciprocal rank (MRR) is a quality metric that takes the values of $\mathrm{rank}_{q,1}$ for all considered questions into account:

$$\mathrm{MRR} = \frac{1}{|Q|} \sum_{q \in Q, \mathrm{yes}_q > 0} \frac{1}{\mathrm{rank}_{q,1}}.$$

In order to obtain deterministic results when several results share the same score, we again assume 'pessimistic' tie breaking (i.e. all wrong results are shown first). Therefore the actual results of the models in practice will be slightly better than the shown MRR scores based on a worst-case tie breaking.

Since LogAnswer displays only a limited number of results (usually $k = 5$), we are also interested in the number ANS_k of questions with at least one correct result displayed at rank k or better, i.e.

$$\mathrm{ANS}_k = \left|\{q \in Q : \mathrm{yes}_q > 0, \mathrm{rank}_{q,1} \leq k\}\right| .$$

ANS_k measures the number of answered questions from the perspective of a user who views a generated result page with k results. Recalling that there are only 127 questions with a correct candidate passage in the test set, ANS_k is bounded by 127 in the experiments.

Experimental Results The evaluation results for the considered models on the test set are shown in Table 1. Using the standard model of LogAnswer for factoid questions (CMT) as a reference, we can make the following observations.

- Comparing CMT (the model that allows up to three relaxation steps) with the CMZ model that uses logic but no relaxation, a positive effect of relaxation on the MRR can be observed. This means that the quality of the presented answers profits from the introduction of the relaxation mechanism.
- Though factoid questions ask only for a single specific fact, considering the $k = 3$ best answers during training (CMT) achieved better results than a direct training for the single best-ranked answer (CMOT).
- As shown by the results of CMT vs. CNT, the support for monotonicity specifications that describe the known positive or negative effect of certain features (like percentage of proved literals) on probability estimates, results in an improvement of all considered quality metrics.
- The coupled learning approach (CMT) that was introduced to obtain commensurable probability estimates for parsed vs. non-parseable snippets outperforms independent learning

[21] For that reason the *precision@k*, *MAP* (mean average precision), and *R-precision* metrics [1,33], which are quality measures for ranked retrieval seeking for multiple results, do not appear to be appropriate for assessing the quality of result rankings for factoid questions.

Model	MRR	ANS_1	ANS_2	ANS_3	ANS_4	ANS_5
CMT	0.57	67	80	89	96	99
CMZ	0.55	63	81	90	95	97
CMFT	0.55	63	78	89	92	96
CMOT	0.54	63	77	87	91	95
BMT	0.54	61	75	90	99	100
SH	0.54	59	76	88	95	101
CNT	0.53	60	76	88	90	95
AMT	0.52	58	73	90	94	97
IR	0.40	40	52	62	76	80

Table 1
Results of the models on the full test set (ordered by MRR)

(AMT and BMT). This combination is important since a robust QA system must be able to merge results for sentences with a failed parse. BMT appears to work slightly better than CMT for larger ANS_k windows, but this does not result in an overall MRR advantage.
- Comparing CMT and SH (the model based on shallow features only), there are 8 additional correct passages for ANS_1. This indicates a clear benefit of using logic with respect to the quality of the top-ranked results. While SH performs surprisingly well for larger ANS_k windows, this does not express in a better overall ranking judging from the MRR.
- Finally, all models outperform the original (tf-idf based) ranking of the passage retrieval system by a large margin.

11.3. Prover Performance Evaluation

For a theorem prover to be suitable as a reasoner within LogAnswer it must be able to quickly solve the typical logic problems which occur in a question answering implementation. These are characterized by a large number of axioms, of which only a few are relevant for a proof. For an evaluation of automated theorem provers we used the previously mentioned set of 1,805 problems from the QA@CLEF 2007 competition (see Section 4). Together with the background knowledge axioms, each pair of query and passage forms a logic problem. The original MultiNet representations of these problems are known to be theorems due to proofs obtained with a specialized MultiNet prover. We reduced this set to pure first-order logic (no extensions) in order to test the problems with E-KRHyper and other theorem provers, allowing a valid comparison of the systems. As first-order logic does not cover the full expressivity of MultiNet, some aspects of the original problems were lost in the translation and hence the logic representations are not necessarily theorems. The problem set is available from the LogAnswer-website[22] and a selection of the problems will be included in a future TPTP-release as a part of the CSR-domain (common sense reasoning). Apart from E-KRHyper we chose several ATP systems[23] which had performed well in the CASC competitions: E 1.0 [50], iProver 5.0c [29] and Vampire 10.0 [47]. Also tested was SInE 0.3 (see Section 8). All systems were run with their respective default settings as used in CASC. For E-KRHyper we made an additional test run where our prover used the same settings as within LogAnswer[24]. The time for solving one problem was

