LogAnswer - A Deduction-Based Question Answering System (System Description)

Ulrich Furbach[1], Ingo Glöckner[2], Hermann Helbig[2], and Björn Pelzer[1]

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

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

Abstract. LogAnswer is an open domain question answering system which employs an automated theorem prover to infer correct replies to natural language questions. For this purpose LogAnswer operates on a large axiom set in first-order logic, representing a formalized semantic network acquired from extensive textual knowledge bases. The logic-based approach allows the formalization of semantics and background knowledge, which play a vital role in deriving answers. We present the functional LogAnswer prototype, which consists of automated theorem provers for logical answer derivation as well as an environment for deep linguistic processing.[3]

1 Introduction

Question answering (QA) systems generate natural language (NL) answers in response to NL questions, using a large collection of textual documents. Simple factual questions can be answered using only information retrieval and shallow linguistic methods like named entity recognition. More advanced cases, like questions involving a temporal description, call for deduction based question answering which can provide support for temporal reasoning and other natural language related inferences. Logic has been used for such semantic NL analysis in the DORIS [1] system, although this is aimed at discourse regarding a limited domain instead of open-domain QA. There are also several examples of logic-based QA systems (like PowerAnswer [2] and Senso [3]), as well as dedicated components for logical answer validation like COGEX [4] or MAVE [5]. However, most of these solutions are research prototypes developed for the TREC or CLEF evaluation campaigns, which ignore the issue of processing time.[4] For actual users, getting the answer in a few seconds is critical to the usefulness of a QA system, though. A QA system must achieve these response times with

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

[4] See http://trec.nist.gov/ (TREC), http://www.clef-campaign.org/ (CLEF).

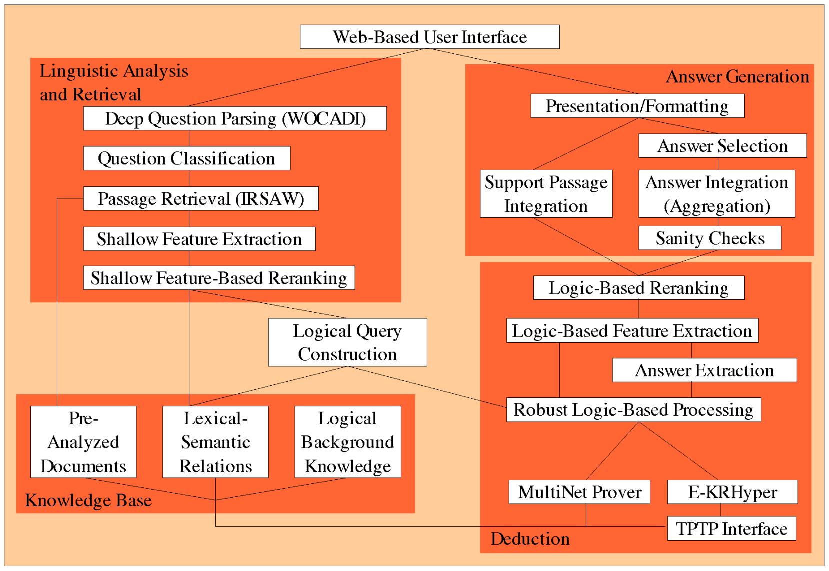
Web-Based User Interface
Linguistic Analysis and Retrieval
Deep Question Parsing (WOCADI)
Question Classification
Passage Retrieval (IRSAW)
Shallow Feature Extraction
Shallow Feature-Based Reranking
Logical Query Construction
Answer Generation
Presentation/Formatting
Answer Selection
Support Passage Integration
Answer Integration (Aggregation)
Sanity Checks
Logic-Based Reranking
Logic-Based Feature Extraction
Answer Extraction
Robust Logic-Based Processing
MultiNet Prover
E-KRHyper
TPTP Interface
Deduction
Pre-Analyzed Documents
Lexical-Semantic Relations
Logical Background Knowledge
Knowledge Base

Fig. 1. System architecture of the LogAnswer prototype

a knowledge base generated from tens of millions of sentences. A second challenge for logic-based QA is robustness. The processing chain of a logic-based QA system involves many stages (from NL via syntactic-semantic parsing to logical forms and knowledge processing and back to NL answers). Therefore fallback solutions like the usage of shallow features are needed to ensure baseline performance when one of the deep NLP modules fails, and gaps in the background knowledge must be bridged by robustness-enhancing techniques like relaxation.

