Natural Language Engineering **12** (4): 305–334. © 2005 Cambridge University Press 305
doi:10.1017/S1351324905004018 First published online 6 December 2005 Printed in the United Kingdom

Reading comprehension tests for computer-based understanding evaluation

BEN WELLNER, LISA FERRO,
WARREN GREIFF and LYNETTE HIRSCHMAN
The MITRE Corporation, 202 Burlington Road, Bedford. MA 01730, USA
e-mail: wellner@mitre.org

(*Received 7 August 2003; revised 4 February 2005*)

Abstract

Reading comprehension (RC) tests involve reading a short passage of text and answering a series of questions pertaining to that text. We present a methodology for evaluation of the application of modern natural language technologies to the task of responding to RC tests. Our work is based on ABCs (Abduction Based Comprehension system), an automated system for taking tests requiring short answer phrases as responses. A central goal of ABCs is to serve as a testbed for understanding the role that various linguistic components play in responding to reading comprehension questions. The heart of ABCs is an abductive inference engine that provides three key capabilities: (1) *first-order logical representation* of relations between entities and events in the text and rules to perform inference over such relations, (2) *graceful degradation* due to the inclusion of abduction in the reasoning engine, which avoids the brittleness that can be problematic in knowledge representation and reasoning systems and (3) *system transparency* such that the types of abductive inferences made over an entire corpus provide cues as to where the system is performing poorly and indications as to where existing knowledge is inaccurate or new knowledge is required. ABCs, with certain sub-components not yet automated, finds the correct answer phrase nearly 35 percent of the time using a strict evaluation metric and 45 percent of the time using a looser inexact metric on held out evaluation data. Performance varied for the different question types, ranging from over 50 percent on *who* questions to over 10 percent on *what* questions. We present analysis of the roles of individual components and analysis of the impact of various characteristics of the abductive proof procedure on overall system performance.

1 Introduction

This article presents the results of the latest efforts in a research program that uses reading comprehension tests as the basis for the development and evaluation of natural language processing (NLP) technology. The primary motivation for undertaking this research is our belief that the reading comprehension (RC) task can serve as an important driver of fundamental NLP research. On one level, accurately answering RC questions will require improvements in component NLP technologies such as dependency parsing, coreference, and word sense. But at a deeper level we believe that the RC task will promote the development of technology that might be said to "understand" the text it is processing. Reading comprehension tests,

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press

Climbers Reach Top of the World

(NEPAL, 1953) – Two mountain climbers have reached the top of the world's tallest mountain. When they got to the top of Mount Everest, the two men had climbed 29,028 feet.

The men were helped by a team of climbers. First, they set up a base camp. This is where supplies are kept. Other camps were made higher up the mountain. Here, the climbers could get away from the wind. At times, snowstorms made it hard to climb. So they stayed in the camps until the weather got better.

When the storms cleared, they moved on. Edmund Hillary was the best climber. He took along his guide, named Tenzing.

After hours of fighting the snow and ice, they reached the top, called the summit. Eight teams of climbers tried and failed before them. But Hillary and Tenzing proved it can be done!

1. *Who reached the top of Mount Everest?*
2. *What is another name for a mountain top?*
3. *When did the climbers reach the top?*
4. *Where is Mount Everest?*
5. *Why do you think it is so hard to reach the top?*

Copyright 1989 Remedia Publications

Fig. 1. Remedia story.

such as the one illustrated in Figure 1, are routinely used to assess the degree to which people comprehend what they read. It is then reasonable to use these same tests to assess the degree to which a machine "comprehends" what it is reading.

There has been a long history of related research. As far back as the early 1960s, programs were developed to answer questions posed in natural language. Systems such as BASEBALL (Green, Wolf, Chomsky and Laughery 1986) and LADDER (Hendrix, Sacerdoti, Sangalowicz and Slocum 1986) were designed to transform questions expressed in human language into formal queries which could be executed against a structured database. Work of this nature was actively pursued through the 1970s (Grosz, Jones and Webber 1986; Webber 1986). By the end of the decade, researchers began to pay greater attention to the problem of answering questions based on the comprehension of stories. In contrast to situations in which the knowledge needed to answer the questions was encoded in a formal structure in the machine, the computer program was now expected to interpret human-created text. In particular, work at Yale focused on the development of a framework for story understanding. This resulted in the use of scripts and plans based on the representational theory of conceptual dependencies. Wendy Lehnert, in particular, studied the task of answering questions based on a story, and focused on pragmatic issues and the importance of the context of the story in responding to questions (Lehnert 1986).

In the late 1990s a research program in NLP was begun that utilizes standardized RC tests written for human students. Under this program, the Deep Read system (Hirschman, Light, Breck and Burger 1999) was designed as a baseline experiment to test what was possible with the standard stock of component technology available at that time. The system decomposed the story sentences and

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press

questions into unordered bags of words (stemmed, and with stop-words removed), recognized low-level semantic information in the form of named entities, and identified the answer type of each question (e.g. whether the answer is looking for a date or a person). To answer an RC question, the system would return a story sentence that had the greatest word overlap with the question. On *who*, *what*, *when*, *where*, and *why* questions, this simple system got 34% of the answer sentences correct. Following this, the Workshop on Reading Comprehension Tests as Evaluation for Computer-based Understanding Systems was held at ANLP/NAACL 2000. Participants used the same corpus and evaluation methodology (identifying the correct sentence containing the answer) as the Deep Read system.

Enhancements to the core technology of Deep Read and the analysis of additional corpora revealed critical gaps in system capabilities. Light, Mann, Riloff and Breck (2001) showed that even if an RC system were provided with the correct answer type and the sentence containing the answer, there is still a significant margin for error because sentences often contain several items of the same type. On the corpora they examined, an ideal system under these experimental conditions would only achieve an accuracy of around 60 percent. A reasonable conclusion one can draw from these results is that improved performance will only be possible if the system recognizes the grammatical relations between sentence constituents, thus allowing the system to identify, on principled grounds, the one that corresponds to the correct answer.

Meanwhile, rapid progress was being made in the field of question answering in general, primarily due to the inclusion of a *Question Answering* (QA) track in the annual TREC (Text REtrieval Conference) meetings. The most significant advances were made by those who incorporated symbolic reasoning and did not rely solely on statistical processing of surface features of the text. In TREC 2002, the best performing system was able to answer 415 (83%) of the 500 questions correctly (Moldovan, Harabahiu, Girju, Morarescu, Lacatusu, Novischi, Badulescu and Bolohan 2002). To search for the best response to a question, this system employs a theorem prover operating on an internal representation of the knowledge embodied in a text sample.

While question answering is an important component of the RC task, it differs from the TREC-style question answering task. In the RC task there is no information retrieval component and the answer to a given question must be found in the accompanying RC text document. This requires that the system cope with the form of the information as it is phrased in the given document. In particular, the system is unable to exploit the redundancy of an answer appearing in multiple documents and perhaps appearing in an easily accessible form in a subset of those documents. As Light *et al.* (2001) show, the TREC-8 systems' performance for a particular question was highly correlated with the number of occurences of the answer in the corpus. As a research methodology, the focus on extracting answers from a specified document also provides opportunities for characterization of the comprehension task and analysis of the automated system. Because the answer to a question must come from the given document, aspects of language usage and story structure can be examined in the context of responding to test questions; the need for component

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press

technologies can be identified; and the relative importance of the various component technologies to system performance on comprehension exams can be analyzed. RC questions have a wide spectrum of difficulty from simple factoid questions which constitute the bulk of TREC-style questions to more complex questions in which information found in different sentences must be pieced together. For example, in Figure 1, the answer to the first question requires co-reference information across sentences – connecting "Two mountain climbers" with "Hillary and Tenzing".