[22] `http://www.loganswer.de/resources/loganswer_tptp_problems.tar.gz`

[23] We also tested other ATP systems (Otter 3.3, Spass 3.0 [56]), which were unable to load the background knowledge base, apparently due to its size exceeding internal limitations of the provers. This illustrates the abnormity of the LogAnswer-related problems, as Spass 3.0 in particular is usually a well-performing prover for TPTP problems. In fairness it should be mentioned that we tested E-KRHyper on problems of the CASC LTB-division for large theories. These problems involve knowledge bases considerably larger than the background knowledge in LogAnswer, and we found that the act of loading the LTB knowledge into E-KRHyper would exceed the competition's time limit, thus producing no useful results under the given constraints. This will have to be addressed as the LogAnswer background knowledge grows and we include other knowledge sources (see Section 13).

[24] The optimized parameters affect, among other things, the term weight computation and the iterative deepening strategy of the hyper tableau construction. These settings were the same for all problems during the additional test

limited to 60 seconds. This included the time for loading the background knowledge into the theorem prover, as all provers operated in the usual fashion of being invoked for one problem and terminating with a result. While this time limit is shorter than the one imposed during CASC or the TPTP prover performance ratings, it is well above the time slots allowed for reasoning in LogAnswer, even when used in the QA@CLEF evaluation where response times are not as critical as in the web-based QA usage model. E-KRHyper neither used relaxation (see Section 7), as we wanted to measure the times required for valid proofs, nor did it use the server mode optimization of loading the background knowledge only once and sharing it for all problems (see Section 8), since this would have involved changing the test setup in a manner not supported by most other ATP systems. All tests where carried out on an Intel Q6600 system with 2.4GHz and 2GB RAM for each problem. The results are shown in Table 2.

E-KRHyper solves most problems and is the third-fastest system under default settings, closely approaching the speed of the fastest stand-alone prover Vampire 10.0. With optimized settings E-KRHyper almost halves the average time required for a proof and becomes the fastest stand-alone ATP, only beaten by the meta-prover SInE 0.3. The prover E 1.0 nearly solves as many problems as E-KRHyper, but requires considerably more time for each proof. The number of problems solved by E-KRHyper does not change under optimized settings. Even with default parameters there are just 12 outlier problems (0.7% of the 1,805) which are solved within the time limit but which require at least 20 seconds, and only one of these requires more than 40 seconds. The remaining 4.2% of unsolved problems appear to be too hard for them to benefit from the higher speed at optimized settings.

Reasons for the good performance of E-KRHyper can be found in the hyper tableaux calculus, which emphasizes unit operations and which can use multiple unit facts as premises for a single inference where a different calculus would need several inference steps. Another reason is the strategy implemented in E-KRHyper: the evaluation is executed in rounds, with each round exhaustively carrying out all inferences which involve the clauses from the input and those derived in the previous round, and the inference results are only added to the working set for use as premises at the start of the following round. In problems with comparatively flat proofs and large numbers of eventually mostly irrelevant clauses, this strategy ensures that the prover does not follow a futile inference chain for too long. This is unlike a prover which adds inference results to its working set with little delay, and which may then go on to use them as premises before older possibilities have been evaluated.

While the meta-prover SInE is even faster than E-KRHyper, it solves significantly less problems. Regarding logic extensions, SInE is inevitably limited to those of the ATP it employs, and it is unclear to which degree the relevance based selection criteria are compatible with extensions at all. However, the speed advantage of SInE is undeniable, and its axiom selection warrants the more detailed discussion in Section 8. When operating within LogAnswer, E-KRHyper reuses the background knowledge for all problems and does not have to reload this large knowledge base for each proof attempt, see Section 8. This would reduce the average time E-KRHyper requires for one proof by 0.9 seconds, resulting in an average proof time of 1.18 seconds. While this is still slower than SInE, it must be noted that the average proof times of both provers are still too high to usually allow for a full proof within LogAnswer's web-based usage model, and the supporting methods like relaxation remain a necessity.