2 Description of the LogAnswer System

The system architecture of the LogAnswer QA system is shown in Fig. 1. In the following we describe the processing stages of the system.

User interface The natural language question is entered into the LogAnswer web search box.[5] Depending on user preferences, the system answers the question by presenting supporting text passages only or alternatively, by presenting exact answers together with the supporting passage.

Deep Question Parsing The question is analyzed by the WOCADI parser [6], which generates a semantic representation of the question in the MultiNet formalism [7]. A question classification is also done in this phase, which currently

[5] The system is available online at `www.loganswer.de`.

discerns only definition questions (*What is a neutrino?*) and factual questions (*Who discovered the neutrino?*). While factual questions can be answered by logical means alone, definition questions need additional filtering in order to identify descriptions that are not only true but also represent defining knowledge.

Passage Retrieval The document collection of LogAnswer comprises the CLEF news collection and a snapshot of the German Wikipedia (17 million sentences total). In order to avoid parsing of documents at query time, all documents are pre-analyzed by the WOCADI parser. The resulting MultiNet representations are segmented into passages and stored in the IRSAW retrieval module [8], which uses the terms in the passage for indexing.[6] Given the query terms, IRSAW typically retrieves 200 (or more) passages as the basis for logical answer finding.

Shallow Feature Extraction and Reranking In order to avoid logical processing of all retrieved passages, LogAnswer tries to identify the most promising cases by reranking passages using shallow features (like overlap of lexical concepts, proper names and numerals of the question with those found in the passage). It is important that these features can be computed very quickly without the help of the prover. The machine learning approach and the set of shallow features are detailed in [9,10].

Logical Query Construction The semantic network for the question is turned into a conjunctive list of query literals. Synonyms are normalized by replacing all lexical concepts with canonical synset representatives.[7] For example, *Wie viele Menschen starben beim Untergang der Estonia?*[8] translates into the following logical query (with the $FOCUS$ variable representing the queried information):

$$\begin{aligned}&\mathrm{sub}(X_1, \mathit{estonia.1.1}), \mathrm{attch}(X_1, X_2), \mathrm{subs}(X_2, \mathit{untergang.1.1}), \mathrm{subs}(X_3, \mathit{sterben.1.1}),\\&\mathrm{circ}(X_3, X_2), \mathrm{aff}(X_3, FOCUS), \mathrm{pred}(FOCUS, \mathit{mensch.1.1})\,.\end{aligned}$$

Robust Logic-Based Processing As the basis for answer extraction and for improving the passage ranking, LogAnswer tries to prove the logical representation of the question from the representation of the passage and the background knowledge.[9] Robustness is gained by using relaxation: if a proof is not found within a time limit, then query literals are skipped until a proof of the remaining query succeeds, and the skip count indicates (non-)entailment [5,10]. For efficiency reasons, relaxation is stopped before all literals are proved or skipped. One can then state upper/lower bounds on the provable literal count, assuming that all (or none) of the remaining literals are provable.

[6] The current version of LogAnswer uses a segmentation into single sentences, but we will also experiment with different passage sizes (paragraphs and full documents).

[7] The system uses 48,991 synsets (synonym sets) for 111,436 lexical constants.

[8] *How many people died when the MS Estonia sank?*

[9] The background knowledge of LogAnswer comprises 10,000 lexical-semantic facts (e.g. for nominalizations) and 109 logical rules, which define main characteristics of MultiNet relations and also handle meta verbs like ‘stattfinden’ (take place) [5].