Within this context, our next phase of research under the RC program has sought to incorporate grammatical representation and reasoning into the system architecture. These are the primary features of our current system, ABCs (Abduction Based Comprehension system). The methodology we used to evaluate ABCs was further motivated by the desire to decompose the problem of responding to RC test questions and to analyze the problem of automating the RC task. To this end, we have continued the approach developed in Hirschman *et al.* (1999) based on extensive manual annotation of corpora and analysis of the role of individual components and their effect on system performance. In this paper we report on the performance of an initial version of ABCs on an RC corpus of published remedial reading materials, and we present results of experiments run to analyze the characteristics of the system, the corpus and their interaction.

The rest of this paper is organized as follows. In section 2 we discuss the corpus and evaluation methodology used for this study. Section 3 presents ABCs' system architecture and overall approach to computerized reading comprehension test taking as well as overall performance results and analysis. Section 4 describes ABCs' components in greater detail and presents evaluation results and analyses at the component level. Section 5 provides detailed analyses of the abductive inferences made by the system as well as a look at alternative solutions for a given question. Section 6 concludes the article and highlights ongoing and future work.

2 Corpus and evaluation methodology

For our development and evaluation data, we adopted the Remedia corpus used previously (Hirschman *et al.* 1999; Charniak, *et al.* 2000; Riloff 2000; Wang 2002). Figure 1 contains a sample story from this corpus, which consists of teaching materials for grade school children purchased in hard copy form from Remedia Publications. Called *The 5 W's*, the collection is divided into four sets of roughly 30 stories, with each set targeted to a different reading grade level, beginning at level 2 and ending with level 5 (the ages range from 7–11). Accompanying each story are five short questions: who, what, when, where, and why. We did not include the *why* questions in the experiments described in this paper.[1]

[1] We did not include *why* questions for several reasons: they are often answered with an entire clause or sentence or fragments across multiple sentences; and they also tend to require significant discourse processing. These issues require capabilities beyond the scope of our current system and our grading infrastructure, which is geared to concise answer phrases.

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press

Each story is approximately 150–200 words in length and is roughly modeled after news stories found in the human interest and cultural sections of newspapers. Like a newspaper, each story has a headline with a location and date leading the story. Each story typically has a particular event or individual as its focus, and many stories also contain background and historical information to illuminate the importance and relevance of the topic. The stories cover a wide time period, ranging mostly across the 19th and 20th centuries, and include a wide range of "current event" topics including science, natural disasters, economy, sports, and the environment. The questions themselves are simple "factoid" questions that are actually atypical of what most education experts consider ideal for stimulating the interest of young human readers. Except for *why* questions, which are not being considered in this study, the questions do not typically ask students to express an opinion or make a prediction based on what they have read in the story. While this makes for less exciting reading material for children, it makes the corpus more suitable for our purposes.

Because the stories are written for young children, the sentences are rather short, ranging from an approximate average of eight words per sentence for Level 2 to 10 words per sentence in Level 5.[2] In many ways, for today's NLP systems, this "elementary" reading material can be more challenging than adult material. The shorter the sentences, the more propositions about entities are scattered over several sentences rather than being contained in a single sentence. Although adequate sentence parsers are readily available, little technology exists for extracting inter-sentential information in open domain text. Thus, locating the basic "who, what, when, where, why" information about a topic becomes more difficult.

2.1 Evaluation methodology

In this section we describe the methodology and metrics used to assess the accuracy of the system's responses to the reading comprehension questions. We wished to manually evaluate the system's answers, even during the system development phase, but this required a methodology that would allow us to do the evaluation in an efficient and consistent manner. The process adopted made use of the following three components:

- An answer key
- Grading guidelines
- Grading software

For the purpose of semi-automatic evaluation of our computer system, the teacher's answer key that came with the Remedia stories for children was insufficient for two reasons. First, the original answer key often paraphrased strings from the text rather than rendering them word-for-word. Because our system is currently only capable of extracting answer strings from the story text, it could never hope

[2] We compared this to 10 randomly selected Associated Press newswire stories; the average number of words per sentence in those stories is 21.

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press

to match the human-generated paraphrases. Second, the original answer key only contained one possible wording per question, whereas there are often multiple possible wordings available from the text, such as "Mr. and Mrs. Fischer" vs. "The Fischers".

To remedy these shortcomings, we had a person who was not a developer of the system read each story and enrich the answer key as appropriate, adding multiple possible answer strings. Although the resulting answer key uses phrases from the story as much as possible, it still only contains felicitous answers such as those expected from a human test-taker. For example, in response to the question, *What did Lynne do to get ready for her swim?*, one expects a clause or verb as an answer. If the semantically closest answer available in the text were instead a noun phrase such as *the swim that she took each day in ice cold water (helped to prepare Lynne for her swim)*, it would not be added to the answer key as a possible correct answer. (In such cases, our system is currently incapable of providing a felicitous answer, since it cannot generate a verb phrase from a noun phrase.)

We considered the enriched answer key to be our baseline answer key. From there, a human grader compared the system's output against the answer key. To facilitate frequent manual grading, we developed the Key Updater Tool (KUT). This software compared the system's responses to the answer key. If any exact matches were found, they were automatically graded. System responses that did not match the answer key were fed to the human grader to evaluate, and then KUT would record the human grader's judgments by updating the key. New correct answers (not noticed during the original "enrichment" process) would be added, as well as new incorrect and inexact answers (the "inexact" metric is explained below), creating an "anti-key" and an "inexact-key." In subsequent uses of the tool, KUT would again only show the human grader responses that did not match any of the key strings. Over time, this tended to reduce the number of answers that needed grading.

As the administrators of the TREC corpus learned, human graders do not always agree on what is a correct answer (Voorhees 1998). Thus, for our effort, the same person who created the baseline answer key also drafted a document to capture basic answer-grading guidelines. A second person was given the guidelines and asked to read the stories and then grade the same system output using the baseline answer key for the training and DevTest[3] data (228 questions). As with the TREC evaluations, one judge was consistently more lenient than the other; under Grader B the system got six answers wrong that were marked correct by Grader A, an overall agreement rate of 97.4 percent.

We used a scoring metric that allowed the human graders to mark answers in one of three ways: correct, incorrect, and inexact.

[3] For the purposes of all the experimentation reported in this paper, the Remedia corpus was divided into three parts: training data, which was freely exploited for the purposes of developing the system; DevTest data, which was not directly observed by system developers at any time, but which was used during system development for experimenting with system modifications, alternate parameter settings and the like; and test data, which was reserved for infrequent evaluations, such as those performed in the production of this paper.

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press

- Correct: the response consists of just the phrase or clause needed to answer the question.
- Incorrect: the response does not answer the question.
- Inexact: the response contains more than just the answer, or is missing bits of the answer, or is technically correct, but is at the wrong level of semantic specificity.

Example 1 shows the former type of 'inexact"; example (2) the latter.

