Exploring Robustness Enhancements for Logic-Based Passage Filtering

Ingo Glöckner[1] and Björn Pelzer[2]

[1] Intelligent Information and Communication Systems Group (IICS),
University of Hagen, 59084 Hagen, Germany
ingo.gloeckner@fernuni-hagen.de
[2] Department of Computer Science, Artificial Intelligence Research Group
University of Koblenz-Landau, Universitätsstr. 1, 56070 Koblenz
bpelzer@uni-koblenz.de

Abstract. The use of logic in question answering (QA) promises better accuracy of results, better utilization of the document collection, and a straightforward solution for integrating background knowledge. However, the brittleness of the logical approach still hinders its breakthrough into applications. Several proposals exist for making logic-based QA more robust against erroneous results of linguistic analysis and against gaps in the background knowledge: Extracting useful information from failed proofs, embedding the prover in a relaxation loop, and fusion of logic-based and shallow features using machine learning (ML). In the paper, we explore the effectiveness of these techniques for logic-based passage filtering in the LogAnswer question answering system. An evaluation on factual question of QA@CLEF07 reveals a precision of 54.8% and recall of 44.9% when relaxation results for two distinct provers are combined.[1]

1 Introduction

The necessity to incorporate reasoning capabilities into QA systems was recognized as early as 2000 in the NIST roadmap for question answering [1]. Today, there is also experimental evidence for the claim that question answering can profit from the use of logical subsystems: A comparison of answer validators in the Answer Validation Exercise 2006 found that systems reported to use logic generally outperformed those without logical reasoning [2]. There is also evidence from the related task of Recognizing Textual Entailment (RTE) that a high accuracy of entailment recognition can only be achieved by structure-sensitive methods like logic or graph matching [3]. And the best system in the TREC 2006 evaluation of QA systems, PowerAnswer [4], made use of a theorem prover. However, the logical approach suffers from brittleness: A proof of the question from a document of interest and the background knowledge succeeds only if the question and document are analyzed correctly and if every piece of knowledge required to prove the question is actually available in the knowledge base.

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

I. Lovrek, R.J. Howlett, and L.C. Jain (Eds.): KES 2008, Part I, LNAI 5177, pp. 606–614, 2008.

Therefore the successful applications of logic mentioned above resort to robust methods for logic-based knowledge processing, which show a graceful degradation of results when there are errors of linguistic analysis or knowledge gaps. In this paper, we are interested in robustness-enhancing techniques which can be added to existing theorem provers with minor internal changes. Therefore we do not try to build an approximate inference engine by adopting an approximate graph matcher like [5] and adding support for logical rules – this would almost amount to building a prover from scratch. More suitable solutions are the extraction of useful information from failed proofs [6], combining logic-based and shallow features using machine learning [7,8], and finally relaxation techniques [4,6] which reduce the query by subsequently dropping literals until a proof of the simplified query succeeds. But relaxation does not necessarily find the largest provable query fragment, since it only inspects a single sequence of simplification steps. Moreover the choice of skipped literals usually depends on factors like internal literal order of the prover which are arbitrary to some degree. We therefore propose to abstract from such idiosyncratic aspects by combining relaxation results of two different provers. The effectiveness of the relaxation approach is explored in a logical passage filtering task. We also study the effectiveness of robustness-enhancing techniques like extracting useful information from failed proofs and combining logical features with shallow ones by machine learning.

The remainder of the paper is organized as follows. Section 2 introduces the LogAnswer QA system which provides the testbed for the experiments. Section 3 describes the two provers used by LogAnswer and discusses some possible combinations of the relaxation results of both provers. Section 4 presents the results of passage filtering experiments for several robustness-enhancing techniques. The main results of the paper are summarized in Sect. 5.

2 Overview of the LogAnswer System

This section sketches the architecture of the LogAnswer QA system which forms the basis for the experiment described in the paper. The main innovation of LogAnswer is its use of logic for filtering retrieved passages and for answer extraction. By avoiding the inefficiency of answer validation, LogAnswer answers questions in only a few seconds [8]. The system comprises the following processing stages.

Question Analysis. WOCADI [9], a robust parser for German, is used for a deep linguistic analysis of the given question. The syntactic-semantic analysis results in a semantic representation expressed in the MultiNet formalism [10].

Passage Retrieval. The IRSAW QA framework [11] is used for finding passages with a standard IR approach. Using WOCADI, all texts are analyzed prior to indexing, so that no documents must ever be parsed at query time.