Answer Extraction If a proof of the question from a passage succeeds, then LogAnswer obtains an answer binding which represents the queried information. For finding more answers, the provers of LogAnswer can also return a substitution for a proven query fragment when a full proof fails. Given a binding for the queried variable, LogAnswer uses word alignment hints of WOCADI for finding the matching answer string, which is directly cut from the original text passage.

Logic-Based Feature Extraction For a logic-based refinement of relevance scores, LogAnswer extracts the following features, which depend on the limit on relaxation cycles and on the results of answer extraction:

– *skippedLitsLb* Number of literals skipped in the relaxation proof.
– *skippedLitsUb* Number of skipped literals, plus literals with unknown status.
– *litRatioLb* Relative proportion of actually proved literals compared to the total number of query literals, i.e. $1 - \mathit{skippedLitsUb}/\mathit{allLits}$.
– *litRatioUb* Relative proportion of potentially provable literals (not yet skipped) vs. all query literals, i.e. $1 - \mathit{skippedLitsLb}/\mathit{allLits}$.
– *boundFocus* Indicates that a binding for the queried variable was found.
– *npFocus* Indicates that the queried variable was bound to a constant which corresponds to a nominal phrase (NP) in the text.
– *phraseFocus* Signals that an answer string has been extracted.

Logic-Based Reranking The logic-based reranking of the passages uses the same ML approach as the shallow reranking, but the shallow and logic-based features are now combined for better precision. Rather than computing a full reranking, passages are considered in the order determined by the shallow feature-based ranking, and logical processing is stopped after a pre-defined time limit.

Support Passage Selection When using LogAnswer for retrieving text snippets which contain an answer, all passages are re-ranked using either the logic-based score (if available for the passage) or the shallow-feature score (if there is no logic-based result for the passage due to parsing failure or time restrictions). The top k passages are chosen for presentation ($k = 5$ for the web interface).

Sanity Checks When the user requests exact answers rather than snippets which contain the answer, additional processing is needed: a triviality check eliminates answers which only repeat contents of the question. For the question *Who is Virginia Kelley?*, this test rejects trivial answers like *Virginia* or *Virginia Kelley*. A special sanity check for definition questions also rejects the non-informative answer *the mother* (instead of the expected *the mother of Bill Clinton*), see [5].

Aggregation and Answer Selection The answer integration module computes a global score for each answer, based on the local score for each passage from which the answer was extracted. The aggregation method already proved effective in [5]. The $k = 5$ distinct answers with the highest aggregated scores are then selected for presentation. For each answer, the supporting passage with the highest score is also shown in order to provide a justification for the presented answer.

3 Theorem Provers of LogAnswer

The robust logic-based processing (see Section 2) has to merge contrasting goals: it has to derive answers from a logical knowledge representation using precise inference methods, but it must also provide these answers within acceptable response times and account for imperfections of the textual knowledge sources and their formalization. Thus a theorem prover must meet several requirements if it is to serve as the deduction component in LogAnswer.

Handling of Large Knowledge Bases Of high importance is the ability to work on the large set of axioms and facts forming the knowledge base, which will keep growing in the future. This includes the way these clauses are supplied to the prover. Theorem provers usually operate on a single-problem basis: the prover is started with the required clauses and then terminates after a successful proof derivation. For LogAnswer this approach is impractical. In order to achieve a usability comparable to conventional search engines, the system should spend all of the available processing time for actual reasoning and not for loading the background knowledge into the prover. Since any two query tasks use the same background knowledge and only differ in a few clauses representing the query and a text passage, a LogAnswer prover should be able to stay in operation to perform multiple query tasks, loading and retracting the query-specific clauses while keeping the general knowledge base in the system.

Relaxation Loop Support The prover must also support the robustness enhancing techniques, in particular by providing guidance to the relaxation loop. The large knowledge base with its imperfections often causes the prover to reach the time limit, where LogAnswer will interrupt the reasoning and relax the query. The prover must then report details about its failed proof attempt so that the relaxation loop can select the query literal most suited for skipping.

Answer Extraction Support Finally, if a proof succeeds, then the prover must state any answer substitutions found for the *FOCUS* variable.

The current LogAnswer prototype includes two theorem provers for comparison purposes in the development phase.