1. *Who is Christopher Robin?*
   - Correct answer: *the same person that you read about in the book, Winnie the Pooh*
   - Inexact answer: *the same person that you read about*
2. *Where was President Lincoln born?*
   - Correct answer: *Kentucky*
   - Inexact answer: *United States*

We were particularly interested in those inexact cases where the system identified the correct entity for answering a question, but chose an inappropriate mention (or mention extent as in (1) above) of the entity as its answer. To address this, we also evaluated the system in a looser fashion in which the system is given credit if it simply identifies a mention referring to correct underlying entity. The correct underlying entity is identified here as the coreference chain (using manual coreference annotations) that represents the entity

3 System architecture and overall performance

3.1 Architecture

This section gives an overview of ABCs, the system we built to take reading comprehension exams. While overall performance on the reading comprehension task was obviously a major factor influencing the design of the system, other factors also drove our approach. A primary motivation for our research is to develop an understanding for how and why the system succeeds and fails. To this end, we have taken into account a number of high-level requirements when building ABCs. First, the system has been designed to return exact phrase answers to questions where appropriate. This goes beyond our earliest RC system Deep Read that returned an entire sentence. The exact answer metric is consistent with how humans are expected to answer RC tests; it is also consistent with the most recent TREC QA evaluation. Another requirement has been for the system to be divided into components that are capable of independent evaluation. This allows us to assess how parts of the system contribute to overall performance. Finally, a major aspect of the design has been the use of a logical inference engine that makes use of abductive reasoning. A logical inference approach allows for a knowledge-based element to the process of question answering. The system is then able to justify its answer (in the form of a logical proof). This facilitates diagnosis because it is easier to understand

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press

why the system selected a particular phrase as the answer to a question. Also, as shown in the next section, an abductive reasoning framework allows for components of the system to be removed in ablation experiments that are useful for system analysis.

The major components of ABCs are similar to those of many other QA or RC systems. The system consists of four central processes.

Story processing is the process of converting the story into a *story knowledge base* – a set of propositions or facts representing syntactic and semantic information contained in the story. The process takes into account coreference information, named entity identification, syntactic parse information and any other results from linguistic processing phases.

Question processing in ABCs involves converting the question into a logical form query that can be run against the story knowledge base. This process is much the same as story processing in that the question is converted into a set of logical predicates. An additional phase of processing converts these logical predicates into a set of query terms with the appropriate unbound variables.

Answer selection involves searching for a proof of the logical query that results from question processing. The query is run against the knowledge base for the story to which the question applies. An abductive proof procedure is used to find a solution to the query. The proof procedure is abductive in that it will make assumptions when it is unable to find a purely deductive proof for the query. The solution to the query will include the binding of the variable associated with the answer to a particular phrase in the story.

Answer formulation This procedure takes the solution derived in answer selection and formulates the appropriate phrase to return as an answer. This currently involves choosing an appropriate mention among coreferential mentions. The system will also choose the appropriate extent of the phrase to return. It considers, for example, whether or not to include post-modifiers in the response.

The four main processes just described can be identified in Figure 2. Linguistic processing such as tokenization, sentence identification, named entity tagging and parsing are involved in both story and question processing. In ABCs, the results of the linguistic processing serve as input to the story interpretation module in the case of story processing and as input to the question interpretation module in the case of question processing.

The final phase of story processing is story interpretation, which involves mapping the annotations derived via linguistic processing into logical form. Similarly, the final phase of question processing is an interpretation process that maps the linguistic annotations into a set of query predicates.

The result of story processing is a set of logical facts comprising the story knowledge base. The knowledge base, as a whole, also consists of a set of inference rules and a semantic ontology. In the current version of the system, we use an ontology that is based on the WordNet hypernym/hyponym hierarchy. During the processing of a series of stories, the set of facts comprising the story knowledge

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press

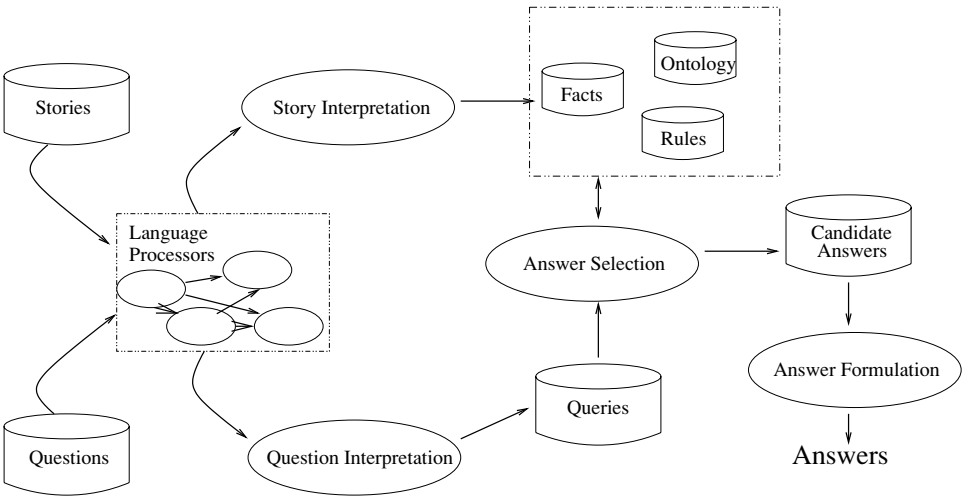


Fig. 2. ABCs architecture.

base will change for each new story. The inference rules and the ontology, however, remain the same.

Answer selection is carried out using an abductive proof procedure. This process takes as input the query predicates for a particular question. It tries to find a solution to the query using the rules, ontology and facts in the knowledge base. The abductive nature of the procedure, described in detail below, ensures that a solution is produced for each query even if an exact deductive proof for the query can not be found. Solution of the query results in a binding of the variables in the query to logical constants in the knowledge base. These logical constants can be understood as unique identifiers for phrases in the story. Thus, the bindings on the variables in the query correspond to phrases in the story, with one corresponding to the answer. Given this binding, the answer formulation phase is responsible for assembling an appropriate response from the text in the story.

3.1.1 Manual annotations

We have manually annotated the Remedia corpus to provide a gold-standard against which the system components can be evaluated. For example, we manually annotated the named entities in the stories according to the guidelines in Chinchor (1998). This allows us to compare the behavior of the system when using automatically produced named entity tags with system performance when manually produced tags are used instead. In this way, we can begin to assess the value of using named entity annotations and the effect of using the less-than-perfect named entity tagging components we currently have available.

In addition, we have created manual annotations for potential system components that do not yet exist in ABCs, allowing us to study how the inclusion of such a component might result in improved system performance. In place of a coreference component, we have manually annotated according to the MUC-7 guidelines (Hirschman 1997).

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press

Table 1. *Processing phases generated manually and by automated components*

	Sent. boundaries	Named entities	Parsing	Timex2	Coreference
Human	X	X		X	X
System	X	X	X		

Another manual annotation task involved temporal expressions. Both story and question text were assigned Timex2 annotations, in conformance with the TIDES temporal annotation standard (Ferro 2001; Gerber, Ferro, Mani, Sundheim, Wilson and Kozierok 2002). This standard goes beyond traditional named entity (TIMEX) tags by tagging both fully referring expressions (e.g. *March 5, 2002*) and indexical expressions (e.g. *yesterday, next week, three years ago*), as well as identifying the date or time referenced by the expression and normalizing it according to an extended version of the ISO 8601 (1997) standard (ISO1 1997).

Table 1 lists the types of annotations used to run experiments with ABCs. The table specifies in each case whether or not we currently have an automated component available for generation of the annotation type. Also indicated are the components for which manual annotations have been produced.[4]

The *standard* configuration for our system consists of all automated components *except for coreference and time normalization (Timex2)* which are provided in the form of manual annotations.

