DalTREC 2006 QA System Jellyfish: Regular Expressions Mark-and-Match Approach to Question Answering

Vlado Kešelj Tony Abou-Assaleh
Nick Cercone
Faculty of Computer Science
Dalhousie University, Halifax, Canada
{vlado,taa,nick}@cs.dal.ca

24 October 2006

Abstract

We present a question-answering system Jellyfish. Our approach is based on marking and matching steps that are implemented using the methodology of cascaded regular-expression rewriting. We present the system architecture and evaluate the system using the TREC 2004, 2005, and 2006 datasets. TREC 2004 was used as a training dataset, while TREC 2005 and TREC 2006 were used as testing dataset. The robustness of our approach is demonstrated in the results.

1 Introduction

Previous work has explored the application of several approaches to question answering in the overlapping area of unification-based and stochastic NLP (Natural Language Processing) techniques [5, 1]. Two novel methods that were explored relied on the notions of *modularity* and *just-in-time sub-grammar extraction.*

One of the learned lessons of the previous experiments is that the regular expression (RegExp) substitutions are a very succinct, efficient, maintainable, and scalable method to model many NL subtasks of the QA task. This is also observed in the context of lexical source-code transformations of arbitrary programming languages [2], where RegExp substitutions are an alternative to manipulating the abstract syntax tree, and proved to be more robust in the face of missing header files, errors, usage of macros, templates, and other embedded programming language constructs.

We employ RegExp rewriting as a primary technique in our question-answering (QA) system Jellyfish. We use RegExpmatching and rewriting at various stages of the system, whose architecture is described in section 3.

We evaluate our system using the datasets from TREC 2004, TREC 2005, and TREC 2006. Our system is developed with respect to the TREC 2004 dataset, which acts as a training dataset. The TREC 2005 and TREC 2006 datasets are used for testing.

Our goal is to apply a unification-based approach as a high-level answer-extraction step on top of the low-level RegExp processing.

2 Regular Expression Rewriting

The basic method used at various components of the QA system is *RegExp rewriting.* The open angle bracket (<) is used as a special escape character, hence we make sure that it does not appear in the source text, which is either a question or a passage. The basic text substrings, such as the target or named entities, are recognized using regular expressions and replaced with an angle-bracket-delimited expression. For example, the target is marked as <TARGET>. More commonly, a named entity e of type t is replaced with <t_e_s>, where e_s is the named entity e encoded as a string of printable characters that do not include <. The RegExp rewriting can be seen as a bottom-up deterministic parsing technique. For example, the rewriting in which "<NP_x> <VP_y>" is replaced with "<S_z>" corresponds to the context-free rule $S \rightarrow NP\ VP$. The value z is obtained by decoding x and y, concatenating them, and encoding the result again.

3 System Architecture

The current system consists of the following phases: 1) Question Processing, 2) Passage Retrieval, 3) Target Marking, 4) Question Category Marking, 5) Matching, 6) Answer selection, and 7) Post-processing.

3.1 Question Processing

The Question Processing phase takes the original questions as input, parses them, and generates complete questions as output. Questions are parsed using RegExp matching and substitution to identify the question category and extract some related metadata. Some metadata extracted during parsing is analyzed using WordNet [3] to identify additional metadata and relationships between the target and what the question is asking about.

In TREC datasets, question are grouped by targets. Replacing the anaphoric references in questions with the target results in self-contained questions. These *complete* questions are used in passage retrieval.

3.2 Passage Retrieval

The passage retrieval from the AQUAINT dataset is performed by an external search engine using the full questions generated in the question processing

phase. We use two different sets of passages. The first set is the passages provided by NIST using the PRISE search engine. The second set is generate using our MySQL-based search engine that employed MySQL's full-text indexing functions. In both cases, the PRISE and MySQL, the results are treated in the same way—as passages relevant to the question.

3.3 Target Marking

The string in the question that constitutes that target is identified during the question processing step (section 3.1). Using simple RegExp rewriting rules, the target is identified in the passages and replaced with the `<TARGET>` tag. In effect, this phase annotates sentences in the passages that may contain an answer. Currently, our system does not handle intra-sentence references.

3.4 Question Category Marking

During this step, the system scans all the relevant passages and uses RegExp rewriting to mark entities that belong to the question category. The type of RegExp used depends on the question category and may be a simple keyword-based RegExp or a sophisticated multipart RegExp. Question category marking acts as a just-in-time annotation phase where only the passages that may be relevant to the current question are annotated, and the annotation data is customized based on the question category.