12. Limitations

The field of open domain question answering has yet to produce a system which can reliably provide the performance most users would expect. All existing implementations exhibit severe limitations in practice, which is not surprising considering that an optimal QA system would have to approximate human intelligence in its textual understanding, still an elusive goal of AI research. At this point QA systems first and foremost aim to demonstrate the feasibility of the methods they implement.

It is also important to understand that the level of accuracy obtained by a QA system on the QA@CLEF 2007 and 2008 question sets cannot be

series for E-KRHyper. They were not individually adjusted for each problem.

ATP system	solved	avg. time
E-KRHyper 1.1.2 (D)	1,729 (95.8%)	3.80 sec
E-KRHyper 1.1.2 (L)	1,729 (95.8%)	2.08 sec
E 1.0	1,664 (92.2%)	30.82 sec
iProver 0.5c	1,373 (76.1%)	7.97 sec
SInE 0.3	1,311 (72.6%)	0.61 sec
Vampire 10.0	1,157 (64.1%)	3.73 sec

Table 2

ATP results on LogAnswer query set of 1,805 problems. The table lists the ATP system name, the number of problems solved by the ATP and the average time required for a successful proof. (D) indicates the results for E-KRHyper when run with default settings, whereas (L) marks the results when the parameters were optimized for LogAnswer problems.

expected when the system is confronted with real user queries. This is because the questions used in QA@CLEF are carefully chosen such that there exists a passage (typically even a single sentence) in the document collection that directly contains the answer to the question.[25] For ad-hoc user questions, this is often not the case. For example, the frequent use of pronouns and nominal anaphora in the texts (instead of repeating the name of a person every time it is referenced), can make it difficult to spot the relevant information in practice.

We have started to address this problem by adding descriptors obtained from coreference resolution of pronouns to the index in order to increase the chance that the passage retrieval stage will find a relevant sentence. However, the snippet representations that are visible to the prover are still based on a single sentence of the textual knowledge sources. This means that an answer cannot be derived from facts spread out over multiple sentences. Apart from the use of referential expressions mentioned above, the ability to handle several sentences simultaneously is also important when the answer is too complex to consist of a single phrase, for example in the case of definition questions or 'how-to' questions that ask for the description of a procedure.

On the other hand it is possible for a single answer candidate to contain multiple answers to the same question. For example, a sentence listing all members of The Beatles in 1964 would contain four valid answers to the question *'Who was a member of The Beatles?'* Usually only one proof per answer candidate will be considered, though, and after the first successful answer derivation the deductive processing will move on to the next candidate. As these situations are relatively rare, our current method seems preferable over rechecking each successful candidate for generally unlikely additional answers.

Another limitation concerns keeping the knowledge base current. The entire formalized knowledge of LogAnswer is the result of pre-processed sources. While periodically updated by new snapshots from Wikipedia, for all practical purposes our knowledge base must be considered to be static, as LogAnswer acquires no new information from outside the system at runtime. LogAnswer is therefore restricted to answering questions requiring encyclopedic knowledge only. Questions which need current and fluctuating knowledge (*'Is it raining in Paris right now?'*) cannot be answered using our knowledge base.

Most of these shortcomings are being addressed in the ongoing work on LogAnswer in the following conclusion.

13. 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 linguis-

[25] To be precise, the QA@CLEF 2008 test set also comprises a few so-called NIL questions known to find no answer in the corpus. These NIL questions serve to evaluate the ability of the system to detect missing information.

tic 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 a feedback loop to gradually relax the logical query. We have also shown how the relaxation results can be interpreted by a machine learning method suited for grouped data.

An evaluation of the approach based on questions from QA@CLEF 2008 confirms that the performance of our QA system profits from the automated reasoning capabilities of the logic prover and from the proposed relaxation technique. The use of logic is particularly effective for improving the top-ranked results. 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, utilizing the logic extensions provided by E-KRHyper. This will enable us to fully exploit the expressivity of the semantic networks in combination with automated reasoning.