3.2 Performance

We evaluated our *standard* system on a blind set of evaluation data. In following earlier practice with the Remedia corpus (Hirschman *et al.* 1999), the evaluation data consisted of 60 stories. The remaining 57 stories constituted training and development test data which we divided roughly evenly between the two. In total, there are four natural divisions of this corpus based on grade level (levels 2 to 5). For historical reasons, levels 3 and 4 comprise the evaluation data and levels 2 and 5 training and DevTest data. Our DevTest data consisted of half of the stories from level 2 and half of the stories from level 5 with the other halves used as training data. Figure 3 shows the results of running the system on the evaluation data. For comparison, corresponding results for the DevTest data are shown as well. Scores are broken down by question type and both the *correct* scores and the (higher) *inexact* scores are indicated.

Performance varies by question type. *What* questions are particularly difficult for a number of reasons. During the generation of the logical queries it is difficult for the system to assign a correct semantic class to the focus of the question, i.e.

[4] We have not used the human annotated named entity annotations for the experiments reported here, however, as performance is typically identical (or very similar) to performance using the system generated named entitites on the Remedia data set.

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press

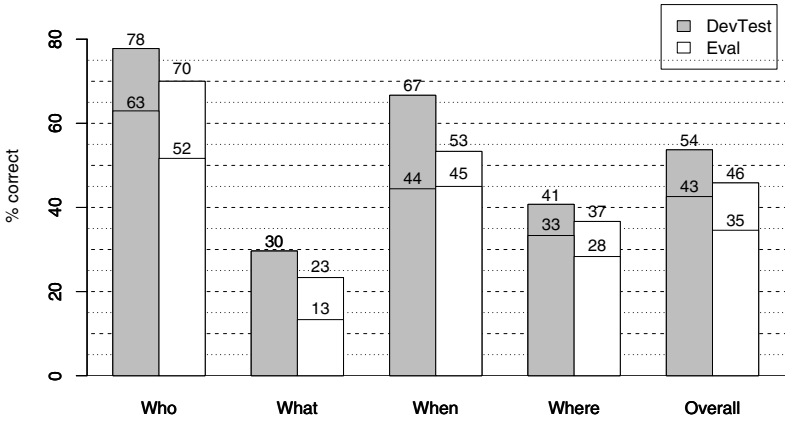


Fig. 3. Results on DevTest & Eval using standard system configuration: lower score is the strict correct metric, the higher score is the inexact metric.

Table 2. *Answer Types for* what *Questions in Training and DevTest. The number of* what *questions having the expected answer type is indicated*

activity	13	animal	9	artifact	7	vehicle	5	person	4
facility	3	organization	3	location	3	event	2	geological	2
body_part	1	celestial	1	music	2	organism	1	phenomenon	1
quantity	1	sound	1	statement	1	technique	1	weather	1

the *what* phrase. We currently have a set of very simple heuristics which assign a PERSON semantic type to *who* questions, DTIME (date/time) to *when* questions, and LOCATION to *where* questions. In the Remedia corpus such heuristics work relatively well for these question types. However, in the training and DevTest data there are 21 different semantic types referenced by the *wh*-word in *what* questions, as shown in Table 2.

Examples of questions that have activities as answers include *What did Jackie Cochran do?* and *What happened to Lincoln on April 14, 1865?*. The question *What is another name for groundhogs?* is answered with *woodchucks,* which is a reference to an animal. *What do they make?* is answered with a reference to an artifact (*sewing machines*), and so on. Without more sophisticated question analysis, this wide range of answer types for *what* questions makes it more difficult for the system to "understand" the question. This, and the relatively large percentage of Link parser errors on these types *what* questions, decrease the likelihood that the proof procedure will identify the correct answer.

Where questions are also particularly challenging. This is due in some part to the way in which locational information is conveyed in the stories. As shown in Table 3,

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press

Table 3. *DevTest questions that can be answered with a single sentence vs. questions for which the answer is spread across multiple sentences*

	Who	What	When	Where	Total
single	19	21	4	10	54
multiple	8	6	23	17	54

on average about half of the questions in our corpus can be answered on the basis of a single sentence. But this is not true of *when* and *where* questions, where an analysis of the DevTest data shows that the majority of *when* and *where* questions require more than one sentence to provide the evidence necessary to answer the question.

In examining the data, we have found that the *when* and the *where* of a story are often conveyed as the general setting of the topic, and are therefore not always within the same story sentence as the mentions of the entities or events that are the topic of a particular question. The inference engine is therefore much less likely to locate the correct answer phrase in these cases. The answers to *when* questions are easier to locate under these circumstances because most are answered with phrases that have been manually annotated with Timex2 tags (see section 3.1.1). For example, consider the following story excerpt containing Timex2 tags:

(NEW YORK, <TIMEX2 VAL="1929-10-29">October 29, 1929</TIMEX2>)
- <TIMEX2 VAL="1929-10-29">Today</TIMEX2> will go down in history
as one of the worst days ever. Millions of people lost some or all of their money
in the stock market... [seven sentences later]... People are saying the stock
market crashed.

Answering the question *When did the stock market crash* requires looking beyond the story sentence *People are saying the stock market crashed*. Fortunately, the answer does exist elsewhere in the document (in this case, twice), with the full reference encoded by the VAL attribute of the Timex2. Thus, this markup significantly narrows the field of possible answer phrases for *when* questions. A parallel example for the *where* question *Where is Mount Everest?* is seen in the following excerpt.

(<ENAMEX TYPE="LOCATION">NEPAL</ENAMEX>, 1953)
- Two mountain climbers have reached the top of the world's tallest mountain.
When they got to the top of <ENAMEX TYPE="LOCATION">Mount Everest</ENAMEX>, the two men had climbed 29,028 feet.

Here, to answer the question, one must obtain the answer *Nepal* from the by-line of the story, not from the sentence that mentions Mount Everest. In this case the

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press

task is made easier because *Nepal* has been tagged with a LOCATION tag. However, only *named* locations have been so annotated, and in the training and DevTest data *where* questions are answered by named locations only 53% of the time. The rest are answered by common noun phrases such as *an island in Alaska* or *cities with more than 10,000 people*. Furthermore, 36% of the common noun answers are entity types that our system is not designed to recognize as locations, such as facilities (e.g. *a pretty home, a museum, Sea World*), as well as non-locations such as organizations (e.g. Question: *Where did she serve for four years before becoming mayor?* Answer: *City Council*) and even haystacks (in answer to the question *Where were the men eating their lunch*?). Thus, in contrast to *what* questions, the challenge is not so much in understanding the *where* question, as in locating the answer in the text.

To get a sense for the significance of the difference between system performance on the DevTest data (53.7% of the questions answered correctly or inexact) and performance on the Eval data (45.0% correct or inexact), we ran one million replications of the following bootstrap procedure (Enfron and Tibshirani1993). For each replication, 27 stories were randomly selected as the *bootstrap sample* DevTest stories from the 87 stories of the combined DevTest and Eval sets,[5] and the remaining 60 stories were designated as the bootstrap sample Eval stories. With this partition, scores on both sets were calculated and the difference determined. Of these million differences, a difference as large as $53.7 - 45 = 8.7\%$ or larger was observed, by chance, 10.41% of the time. These results show that the differences between the DevTest and Eval tests, though consistent, are not significant at the standard $p = 5\%$ level.

Although the difference between the DevTest and Eval results may not be statistically significant, there are notable qualitative differences between the data sets in terms of overall language difficulty and the types of questions. In particular, the performance on *what* questions is considerably lower. We noticed considerable qualitative difference in *what* questions between questions from level2 and level5 in the training data. These differences affected the query generation process, especially. We thus suspect that further differences in *what* questions are to be found in the evaluation data which would be a possible explanation for the lower score on those questions.