Query Construction. The semantic network for the question is finally turned into a conjunctive list of query literals. This step involves a synonym normalization by

replacing all lexical concepts with canonical synset representatives. For example, the question *Wie hieß der Sänger von Nirvana?*[2] translates into the logical query

$$\text{val}(X_1, \textit{nirvana.0}), \text{sub}(X_1, \textit{name.1.1}), \text{attr}(X_2, X_1), \text{attch}(X_2, X_3),$$
$$\text{sub}(X_3, \textit{gesangsolist.1.1}), \text{subs}(X_4, \textit{heißen.1.1}), \text{arg1}(X_4, X_3), \text{arg2}(X_4, \mathit{FOCUS})$$

based on the lexical concepts (word senses) *nirvana.0* (Nirvana), *name.1.1* (name), *gesangssolist.1.1* (singer soloist), and *heißen.1.1* (be named). Here synonym normalization has replaced *sänger.1.1* (singer) with the canonical *gesangssolist.1.1* (singer soloist). The $FOCUS$ variable represents the queried information.

Robust Entailment Test. The basic idea of the logic-based passage filtering is that the passage contains a correct answer to the question if there is a proof of the question from the passage representation and the background knowledge.[3] A relaxation loop is used to gain more robustness. Suitable provers must be able to return the proven portion of a query in the case of a failed proof, and also identify the literal which caused a complete proof to fail. The relaxation process then skips the failed literal and tries to prove the resulting query fragment. The process can be repeated until a proof of the remaining query succeeds, and the skip count is a useful feature for recognizing (non-)entailment. Consider the question for the Nirvana lead singer, and this passage (translated from German): *Fans and friends of the lead singer of US-American rock band Nirvana reacted with grief and dismay to the suicide of Kurt Cobain this weekend.* Here the parser misses a coreference and produces two distinct entities for Cobain and the Nirvana lead singer. Thus the system finds only a relaxation proof with skipped literals $\text{sub}(X_3, \textit{gesangsolist.1.1})$, $\text{attch}(X_2, X_3)$ and the $FOCUS$ variable bound to Kurt Cobain. In practice, relaxation is stopped before all literals are proved or skipped. One can then only state upper/lower bounds on the provable literal count, assuming that all (or none) of the remaining literals are provable.

Feature Extraction. The classification of passages rests on the following logic-based features which depend on the chosen limit on relaxation cycles:

- *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. $1 - \textit{skippedLitsUb}/\textit{allLits}$.
- *litRatioUb* Relative proportion of potentially provable literals (not yet skipped) vs. all query literals, i.e. $1 - \textit{skippedLitsLb}/\textit{allLits}$.
- *boundFocus* Fires if a relaxation proof binds the queried variable.
- *npFocus* Indicates that the queried variable was bound to a constant which corresponds to a nominal phrase (NP) in the text.
- *phraseFocus* Signals that extraction of an answer for the binding of the queried variable was successful.

[2] *What was the name of the singer of Nirvana?*

[3] LogAnswer uses the same background knowledge as the MAVE answer validator [6].

A simplistic solution for answer extraction is used for computing the *phraseFocus* feature. The method leverages information on word alignment, as provided by the parser for nominal phrases (NPs), in order to find answer strings for the bindings of the queried variable. This is done by cutting verbatim text from the original text passage. The basic idea behind the *boundFocus*, *npFocus* and *phraseFocus* features is that the ability of the answer extraction stage to verbalize the answer binding might also have something to say about the relevance of the passage.

In addition to the logic-based features, five 'shallow' features are used which do not require a deep parse and can be computed without the help of the prover:

- *failedMatch* Number of lexical concepts and numerals in the question which cannot be matched with the candidate document.
- *matchRatio* Relative proportion of lexical concepts and numerals in the question which find a match in the candidate document.
- *failedNames* Proper names mentioned in the question, but not in the passage.
- *irScore* Original passage score of the underlying passage retrieval system.
- *containsBrackets* Indicates that the passage contains a pair of parentheses.

The matching technique used to obtain the values of these shallow features also takes into account lexical-semantic relations (e.g. synonyms and nominalizations), see [6]. The *containsBrackets* feature is motivated as follows: Often the queried information is contained in parentheses, e.g. 'Kurt Cobain (Nirvana)', but the linguistic parser has difficulty finding the relationship between the basic entity and the information given in brackets. Thus containment of a pair of brackets in the passage is significant to the relevance judgement.