Another area of future work concerns the aforementioned problem of synthesizing answers from multiple text passages, see the previous Section 12. At the moment an answer candidate is based on a single sentence. However, the facts which make up an answer may very well be distributed over several sentences within a text, or even scattered throughout the knowledge base. Derivation of such answers will involve the combination of multiple answer candidates. In the first place we expect this to pose a quantitative problem, but it may also raise questions necessitating further research.

We also plan to connect LogAnswer to other systems. On the one hand we want to make use of web services which provide current data regarding the weather, finances, timetables etc. Access to this kind of information would enable LogAnswer to handle questions which cannot be answered by encyclopedic knowledge, and thereby address the limitations imposed by our static knowledge base. As the data from web services is delivered in a structured form, little NL processing is required. Instead the information can be directly entered into the proof derivation. Difficulties arise from the lack of standardization among web services, requiring the implementation of many individual interfaces, and from the response times of web services, which make it unacceptable to have the prover halt its operation until the data has been delivered. An asynchronous approach and caching of web service data will be necessary.

On the other hand we want to connect LogAnswer to QA forums, i.e. communities on the web where users can ask questions and other users give the answers. Here LogAnswer can assume the role of a responder and deliver answers almost instantly in comparison to the wait for a human reply. As QA forums usually allow the forum participants to judge answers, this would give us a real-world evaluation of our system. Another possible role for LogAnswer in the context of QA forums is to semantically compare new questions with old questions from the forum archive. If an equivalent older question is found, LogAnswer can refer an inquirer to existing archived answers.

Outside of LogAnswer, the modifications to E-KRHyper may prove useful for an application to large logical knowledge bases in general. We are experimenting with the OpenCyc [31] ontology and the SUMO (Suggested Upper Merged Ontology) [38]. The sizes of these knowledge bases push the limits of automated theorem provers, and we are exploring the use of partitioning mechanisms to make the ontologies manageable. Likewise the ability to save and load states of the prover's internal clause storage can help in quickly proving series of conjectures in conjunction with a common large ontology, a task very similar to the usage model of E-KRHyper within LogAnswer.

References

[1] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. ACM Press, New York, 1999.

[2] Jon Barwise and Robin Cooper. Generalized quantifiers and natural language. *Linguistics and Philosophy*, 4:159–219, 1981.

[3] Peter Baumgartner, Ulrich Furbach, and Ilkka Niemelä. Hyper Tableaux. In *JELIA'96, Proceedings*, pages 1–17, 1996.

[4] Peter Baumgartner, Ulrich Furbach, and Björn Pelzer. Hyper Tableaux with Equality. In *Automated Deduction - CADE-21, Proceedings*, 2007.

[5] Bernhard Beckert and Christian Pape. 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.

[6] Johan van Benthem. Questions about quantifiers. *Journal of Symbolic Logic*, 49, 1984.

[7] D.G. Bobrow, C. Condoravdi, R.S. Crouch, V. de Paiva, R.M. Kaplan, L. Karttunen, T.H. King, and A. Zaenen. A basic logic for textual inference. In *Proceedings of the AAAI Workshop on Inference for Textual Question Answering*, Pittsburgh, PA, Jul 2005.

[8] Johan Bos. DORIS 2001: Underspecification, resolution and inference for discourse representation structures. In *ICoS-3 - Inference in Computational Semantics, Workshop Proceedings*, 2001.

[9] Johan Bos and Katja Markert. When logical inference helps determining textual entailment (and when it doesn't). In *Proc. of 2nd PASCAL RTE Challenge Workshop*, 2006.

[10] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[11] Jean-François Couchot and Thierry Hubert. A graph-based strategy for the selection of hypotheses. In *FTP'07, Int. Workshop on First-Order Theorem Proving*, Liverpool, UK, September 2007.

[12] Jon Curtis, Gavin Matthews, and David Baxter. On the effective use of cyc in a question answering system. In *Proceedings of the IJCAI Workshop on Knowledge and Reasoning for Answering Questions (KRAQ'05)*, pages 61–70, 2005.

[13] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science and Technology*, 41(6):391–407, 1990.

[14] Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database (ISBN: 0-262-06197-X)*. MIT Press, first edition, 1998.