The remainder of the analyses in this article are carried out on the DevTest data. This choice was made to minimize our interaction with and knowledge of the evaluation data, so that these stories may be used for future evaluations.

4 System design and component evaluation

The system, as briefly described in section 3.1, has a number of processing components or modules. A major part of designing and improving a system involves being able to determine how each part of the system contributes to the overall performance. While many of the system components interact with each other in complicated ways,

[5] The bootstrap sampling was based on stories rather than individual questions, to take into account the interdependence among questions of the same story.

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press

a useful method to estimate the utility of a particular processing stage or component is to remove that component from the system and then evaluate the diminished system. This *ablation* technique was used in earlier RC experiments (Hirschman *et al.* 1999). Another method of component analysis is to substitute the automated component with a manually derived set of annotations that represent the ideal system output for that particular phase of processing. This allows one to get an upper bound on the potential improvement offered by perfecting that component. This section discusses a number of experiments run to analyze system performance together with more detailed descriptions of ABCs, where necessary to understand the analyses.

4.1 ABCs linguistic components

4.1.1 Linguistic processing modules

In ABCs, linguistic processes such as tokenization and syntactic parsing are carried out by independent modules. These processing modules are connected via MITRE's Catalyst architecture (Mardis, Burger, Anand, Anderson, Griffith, Light, McHenry, Morgan and Ponte 2001). Catalyst is a distributed architecture for the communication of standoff annotations among independent text processing components. Catalyst allows for specification of a system of components based on the data dependencies between them. For each module, the input annotations required and the output annotations produced are declared. Catalyst assumes responsibility for delivery of the annotations; only the annotations required by a module are sent to it.

For parsing we made use of the Link Parser made available by Carnegie Mellon University based on the Link Grammar (Grinberg, Lafferty and Sleator 1995). The Link Parser produces dependency links between individual words in the sentence. An example of such a dependency parse is shown in Figure 5. In order to facilitate story and question interpretation, it was necessary to post-process the parser output to produce chunks, i.e. atomic, non-embedded phrases of a sentence.

The following is a brief description of each of the text processing phases.

initialization: To initiate linguistic processing, the story (or question) is read in, and all relevant SGML markup, such as datelines and headline delimiters, are converted by the system to the appropriate Catalyst annotations. Additionally, manual annotations simulating the output of a particular phase of processing are encoded as SGML tags and read in to the system at this point (see section 3.1.1).

tokenization: Tokenization consists of partitioning input text into the constituent lexical units (words, punctuation, etc.). The result is a *token* annotation wrapped appropriately around character sequences. It is these tokens that are the primitive elements processed by downstream linguistic processing modules.

sentence boundary detection: Sentence boundary detection is carried out by receiving *token* annotations and applying rules to determine where sentences begin and end. The result is emitted as a stream of *sentence* annotations.

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press

parsing: The Link Parser determines the parse structure of sentences. For a given sentence, *group* annotations are produced identifying the noun, verb, preposition, adjective and adverb groups composing the sentence. *Link* annotations are also produced, which specify the dependency relations existing between head tokens of the constituent groups. For example, the *Subj* link in Figure 5 indicates that *climbers* is the subject of *reached*.

4.1.2 Lexical classification

Two final phases of linguistic processing identify lexical types for noun, verb and adjective phrases. The first phase examines each of the noun phrases as determined by the Link Parser and classifies each as: *person*, *location*, *organization*, *date_time* or *none*. These classes correspond to the named entity types as defined in MUC-7. Using the training corpus, we have trained a statistical model to classify each noun phrase according to the five types above. In the second phase, noun phrases identified as *none* by the named entity classifier are subject to lexical classification through the use of WordNet. Specifically, the head noun of each candidate noun phrase is normalized using a morphological analyzer, and the WordNet synset corresponding to the resulting lexeme is identified as the type of the noun phrase. No attempt is made to disambiguate word senses; the first synset corresponding to the lexeme is chosen as the word sense. Verb and adjective phrases are handled similarly using WordNet.

4.2 Component performance

Figure 4 shows performance broken down by question types for various configurations of the system where specific components have been removed. The scores shown are based on the *exact correct* scoring metric. The *standard* configuration is the full system as described earlier with all automated components except for coreference and Timex2, which were manually annotated. The other configurations shown involve running the *standard* configuration with the removal of one component (be it an automated or manual component) from just the story processing, not the question processing. The *No NE* configuration removes the named entity classifier so that all noun phrases will get their type via WordNet with default type *entity*. Thus most proper nouns will be assigned type *entity*, since they are unlikely to appear in WordNet. Coreference is removed in the *No Coref* configuration. This primarily affects the answer formulation (discussed in detail later) since alternative coreferential mentions aren't available. *No Timex2* affects only *when* questions, unsurprisingly. The *No Syntax* configuration removes all the dependency links between phrases although the phrase boundaries derived from the parser are maintained. *No WordNet* removes the lexical knowledge found in WordNet from the knowledge base. As with the *No NE* configuration, all nouns will default to type *entity* and all verbs will default to type *event*.

As seen from the graph, removal of WordNet results in the largest drop in overall score. This is not surprising since WordNet is used to assign lexical type to all non

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press

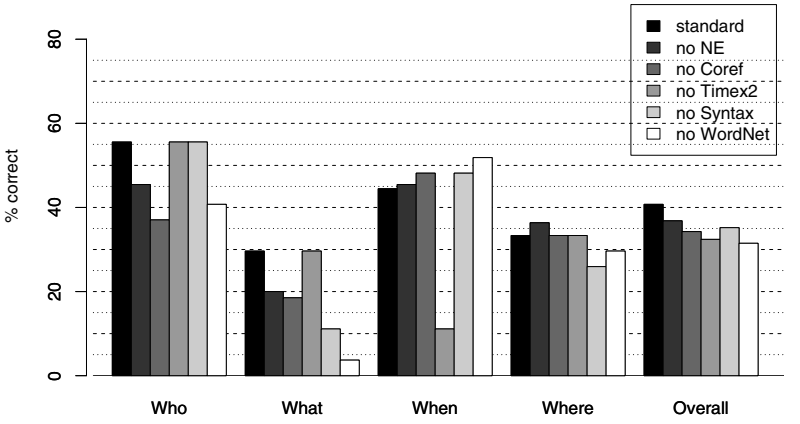
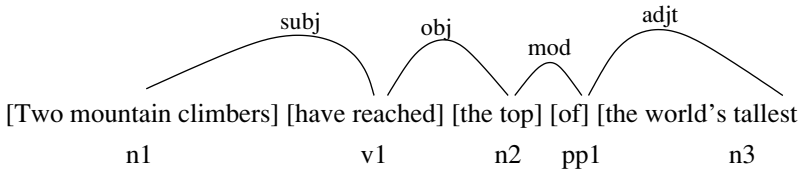


Fig. 4. Scores comparing different system configurations using the *exact correct* scoring metric using hand annotated queries.

named entities – this configuration can roughly be interpreted as running the system with only named entities for lexical identification. Interestingly, removing WordNet hurts performance more than removing named entities. From inspection, some named entities appear to be recovered through WordNet (e.g. "England"). Given the level and nature of this reading material (discussion of very common entitities) this occurs fairly frequently, especially in the case of *where* questions. *When* questions are, not surprisingly, negatively affected by removal of Timex2 annotations. Coreference is particularly important for *who* questions. *What* questions depend crucially on syntactic relationships and WordNet since their answers are often common noun or verb phrases.