3.5 Answer Matching

During answer matching, question metadata and annotated passages are combined using special RegExp rules to generate candidate matches. The rules are applied sequentially. Every time a match is found, it is added to the list of matches. The realtive order of the rules imposes an implicit ranking of matches. Consequently, more specific rules are placed before the more general ones. Since some questions may have no answers in the dataset, one must avoid using rules that are too general. Appropriate rule generality and ordering depends on the question category. Typically, more specific question categories permit the usage of more general rules, and vice versa.

3.6 Answer Selection

This phase is a filtering step that takes the list of answer for each question from the matching phase and selects the answers that are to be included in the output. For TREC, we set the number of answers for factoid questions to 1 and for list questions to 7. Presently, we simply select the first answers in the list.

3.7 Post-processing

This phase formats the output either for evaluation, inspection, or integration in other systems.

4 Evaluation

The TREC 2004 QA dataset was used for deriving the rules and fine tuning the system. Testing of the system used TREC 2005 and TREC 2006 QA datasets. On the training data, our system was able to correctly answer upto 46 out of 230 facoitd questions yeilding an accuracy of 20%. In the TREC 2005 dataset, the number of factoid questions was increased to 364, and included new types of questions dealing with events. Our system answered correctly 41 questions on the testing dataset yielding an accuracy of 11.3%. In the TREC 2006 dataset, the number of factoid question was 403. The formulation of the questions in TREC 2006 relied more heavily on expanding the questions to incorporate the context information provided in the target of each series. Our system answered 36 questions globally correctly, and 2 locally correctly, yielding global correctness accuracy of 8.9%. The results of evaluation on all 3 datasets are presented in table 1, where the number in the *Correct column* reflects the globally correct answers and in paranthesis the locally correct answer; the *Pyramid* column shows a new alternative method for evaluating the "Other" questions. The suffix in the run name is interpreted as follows[1]:m means the MySQL-based search engine is used, p means the top 50 results from the PRISE search engine are used, and e means the top 100 results from PRISE are used.

Run	Correct	Factual	List	Other	Pyramid	Session
Dal06e	36(2)	0.0 89	0.021	0.028	0.032	0.046
Dal06p	33(2)	0.082	0.019	0.033	0.038	0.045
Dal06m	20(1)	0.050	0.010	0.037	0.033	0.033
Dal05p	41	0.113	0.033	0.088	-	0.087
Dal05m	27	0.075	0.017	0.056	-	0.056
Dal04p	46	0.200	0.092	-	-	0.164
Dal04m	41	0.178	0.083	-	-	0.146
Best 06	-	57.8	0.433	0.250	0.251	0.394
Median 06	-	18.6	0.087	0.125	0.139	0.134
Worst 06	-	4.0	0.000	0.000	0.000	0.013

Table 1: Jellyfish results from TREC QA Track

5 Lessons Learned

RegExp rewriting is a simple and powerful parsing technique. We have effectively used coarse-grained modularization of RegExp, and combined it with dynamic loading of RegExp. Fine grained modularization of RegExp is possible, and would simplify the task of creating RegExp rules. Using a macro system, such as Starfish, can greatly reduce the complexity of RegExp rules. Just-in-time RegExp-based annotation can lower the computation requirements

[1]The labels of runs for previous years may differ from the actual submitted labels

of deeper analysis. Our system is fairly robust. There performance of Jellyfish on TREC 2005 and TREC 2006 questions is comparably. There is a slight decline in the accuracy on factoid question, a significant improvement in the list questions score, and a noticeable decline in the "other" questions score. We note that the core rules of the system are almost identical between for our 2005 and 2006 submissions, subjected only to minor bug fixes.

References

[1] N. Cercone, L. Hou, V. Kešelj, A. An, K. Naruedomkul, and X. Hu. From computational intelligence to web intelligence. *IEEE Computer*, 35(11):72–76, November 2002.

[2] A. Cox, T. Abou-Assaleh, W. Ai, and V. Kešelj. Lexical source-code transformation. In *Proceedings of the STS'04 Workshop at GPCE/OOPSLA*, Vancouver, Canada, October 2004.

[3] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[4] V. Kešelj. Question answering using unification-based grammar. In E. Stroulia and S. Matwin, editors, *Advances in Artificial Intelligence, AI 2001*, volume LNAI 2056 of *L.N. in Comp.Sci., Springer*, pages 297–306, Ottawa, 2001.

[5] V. Kešelj. Modular stochastic HPSGs for question answering. Technical Report CS-2002-28, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, June 2002.