[15] Pamela Forner, Anselmo Peñas, Iñaki Alegria, Corina Forăscu, Nicolas Moreau, Petya Osenova, Prokopis Prokopidis, Paulo Rocha, Bogdan Sacaleanu, Richard Sutcliffe, and Erik Tjong Kim Sang. Overview of the CLEF 2008 multilingual question answering track. In Peters et al. [42].

[16] Ulrich Furbach, Ingo Glöckner, Hermann Helbig, and Björn Pelzer. LogAnswer - A Deduction-Based Question Answering System. In *Automated Reasoning (IJCAR 2008)*, Lecture Notes in Computer Science, pages 139–146. Springer, 2008.

[17] Danilo Giampiccolo, Pamela Forner, Anselmo Peñas, Christelle Ayache, Dan Cristea, Valentin Jijkoun, Petya Osenova, Paulo Rocha, Bogdan Sacaleanu, and Richard Sutcliffe. Overview of the CLEF 2007 multilingual question answering track. In *Working Notes for the CLEF 2007 Workshop*, Budapest, Hungary, 2007.

[18] Ingo Glöckner. University of Hagen at QA@CLEF 2007: Answer validation exercise. In *Working Notes for the CLEF 2007 Workshop*, Budapest, 2007.

[19] Ingo Glöckner. Towards logic-based question answering under time constraints. In *Proc. of the 2008 IAENG Int. Conf. on Artificial Intelligence and Applications (ICAIA-08)*, pages 13–18, Hong Kong, 2008.

[20] Ingo Glöckner. University of Hagen at QA@CLEF 2008: Answer validation exercise. In Peters et al. [42].

[21] Ingo Glöckner, Sven Hartrumpf, and Johannes Leveling. Logical validation, answer merging and witness selection: A study in multi-stream question answering. In *Proc. of RIAO-07*, Pittsburgh, 2007.

[22] Ingo Glöckner and Björn Pelzer. Exploring robustness enhancements for logic-based passage filtering. In *Knowledge Based Intelligent Information and Engineering Systems (Proc. of KES2008, Part I)*, LNAI 5117, pages 606–614. Springer, 2008.

[23] Ingo Glöckner and Björn Pelzer. Combining logic and machine learning for answering questions. In Peters et al. [43]. (to appear).

[24] Sanda Harabagiu, Dan Moldovan, Rada Mihalcea, Mihai Surdeanu, and Vasile Rus. Falcon: Boosting knowledge for answer engines. In *Proceedings of the 9th Text Retrieval Conference (TREC-9)*, pages 479–488, 2000.

[25] Sven Hartrumpf. *Hybrid Disambiguation in Natural Language Analysis*. Der Andere Verlag, Osnabrück, Germany, 2003.

[26] Sven Hartrumpf, Ingo Glöckner, and Johannes Leveling. Efficient question answering with question decomposition and multiple answer streams. In Peters et al. [43]. (to appear).

[27] Sven Hartrumpf, Hermann Helbig, and Rainer Osswald. The semantically based computer lexicon HaGenLex. *Traitement automatique des langues*, 44(2):81–105, 2003.

[28] Hermann Helbig. *Knowledge Representation and the Semantics of Natural Language*. Springer, 2006.

[29] Konstantin Korovin. iProver — an instantiation-based theorem prover for first-order logic (system description). In *IJCAR '08: Proceedings of the 4th international joint conference on Automated Reasoning*, pages 292–298, Berlin, Heidelberg, 2008. Springer-Verlag.

[30] Thomas K. Landauer, Peter W. Foltz, and Darrell Laham. Introduction to latent semantic analysis. *Discourse Processes*, 25:259–284, 1988.

[31] Douglas B. Lenat. CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995.

[32] Johannes Leveling. IRSAW – towards semantic annotation of documents for question answering. In *CNI Spring 2007 Task Force Meeting*. Phoenix, Arizona, 2007.

[33] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[34] William McCune. Experiments with Discrimination-Tree Indexing and Path Indexing for Term Retrieval. *Journal of Automated Reasoning*, 9(2):147–167, 1992.

[35] William McCune. *OTTER 3.3 Reference Manual*. Argonne National Laboratory, Argonne, Illinois, 2003.