These experiments provide useful insights into the system's behavior and indicate areas for further investigation. For example, in building a coreference system, it may make sense to concentrate on coreference between phrases of type *person* since coreference affects *who* questions mostly. Lexical knowledge would appear to be useful since performance drops considerably with the removal of WordNet despite the fact that our current use of WordNet is rather primitive. Note, however, that the score on *when* questions actually increases with the removal of WordNet providing an indication that wordsense errors are inhibiting better performance. More fine-grained and accurate lexical classification would therefore appear to be a fruitful area for future work. These ablation studies provide a method for "credit assignment" in the framework of current system performance. In future work, we plan to investigate the issue of *missing knowledge*: the kinds of rules (syntactic, lexical, ontological or inference) that would need to be added for ABCs to answer the remainder of the questions correctly.

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press



instance_of(n1, synset_of('climber'))
instance_of(v1, synset_of('reach'))
instance_of(n2, synset_of('top'))
intsance_of(pp1, 'of')
instance_of(n3, synset_of('mountain'))

subj(n1,v1)
obj(v1,n2)
mod(n2,pp1)
adjt(pp1,n3)

Fig. 5. Parse and corresponding propositions after interpretation.

4.3 Mapping to logical form

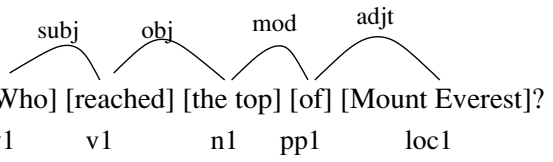
The result of all the language processing components described in section 4.1.1 (including any manual annotations provided) is a set of Catalyst annotations encoding markup over regions of text in the story or question. Recall that these annotations encode the tokens, sentence boundaries, named entities, and Timex2 markup as well as syntactic parse information consisting of phrases and dependency links between them. For stories, these annotations are used to construct the set of facts making up the logical representation of the story. For each question, an analogous process analyzes the annotations to produce a logical query whose solution will provide the answer returned by the system. We shall refer to this process of mapping the annotations to logical form as *semantic interpretation*, or simply as *interpretation*. A sequence of interpretation rules specifies this mapping. Among other things, these rules associate with noun, verb and adjective groups a synset from WordNet (this is simply the first synset associated with the token(s) in question). Figure 5 shows the predicates resulting from the application of a sequence of interpretation rules to a sentence in a story.

4.4 Query construction and evaluation

4.4.1 Query representation language

Properly understanding and interpreting the question is a crucial part of answering RC questions correctly. Interpreting questions in our framework involves deriving a logical form query from the question. This process consists of first deriving the base logical form representation of the question in the same manner as logical forms are produced in the story interpretation process. An additional phase of processing applies a set of condition-action rules to the base logical form to produce a query. These condition-action rules map the simpler, mostly syntactic predicates in the base logical form to more complex query predicates that encode more of the semantic content of the question. These rules have the additional effect of transforming some of the arguments into variables. These variables will then be bound to phrases in the story when the solution to the query is being sought.

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press



instance_of(w1, who)
instance_of(v1, synset_of('reach'))
instance_of(n1, synset_of('top'))
instance_of(pp1, 'of')
instance_of(loc1, location)

subj(w1,v1)
obj(v1,n1)
mod(n1,pp1)
adjt(pp1,loc1)

query(X, [instance_of(X,person), instance_of(Y,location), has_mention(Y,"Mount Everest"), instance_of(Z,top), verb(synset_of('reach'),X,Z,V), modifies(Y,_,Z)])

Fig. 6. Parse, corresponding propositions and query for the question *Who reached the top of Mount Everest?*

As with most of the linguistic processing phases, we have manually converted each question to the correct query, according to our query representation scheme. This provided a gold standard that we could use to measure the performance of the automatic query generator. We have also used this gold standard to assess the relationship of imperfections in query processing to overall system performance. Figure 6 shows a question with its corresponding syntactic parse, the base logical form derived from the parse and lexico-semantic information, and the query derived from the base logical form.

The derived query has two components. The first is a variable, the second a conjunction of terms to be resolved. The resolution of the conjunction will result in a binding of the answer variable to a phrase of the story. This phrase will be returned as the answer to the question. The query representation scheme makes use of seven basic predicate terms. The set of terms covers entity types using (*instance_of*), the string realizations for entities using (*has_mention*), verbs (*verb*), predicate nominals (*copula*), predicate adjectives (*property*), and modifiers (*modifies*).

4.4.2 Evaluation of the query generator

A query can be generated incorrectly due to a number of errors stemming from such subcomponents as parsing, named entity and lexical identification and the query semantic interpretation process. Table 4 shows direct evaluation results of the system generated queries vs. the manually generated queries on the 27 DevTest stories. The *exact match* scores represent the percentage of questions for which the system generated query is exactly the same as the manually derived query modulo variable renaming. The *f-measure* results provide a balanced precision/recall score based on matching predicates between the system queries and manually derived queries. For example, consider two queries q_{man}, and q_{sys} and let $q_{man} = [a(X,Y), b(Y), c(Y,Z)]$ and $q_{sys} = [a(A,C), b(C), c(D,C), d(E)]$. q_{sys} contains an extra predicate not found in q_{man}, namely $d(E)$. This counts as a recall error. Additionally, it is easy to see that a mapping bewteen the variables in the remaining predicates (not considering $d(E)$)

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press

Table 4. *Accuracy of query generator on DevTest data*

	who	what	when	where	overall
exact match	29.6	14.8	44.4	33.3	30.6
f-measure	84.2	77.8	88.2	85.7	84.0

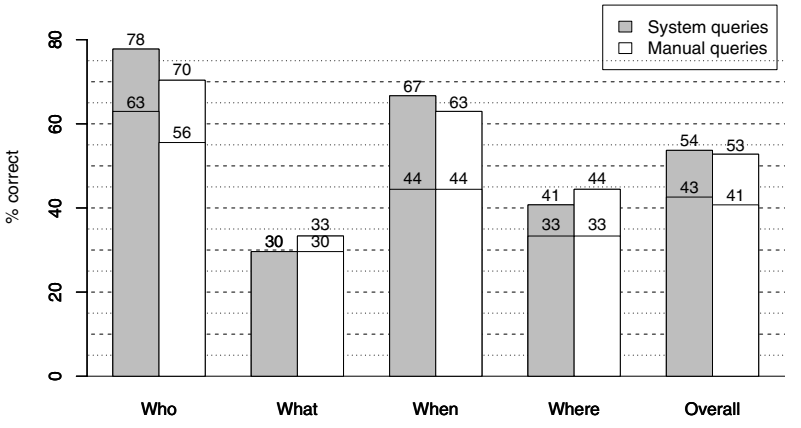


Fig. 7. Comparisons of manually vs. system derived queries.

is not possible. Removing the predicate $c/2$ from *both* querires would result in isomorphic queries $q'_{man} = [a(X, Y), b(Y)]$ and $q'_{sys} = [a(A, C), b(C)]$ with $A = X, Y = C$, however both a recall and precision error would result. The recall for this example would be $\frac{2}{4} = 0.5$ and precision $\frac{2}{3}$. The disparity between the exact match and f-measure scores shown in Table 4 can be explained by noting that generating any particular query term correctly can be done with probability 0.8 (based loosely on the f-measure scores), and getting an entire query correct that involves 5 query terms (which is roughly the average length of a query) would then have probability $0.8^5 = 0.328$, in the ballpark of the exact match scores shown.