The MultiNet Prover The prover of the MultiNet toolset[10] is based on SLD resolution and operates on range-restricted Horn formulas. While very limited in expressive power, it can prove a question from a passage in less than 20ms on average [9]. The prover was optimized by using term indexing, caching, lazy indexing, optimizing literal ordering, and by using profiling tools.

[10] See `http://pi7.fernuni-hagen.de/research/mwrplus`

The E-KRHyper Prover The other system is E-KRHyper [11], a theorem prover for full first order logic with equality, including input which is not Horn and not range restricted. Currently existing alongside the MultiNet prover, E-KRHyper will eventually become the sole reasoning component of LogAnswer once a new translation of MultiNet representations into full first-order logic has been completed. E-KRHyper implements the E-hyper tableau calculus [12]. Designed for use as an embedded knowledge processing engine, the system has been employed in a number of knowledge representation applications. It is capable of handling large sets of uniformly structured input facts, and it can provide proof output for models and refutations. Input is accepted in TPTP syntax [13]. E-KRHyper features several extensions to first-order logic, like arithmetic evaluation, negation as failure and builtin predicates adapted from Prolog. These will be helpful in the ongoing translation of the knowledge base into logic, allowing us to capture the full expressivity of the MultiNet formalism. Compared to other full first-order theorem provers, E-KRHyper also has the pragmatic advantage of being an in-house system, easily tailored to any upcoming difficulties, instead of a black box which we must adapt to.

In the LogAnswer system E-KRHyper is embedded as a reasoning server, and thus it remains in constant operation. On its startup E-KRHyper is supplied with MultiNet background knowledge translated into first-order TPTP syntax. The prover further transforms this into clause normal form, a requirement for the tableaux-based reasoning algorithm of E-KRHyper. Currently this CNF-representation consists of approximately 10,000 clauses. Discrimination-tree indexing serves to maintain this clause set efficiently. Loading the knowledge base into E-KRHyper requires circa four seconds on our test bed system.[11]

Given that an answer to the query may be found in any of the supporting passages (see Section 2), E-KRHyper runs an independent proof attempt for each passage. For such an attempt the query clause (consisting of the negated query literals) and the logical passage representation are added to E-KRHyper's set of clauses. The average query clause for a question from the CLEF-07 contains eight literals, and the average translated passage is a set of 230 facts.

E-KRHyper then tries to find a refutation for the given input. The main LogAnswer system is notified if the prover succeeds. Also, if specific information using a *FOCUS* variable is requested (as described before), then the binding of this variable is retrieved from the refutation and returned to the answer extractor. Finally, E-KRHyper drops all clauses apart from the background knowledge axioms and is ready for the next query or passage.

If on the other hand E-KRHyper fails to find a refutation within the time limit, then it halts the derivation and provides *relaxation loop support* by delivering partial results. These represent the partly successful refutation attempts made so far: during the derivation E-KRHyper evaluates the query clause from left to right, trying to unify all literals with complementary unit clauses from the current tableau branch and thereby yielding a refutation. If a literal cannot be unified, the remaining unevaluated query literals are not considered and

[11] Intel Q6600 2.4 GHz

this attempt stops. Each partial result represents such a failed evaluation; it consists of the subset of refuted query literals, the unifying substitutions and the failed query literal. LogAnswer selects one of the 'best' partial results (i.e. where most query literals could be refuted) and removes the failed literal from the query. E-KRHyper resets its clause set to the background knowledge axioms, and the relaxation loop restarts the derivation with the shortened query. This process is repeated, with another query literal being skipped in each round, until E-KRHyper derives a refutation for the current query fragment or the bound for the number of skipped query literals is reached, see Section 2.[12]

Addressing the *handling of large knowledge bases*, the methods described above reset the clause input before every new task. This is facilitated by the prover's ability to save and restore states of its knowledge base. That way the prover can rapidly drop obsolete subsets of the clauses and their discrimination-trees, with no need to rebuild the extensive index for the background axioms.