ML-based Passage Classification. The Weka machine learning toolbench [12] is used for learning the mapping from features of retrieved passages to yes/no decisions concerning containment of a correct answer in the considered passage. The low precision of the original passage retrieval step means a strong disbalance between positive and negative examples in the data sets. In order to emphasize the results of interest (i.e. positive results) and achieve sufficient recall, *cost-sensitive learning* is applied. In the experiments, false positives were weighted by 0.3 while a full weight of 1 was given to lost positives (i.e. false negatives).The Weka Bagging learner with default settings is used as the classifier. It is wrapped in a Weka CostSensitiveClassifier to implement the cost-sensitive learning.

3 Combining Provers for Increased Robustness

This section introduces the two provers used by LogAnswer for logic-based passage filtering and answer extraction. It then discusses how these provers can be combined for improving robustness of knowledge processing.

3.1 The Regular MultiNet Prover

LogAnswer is equipped with a dedicated prover for MultiNet representations, which is part of the MWR+ toolbench.[4] The MultiNet prover is, in principle,

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

a regular prover for Horn logic based on SLD resolution. To be precise, the supported logic is even more restricted since the additional assumption is made that all facts are variable-free and that (after skolemization), all variables which occur in the conclusion of a rule also occur in its premise. On the other hand, the prover offers builtin support for special MultiNet constructions (e.g. efficient access to so-called layer features of conceptual nodes and a builtin subsumption check for MultiNet sorts). The prover also offers special support for rules with complex (conjunctive) conclusions which are useful for modelling natural language-related inferences. The translation into Horn logic splits such rules into several clauses, which is inefficient because typically all literals in the conclusion are needed when the rule is applied. The MultiNet prover solves this problem by keeping track of complex conclusions. When applying such a rule, the complex conclusion is cached as a lemma in order to shortcut proofs of other literals from the derived conclusion. The knowledge base can be split into several partitions which can be flexibly combined or exchanged as needed. Iterative deepening is used to control the search. While very limited in expressive power, the MultiNet prover is extremely fast, and proving a question from a passage usually takes less than 20ms [8]. The prover has been systematically optimized both for speed and scalability by utilizing term indexing, caching, lazy computation (index structures are only built on demand), by optimizing the order in which literals are proved, and by removing performance bottlenecks with the help of profiling tools.

3.2 Description of the E-KRHyper Prover

E-KRHyper is an automated theorem proving and model generation system for first-order logic with equality [13]. It is an implementation of the E-*hyper tableau calculus* [14], which integrates a superposition-based handling of equality into the hyper tableau calculus [15]. E-KRHyper is the latest version in the KRHyper-series of theorem provers, developed at the University Koblenz-Landau. Designed for use as an embedded knowledge-processing engine, it has been employed in a number of knowledge-representation applications. E-KRHyper is capable of handling large sets of uniformly structured input facts. The system can provide proof output for models and refutations, and it is able to rapidly switch and retract input clause sets for an efficient usage as a reasoning server. E-KRHyper accepts input first-order input in the common TPTP syntax [16].

The principal data structure used in the operation of E-KRHyper is the E-*hyper tableau*, a tree labeled with clauses and built up by the application of the inference rules of the E-hyper tableau calculus. The tableau is generated depth-first, with E-KRHyper always working on a single branch. Refutational completeness and a fair search are ensured by iterative deepening with a limit on the maximum term weight of generated clauses.

Embedded in the LogAnswer system, E-KRHyper is supplied with the MultiNet axioms transformed into first-order TPTP syntax. The inference process then operates on the axioms and the negated query literals, with a refutation result indicating a successful answer and providing the binding for the queried

variable. If the reasoning is interrupted due to exceeding the time limit, then partial results can be retrieved that can guide in the relaxation process.

3.3 Methods for Combining Prover Results

Due to the use of two provers, a pair of results is obtained for each logic-based feature. The most basic approach for incorporating these features into the machine learning approach is juxtaposition (JXT), i.e. both results for each feature are kept and directly passed to the ML method. This method leaves the interpretation of the data entirely to machine learning.