Figure 7 compares the results using the system generated queries to the manually derived queries on the DevTest data. Interestingly, performance using the system generated queries is slightly higher than using the manual queries, with performance on *who* questions standing out in particular. Despite only generating exactly correct queries for 29.6 percent of the *who* questions, the answer accuracy goes *up* on these questions in comparison to the manually derived queries. One possible explanation for this is that certain query predicates that tend to act as distractors are exactly those query predicates not being generated by the query generation component. For example, given the question *Who is the captain of the sailboat?* the corresponding

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press

query would include a query term such as *instance_of(X,sailboat)*. However, the answer sentence for this question is *The name of the captain is Thor*, which does not contain a mention of a sailboat. Given our current system, this question would require the abduction of the *instance_of(X,sailboat)* term. If this query term was not generated at all (as is the case in this example), this term would not need to be abduced, thus lowering the cost associated with this solution and making it more likely to be selected. This points to a need for further work on identifying the salient parts of a question either in the query generation stage or in the answering stage.

4.5 Answer selection and formulation

The two final stages in processing a question are discussed in this section: answer selection and answer formulation. Answer selection involves use of an abductive proof procedure in which the facts derived from the story, the ontological facts, and general inference rules, serve as a knowledge base used to derive a proof for the query. The goal of answer formulation is to select the appropriate span of text to return as an answer *phrase* – this includes disambiguating among coreferential mentions as well as returning the appropriate mention extent.

4.5.1 Answer selection

Answer selection is carried out via an inference mechanism that searches for a solution to a query using a logical proof procedure that includes abduction. A solution for a query results in a binding of the unbound variables in the query to logical constants. The constants correspond to noun, verb, and adjectival phrases in the story. The selected answer results from the binding of the answer variable to one of these logical constants in the course of solving the query.

Inference is performed by way of a proof procedure that applies a combination of deductive and abductive reasoning steps to arrive at a solution to the query. Inference operates over a set of Horn clauses representing the static part of the knowledge base and the logical representation of the story derived from interpretation, lexical classification and coreference as described above. For deduction, we have employed a standard backward-chaining inference engine: SLD resolution as found in Prolog.

Abduction provides a measure of robustness that is typically lacking in logic-based systems. In a purely deductive scheme, every fact supporting a proof must be known to the inference mechanism. With abduction, however, a proof can be accepted if there is partial support for it, even though some facts needed to produce a complete proof may be lacking. In addition, abduction can be useful for diagnosing system behavior. As a result of abduction, proofs will be completed for queries that would otherwise fail. These proofs will leave indications of system weakness in the form of query goals or sub-goals that needed to be abduced because the system was unable to provide adequate proofs. Analysis of the type of abductions made as part of the inference process provides insight into *how* the system is behaving when answering questions. Analysis of which predicates are assumed to be true, how often various

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press

predicate types are assumed to be true, and under what conditions these assumptions are made, is useful for guiding further system development.

Generally, there may be many abductive hypotheses possible for a given set of goals. In practice, abductive reasoning often amounts to finding the *best* hypothesis, with respect to some metric, given a set of goals or observations. In our framework, for each abducible predicate A, we have assigned an integer cost for abducing it – we assigned costs manually, tuning them on the training data. The *best* explanation in ABCs, then, is one with lowest total abductive cost. The proof procedure carries out an iterative deepening-type search where the total cost of allowed abductions is incremented. Thus, all proofs with cost zero are searched for first and if no solutions are found, the procedure tries to find solutions with cost one, then two, etc. Thus, the lowest cost solution or solutions are found first. The abductive cost where the first solutions are found we term the *minimum abductive level*. The proofs found at the minimum abductive level are ordered based on the order of the query terms and the order of the clauses in the knowledge base. The results reported thus far have simply selected the first proof and corresponding solution as the answer solution.

The heart of the knowledge base in ABCs is WordNet. Rules make use of the hypernym/hyponym hierarchy so, for example, it is possible to derive that *the world's tallest mountain* is a *location* since *mountain* is mapped to a synset that falls underneath the top-level synset corresponding to a *location*. Additional rules in ABCs handle syntax. In particular, these rules map the various low-level binary syntax predicates (e.g. *subj*, *obj*) into the query-level terms such as *modifies* and *verb*. Recall that our knowledge base is entirely back-chaining, so the query-level terms in the story are never explicitly realized, but are derived only at query time. These rules were manually developed by examining sentences in the training data.

Figure 8 shows a proof of the query derived from the question *Who reached the top of Mount Everest?*. The proof is based on the query from Figure 6 and the answer sentence from Figure 5. The numbered bold query terms indicate terms found in the query, whereas numbered non-bold terms indicate intermediate resolvents. The indented bulleted terms indicate rules or facts in the knowledge base that are resolved against the current sub goal. Variable bindings for each resolution step are shown on lines beginning with a double hyphen. The first query term resolved is *instance_of(X,person)*. The proof procedure tries to apply the rule *instance_of(X,Y) IF instance_of(X,Z), wn_hypernym(Z,Y)* where the *wn_hypernym(Z,Y)* predicate is true if Y is a WordNet hypernym of Z. In this example, however, there is a word sense error. This goal will fail in step 3 because the particular sense chosen for the lexical item, *climber*, mistakenly refers to the vine-like sense of a climber, i.e. a plant. The failure of the goal *wn_hypernym(Z,Y)* results in the failure of the top-level goal *instance_of(X,person)*. Rather than allowing this goal to fail, it is abduced. A cost of 1 is paid for abducing this predicate. The variable X remains unbound at this point with the hope that it will become bound by other references to the variable in the query. The remainder of the proof follows standard resolution. Note that the variable X, left unbound in the abductive step above, is bound to *n1* due to the syntactic relation it shares with the verb in step 6. The proof finishes with making only one abductive step for a total cost of 1.

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press

Query: instance_of(X,person), instance_of(Y,location), verb(synset_of('reach',X,Z,V), instance_of(Z,synset_of('top')),modifies(Y,_,Z), has_mention(Y,"Mount Everest").

1. **instance_of(X, person)**, . . .
   - instance_of(X, Y) IF instance_of(X, Z), wn_hypernym(Z, Y)
     — {Y=person}
2. instance_of(X, Z), wn_hypernym(Z,person), . . .
   - instance_of(n1, synset_of('climber'))
     — {X=n1, Z=synset_of('climber')}
3. wn_hypernym(synset_of('climber'), person), . . .
   - FAILS DUE TO WORD SENSE ERROR.
     — ABDUCE: instance_of(X, person)
4. **instance_of(Y,location)**, . . .
   - instance_of(n3,location)
     — {Y=n3}
5. **verb(synset_of('reach'), X, Y, V)**, . . .
   - verb(C, X, Y, V) IF subj(X, V), obj(V, Y), instance_of(V, C)
6. subj(X, V), obj(V, Y), instance_of(V, C)
   - subj(n1, v1), obj(v1, n2), instance_of(v1, synset_of('reach'))
     — {X=n1, Y=n2, V=v1,C=synset_of('reach')}
7. instance_of(n2, synset_of('top')), . . .
   - instance_of(n2, synset_of('top'))
8. **modifies(Z, _, n2)**, . . .
   - modifies(Z, _, n2) IF mod(n2, P), adj(P, Z)
9. mod(n2, P), adj(P, Z), . . .
   - mod(n2, pp1), adj(pp1, n3)
     — {P=pp1, Z=n3}
10. **has_mention(n3, ['Mount', 'Everest'])**, . . .
    - has_mention(n3, ['Mount', 'Everest'])
11. DONE

Fig. 8. Proof of the query corresponding to the question *Who reached the top of Mount Everest?*

Our approach utilizing abductive inference for finding the most likely answer for a question bears strong similarity to the approach in Moldavan *et al.* (2002). That work uses a somewhat different logical representation, albeit also Davidsonian. They also accomodate non-weighted abduction in Harabagiu *et al.* (2000), and in subsequent work (Moldavan *et al.* 2002) use a weighted "relaxation" mechansism when exact proofs are not found. Also related is the work of Greenwood (2002), which uses a similar Davidsonian logic form but computes the best answer by measuring the weighted overlap between predicates in the query and predicates in the candidate answer sentence. Our abductive framework here is based strongly on the foundational work in Hobbs *et al.* (1990).