To estimate the performance of the prover for our intended use we tested E-KRHyper without relaxation on 1806 query/passage-combinations from CLEF-07 which are known to contain answers. 77 (4.3%) of these could not be proven within a 360 seconds time limit set for each test case, in part due to the yet incomplete logical translation of the MultiNet background knowledge. In the remaining cases E-KRHyper required on average 1.97 seconds for each proof. 895 proofs (49.6%) could be found in less than one second, and the maximum time needed was 37 seconds. This illustrates that relaxation, and the extraction of answer bindings from incomplete proofs, are imperative when processing time is critical (as in ad-hoc question answering on the web), and multiple passages must be processed for a single query in a very short time frame. However, it also shows that more precise answers can be found within a time frame which may still be acceptable for specific applications where time is less important than quality, so our approach is easily scaled to different uses.

4 Conclusions and Future Work

We have presented a logic-based question answering system which combines an optimized deductive subsystem with shallow techniques by machine learning. The prototype of the LogAnswer system, which can be tested on the web, demonstrates that the response times achieved in this way are suitable for adhoc querying. The quality of passage reranking has been measured for factual questions from CLEF-07: On the retrieved passages, the ML classifier, which combines deep and shallow features, obtains a filtering precision of 54.8% and recall of 44.8% [10]. In the future, the potential of E-KRHyper will be exploited

[12] Continuing our example from Section 2, this is one of the candidate passages that will be found and analysed: *(...) the sinking of the ferry 'Estonia' (...) cost the most human lives: over 900 people died (...)* . After three relaxation steps a proof is found with the value *900* bound to the *FOCUS* variable, and LogAnswer returns the answer *over 900 people* to the user.

by formalizing more expressive axioms that utilize equality and non-Horn formulas. The capability of LogAnswer to find exact answers (rather than support passages) will be assessed in the CLEF-08 evaluation.

References

1. Bos, J.: Doris 2001: Underspecification, resolution and inference for discourse representation structures. In: ICoS-3 - Inference in Compuational Semantics, Workshop Proceedings. (2001)
2. Moldovan, D., Bowden, M., Tatu, M.: A temporally-enhanced PowerAnswer in TREC 2006. In: Proc. of TREC-2006, Gaithersburg, MD (2006)
3. Saias, J., Quaresma, P.: The Senso question answering approach to Portuguese QA@CLEF-2007. In: Working Notes for the CLEF 2007 Workshop, Budapest, Hungary (2007)
4. Tatu, M., Iles, B., Moldovan, D.: Automatic answer validation using COGEX. In: Working Notes for the CLEF 2006 Workshop, Alicante, Spain (2006)
5. Glöckner, I.: University of Hagen at QA@CLEF 2007: Answer validation exercise. In: Working Notes for the CLEF 2007 Workshop, Budapest (2007)
6. Hartrumpf, S.: Hybrid Disambiguation in Natural Language Analysis. Der Andere Verlag, Osnabrück, Germany (2003)
7. Helbig, H.: Knowledge Representation and the Semantics of Natural Language. Springer (2006)
8. Leveling, J.: IRSAW – towards semantic annotation of documents for question answering. In: CNI Spring 2007 Task Force Meeting, Phoenix, Arizona (2007)
9. Glöckner, I.: Towards logic-based question answering under time constraints. In: Proc. of ICAIA-08, Hong Kong (2008) 13–18
10. Glöckner, I., Pelzer, B.: Exploring robustness enhancements for logic-based passage filtering. In: KES2008, Proceedings, to appear. Lecture Notes in Computer Science, Springer (2008)
11. Pelzer, B., Wernhard, C.: System Description: E-KRHyper. In: Automated Deduction - CADE-21, Proceedings. (2007) 508–513
12. Baumgartner, P., Furbach, U., Pelzer, B.: Hyper Tableaux with Equality. In: Automated Deduction - CADE-21, Proceedings. (2007) 492–507
13. Sutcliffe, G., Suttner, C.: The TPTP Problem Library: CNF Release v1.2.1. Journal of Automated Reasoning **21**(2) (1998) 177–203