Another approach (OPT) rests on the observation that the relaxation method is generally pessimistic, since it does not necessarily find the largest provable query fragment. This suggests an optimistic combination of relaxation features:

$$\begin{aligned} \mathit{skippedLitsLb} &= \min(\mathit{skippedLitsLb}_1, \mathit{skippedLitsLb}_2), \\ \mathit{skippedLitsUb} &= \min(\mathit{skippedLitsUb}_1, \mathit{skippedLitsUb}_2), \\ \mathit{litRatioLb} &= \max(\mathit{litRatioLb}_1, \mathit{litRatioLb}_2), \\ \mathit{litRatioUb} &= \max(\mathit{litRatioUb}_1, \mathit{litRatioUb}_2). \end{aligned}$$

The other logical features are chosen according to the best value of *skippedLitsLb*. Thus, if $\mathit{skippedLitsLb}_1 < \mathit{skippedLitsLb}_2$, then $\mathit{boundFocus} = \mathit{boundFocus}_1$, $\mathit{npFocus} = \mathit{npFocus}_1$ and $\mathit{phraseFocus} = \mathit{phraseFocus}_1$ using the results of the regular MultiNet prover. Otherwise the values of these features are provided by E-KRHyper. Notice that E-KRHyper sees slightly different queries compared to the MultiNet prover, which is due to the transformation of the queries from the MultiNet format into TPTP formulas. A scaling by query length is necessary so that both prover results refer to the same number of query literals.

4 Evaluation

The questions of the CLEF07 QA track for German served as the starting point for the evaluation, since they target at the corpora currently supported by the IRSAW IR module (CLEF News and Wikipedia).[5] From the full set of 200 questions, all follow-up questions were eliminated since discourse processing is not of relevance here. Definition questions were omitted as well since knowing the logical correctness of an answer is not sufficient for deciding if it is suitable as a definition. The remaining 96 factual questions were checked for parsing quality and two questions for which construction of a logical query failed and one outlier question with an unusually high number 37 supporting passages were discarded. For each of the remaining 93 questions in the test set, IRSAW retrieved up to 200 one-sentence snippets from the pre-analyzed corpora, resulting in a total of 18,500 candidate passages. The snippets with an incomplete parse were eliminated since they cannot be handled by logical processing. The 12,377 passages

[5] See `http://www.clef-campaign.org/2007.html`

Table 1. Quality of passage filtering as a function of allowable relaxation steps n. Abbreviations: RMP (regular MultiNet prover), KRH (E-KRHyper), JTX (juxtaposition), OPT (optimistic combination), IRB (information retrieval baseline, using *irScore* only), SHB (shallow baseline, using all shallow features). The 0ℓ runs use strict proofs and logical features. The 0s runs use strict proofs and both logical and shallow features.

model	n	precision	recall	F-score
RMP	0ℓ	0.786	0.043	0.082
RMP	0s	0.46	0.457	0.458
RMP	0	0.471	0.421	0.445
RMP	1	0.458	0.386	0.419
RMP	2	0.502	0.398	0.444
RMP	3	0.505	0.409	0.452
RMP	4	0.453	0.382	0.415
KRH	0ℓ	0.702	0.13	0.219
KRH	0s	0.449	0.449	0.449
KRH	0	0.462	0.429	0.445
KRH	1	0.490	0.390	0.434
KRH	2	0.565	0.378	0.453
KRH	3	0.533	0.386	0.447
KRH	4	0.518	0.402	0.452
IRB	–	0.291	0.098	0.147

model	n	precision	recall	F-score
JXT	0ℓ	0.702	0.13	0.219
JXT	0s	0.443	0.441	0.442
JXT	0	0.513	0.461	0.485
JXT	1	0.464	0.402	0.430
JXT	2	0.518	0.390	0.445
JXT	3	0.488	0.398	0.438
JXT	4	0.463	0.394	0.426
OPT	0ℓ	0.688	0.043	0.081
OPT	0s	0.462	0.449	0.455
OPT	0	0.496	0.441	0.467
OPT	1	0.435	0.394	0.413
OPT	2	0.493	0.413	0.450
OPT	3	0.548	0.449	0.494
OPT	4	0.514	0.433	0.470
SHB	–	0.433	0.421	0.427

with a full parse (133 per query) were annotated for containment of a correct answer to the question, starting from CLEF07 annotations. The annotation revealed that only 254 of the 12,377 passages really contain an answer.