[36] Dan Moldovan, Mitchell Bowden, and Marta Tatu. A temporally-enhanced PowerAnswer in TREC 2006. In *Proc. of TREC-2006*, Gaithersburg, MD, 2006.

[37] Dan Moldovan, Christine Clark, Sanda Harabagiu, and Steve Maiorano. COGEX: A logic prover for question answering. In *Proc. of NAACL-HLT 2003*, volume 1, pages 87–93, Morristown, NJ, 2003.

[38] Ian Niles and Adam Pease. Towards a standard upper ontology. In *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001)*, 2001.

[39] Francis Jeffry Pelletier, Geoff Sutcliffe, and Christian Suttner. The Development of CASC. *AI Communications*, 15(2-3):79–90, 2002.

[40] Björn Pelzer and Ingo Glöckner. 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, 2008*, pages 71–80. CEUR Workshop Proceedings, 2008.

[41] Björn Pelzer and Christoph Wernhard. System Description: E-KRHyper. In *Automated Deduction - CADE-21, Proceedings*, pages 508–513, 2007.

[42] Carol Peters, Thomas Deselaers, Nicola Ferro, Julio Gonzalo, Gareth J.F. Jones, Mikko Kurimo, Thomas Mandl, Anselmo Peñas, and Vivien Petras, editors. *Working Notes for the CLEF 2008 Workshop*, Aarhus, Denmark, September 2008.

[43] Carol Peters, Thomas Deselaers, Nicola Ferro, Julio Gonzalo, Gareth J.F. Jones, Mikko Kurimo, Thomas Mandl, Anselmo Peñas, and Vivien Petras, editors. *Evaluating Systems for Multilingual and Multimodal Information Access: 9th Workshop of the Cross-Language Evaluation Forum, CLEF 2008, Aarhus, Denmark, September 17–19, Revised Selected Papers*, LNCS, Heidelberg, 2009. Springer. (to appear).

[44] Rob Potharst, Jan C. Bioch, and Thijs Petter. Monotone decision trees. Technical Report EUR-FEW-CS-97-07, Dept. of Computer Science, Erasmus University Rotterdam, 1997.

[45] John Prager, Eric Brown, Anni Coden, and Dragomir Radev. Question-answering by predictive annotation. In *SIGIR '00: Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 184–191, New York, NY, 2000. ACM Press.

[46] Foster Provost and Pedro Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52(3):199–215, 2003.

[47] Alexandre Riazanov and Andrei Voronkov. The design and implementation of Vampire. *AI Communications*, 15(2-3):91–110, 2002.

[48] Bogdan Sacaleanu, Günter Neumann, and Christian Spurk. DFKI at QA@CLEF 2008. In Peters et al. [42].

[49] José Saias and Paulo Quaresma. The Senso question answering approach to Portuguese QA@CLEF-2007. In *Working Notes for the CLEF 2007 Workshop*, Budapest, Hungary, 2007.

[50] Stephan Schulz. E - a brainiac theorem prover. *AI Communications*, 15(2-3):111–126, 2002.

[51] Mark E. Stickel, Richard J. Waldinger, and Vinay K. Chaudhri. A guide to SNARK, 2000.

[52] Geoff Sutcliffe and Christian Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

[53] Geoff Sutcliffe and Christian Suttner. The State of CASC. *AI Communications*, 19(1):35–48, 2006.

[54] David Tomás and Jesé L. Vicedo. Re-ranking passages with LSA in a question answering system. In *Evaluation of Multilingual and Multi-modal Information Retrieval. 7th Workshop of the Cross-Language Evaluation Forum, CLEF 2006, Alicante, Spain, September 20-22, 2006, Revised Selected Papers*, volume 4730 of *LNCS*, pages 275–279. Springer, 2007.

[55] Richard Waldinger and Jeff Shrager. Answering science questions: Deduction with answer extraction and procedural attachment. *AAAI Spring Symposium: Semantic Scientific Knowledge Integration*, 2008.

[56] Christoph Weidenbach, Renate Schmidt, Thomas Hillenbrand, Rostislav Rusev, and Dalibor Topic. System description: Spass version 3.0. In *Automated Deduction - CADE-21: 21st International Conference on Automated Deduction*, pages 514–520, 2007.