Table 1 shows the precision, recall, and F-measure for the individual provers and for the best combinations that were tried, along with baseline results for exact proofs and for shallow features. These results were obtained by the cost-sensitive learning approach described in Section 2, using 10-fold cross validation. Compared to the IBS run only based on the *irScore* of the passage retrieval system, all regular runs show a dramatic improvement. The shallow processing baseline (SHB) demonstrates the potential of suitably chosen shallow features: Adding the *irScore* and *containsBrackets* features now increased the F-score of the SHB to 42.7%, compared to 36% in earlier work on the same dataset [8]. The brittleness of the logical approach reflects in very poor retrieval scores when using only logic-based features and exact proofs, see runs labeled '0ℓ'. However, as shown by the '0s' runs, combining the logical features based on exact proofs with the shallow features through ML eliminates the brittleness. The results obtained in this way (in particular the RMP-0s run with an F-score of 45.8%) also clearly outperform the shallow feature baseline. As witnessed by the RMP-n and KRH-n results for $n \in \{0, \ldots, 4\}$, the methods for extracting plausible answer bindings for failed proofs and relaxation show no clear positive effect compared to combining exact proofs with shallow features when a single prover is used. An improvement only occurs when results of two provers are combined. The JXT method which juxtaposes the features determined by the two provers shows good filtering results for $n = 0$ relaxation steps (with an F-score of 48.5%)

but does not appear to work very well otherwise. The OPT method shows more stable performance. For $n = 3$, it achieves the best result in this experiment, with a relative improvement of 7.9% over the best F-score for a single prover.

5 Conclusions

We have discussed robustness-enhancing techniques developed for logical passage filtering in the LogAnswer QA system. The evaluation on factual questions of CLEF07 revealed that combining logic with shallow features using an ML classifier is the most effective technique for increasing robustness. For relaxation and extracting answer bindings for failed proofs, an improvement over the use of shallow features was only achieved when results of two provers were combined.

References

1. Burger, J., Cardie, C., Chaudhri, V., Gaizauskas, R., Harabagiu, S., Israel, D., Jacquemin, C., Lin, C., Maiorano, S., Miller, G., Moldovan, D., Ogden, B., Prager, J., Riloff, E., Singhal, A., Shrihari, R., Strzalkowski, T., Voorhees, E., Weishedel, R.: Issues, tasks, and program structures to roadmap research in question & answering (Q&A). NIST (2000)
2. Peñas, A., Rodrigo, Á., Sama, V., Verdejo, F.: Overview of the answer validation exercise 2006. In: Working Notes for the CLEF 2006 Workshop (2006)
3. Giampiccolo, D., Magnini, B., Dagan, I., Dolan, B.: The third PASCAL recognizing textual entailment challenge. In: Proc. of the Workshop on Textual Entailment and Paraphrasing, Prague, June 2007, pp. 1–9. ACL (2007)
4. Moldovan, D., Bowden, M., Tatu, M.: A temporally-enhanced PowerAnswer in TREC 2006. In: Proc. of TREC-2006, Gaithersburg, MD (2006)
5. Haghighi, A.D., Ng, A.Y., Manning, C.D.: Robust textual inference via graph matching. In: Proc. of HLT/EMNLP 2005, Vancouver, BC, pp. 387–394 (2005)
6. Glöckner, I.: University of Hagen at QA@CLEF 2007: Answer validation exercise. In: Working Notes for the CLEF 2007 Workshop, Budapest (2007)
7. Bos, J., Markert, K.: When logical inference helps determining textual entailment (and when it doesn't). In: Proc. of 2nd PASCAL RTE Challenge Workshop (2006)
8. Glöckner, I.: Towards Logic-Based Question Answering under Time Constraints. In: Proc. of ICAIA 2008, Hong Kong, pp. 13–18 (2008)
9. Hartrumpf, S.: Hybrid Disambiguation in Natural Language Analysis. Der Andere Verlag, Osnabrück (2003)
10. Helbig, H.: Knowledge Representation and the Semantics of Natural Language. Springer, Heidelberg (2006)
11. Leveling, J.: IRSAW – towards semantic annotation of documents for question answering. In: CNI Spring 2007 Task Force Meeting, Phoenix, Arizona (2007)
12. Witten, I.H., Frank, E.: Data Mining. Practical Machine Learning Tools and Techniques. Morgan Kaufmann, San Francisco (2005)
13. Pelzer, B., Wernhard, C.: System Description: E-KRHyper. In: Automated Deduction - CADE-21, Proceedings, pp. 508–513 (2007)

14. Baumgartner, P., Furbach, U., Pelzer, B.: Hyper Tableaux with Equality. In: Automated Deduction - CADE-21, Proceedings (2007)
15. Baumgartner, P., Furbach, U., Niemelä, I.: Hyper Tableaux. In: Orłowska, E., Alferes, J.J., Moniz Pereira, L. (eds.) JELIA 1996. LNCS, vol. 1126, pp. 1–17. Springer, Heidelberg (1996)
16. Sutcliffe, G., Suttner, C.: The TPTP Problem Library: CNF Release v1.2.1. Journal of Automated Reasoning 21(2), 177–203 (1998)