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press

4.5.2 Answer formulation

The final step in responding to a question in ABCs involves assembling an answer phrase that forms an exact response to the question. Of all the mentions of an entity within the story, the system must choose the best mention to use in the context of the question asked, as well as the most appropriate form of the mention. Currently, our answer formulation component consists of two parts. The first part selects the correct mention among other possible coreferential mentions of the answer mention. For example, one of the heuristics for selecting the correct coreferential mention is:

- if the mention has a coreferential named mention, return the longest named mention as the answer *unless some exception holds.*

In the example in the previous section, the answer is bound to the reference associated with the mention *Two mountain climbers.* While this mention refers to the correct entity (i.e. the two climbers, Hillary and Tenzig), it is not the preferred phrase to return as an answer. Generally, proper noun phrases are preferred as answer phrases over descriptors, although this is not always the case. For example, the following rule serves as an exception:

- if the answer entity has a coreferential named mention whose string is also found in the question, do not return a named mention, instead return a descriptor.

This exception handles situations arising in questions such as: *Who is Christopher Robin?* This type of question is clearly looking for a descriptor. Generally, however, for the question types found in this corpus, it makes sense to avoid returning answer phrases that are also found in the question. Other heuristics are encoded as exceptions in a similar way.

Also at this stage in the answer formulation process, answers in the form of time expressions or location references may need to be normalized. Timex2 annotations provide normalized time expressions, as discussed in section 3.1.1. Locations are normalized by choosing the appropriate coreferential mention. A useful heuristic in this regard is to prefer the longest string.

The second part of answer formulation is deciding upon the appropriate extent of the mention to return. For many questions, the noun phrase group associated with the answer mention is the appropriate phrase to return. If the noun phrase has a post-modifying expression attached to it, however, this is often an important part of the answer. Indeed, responses that do not include a post-modifying expression where appropriate will typically be considered incorrect or inexact. Given the question *Who is Christopher Robin?* merely answering with "the same person" instead of "the same person that you read about in Winnie the Pooh" is inappropriate. Similarly, in the case of verbs, it would be preferred to return the answer phrase "wrote a book", for example, instead of simply "wrote" in response to the question *What did he do when Chris was three years old?*.

Performance of the answer formulation component can be ascertained by looking at the difference between the single phrase score (i.e. the score obtained using the *standard* system described earlier, and producing a single phrase as output) and

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press

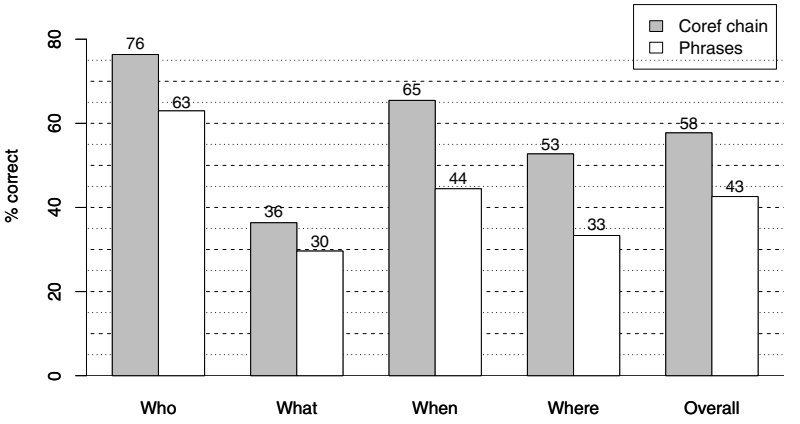


Fig. 9. Single phrase and coreference chain-based evaluation results on DevTest data.

the scores derived by classifying an answer as correct if it found a mention from the correct coreference chain. The presentation of the response, answer formulation, is not considered in this case. Figure 9 compares the single phrase evaluation results with the coreference chain results on the DevTest data. Roughly 54% of the questions are considered exactly correct based on the coreference chain metric. This is in contrast to roughly 43% using the single phrase evaluation. This indicates that better answer generation offers potential for considerably improving performance.

5 Abduction based comprehension

5.1 Abduction results and analysis

In addition to providing flexibility and robustness, the frequency and types of abductive inferences provide detailed insight into the system's behavior. They provide clues to where the system's strengths and weaknesses lie – providing guidance for future development. For example, if the system is often required to abduce that *X is a person*, this may provide an indication that the lexicon for types of people needs to be expanded, or the named entity person identifier needs to be improved. Additionally, examination of the frequency of abduced predicates provides a basis for adjusting the various costs attached to abducible predicates. Table 5 indicates the costs used in the current version of the system.

Each abductive proof of a query has a particular abductive cost associated with it which is the sum of the costs of all the abduced query terms according to Table 5. These costs were assigned manually by trial-and-error on the training data. The maximum possible abductive cost for a solution would be attained if *all* the query terms were abduced. Table 6 shows the averages for the maximum possible abductive cost for a query broken down by question type. The *actual cost* is the

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press

Table 5. *Costs assigned for abducible predicates*

instance_of_time	3	instance_of_location	3	instance_of_entity	1	modifies	1
verb	2	has_mention	1	copula	2		

Table 6. *Average abductive cost broken down by question type*

	who	what	when	where	overall
actual cost	1.70	1.85	2.51	2.33	2.10
maximum cost	9.96	8.78	12.70	12.11	10.89
abductive cost ratio	0.17	0.21	0.20	0.19	0.19

average abductive cost at the minimum abductive level for the solutions actually found by the system. The percentage of the average actual abductive cost to the average maximum abductive cost is close to 20% over all four question types.

To better understand the correlation between abductive cost and performance, it is helpful to compare the abductive cost of the questions answered correctly vs. questions answered incorrectly. Figure 10 shows the number of questions answer correctly and incorrectly at each abductive cost in the top graph and the probability that an answer is correct given the abductive cost in the bottom graph. The expected abductive cost for questions answered correctly is 1.78 compared to 2.48 for questions answered incorrectly. The minimum abductive level for a particular question can be used to estimate the probability of correctness.

5.2 Multiple solutions analysis

Due to the nature of our abductive inference engine, there may well exist multiple acceptable proofs and solutions with the same minimal abductive cost. The order of these proofs is determined by the order of the clauses (rules and facts) in the knowledge base. The results reported to this point in the paper have simply selected the first proof and corresponding solution and ignored the others. Like the abductive cost analysis in the preceding subsection, considering alternative solutions provides details of system behavior and directions for future system improvement. For example, if a large number of solutions exists at the minimum abductive level, this may indicate that the rules or the queries are too general. Consider the extreme case where a *Who* question simply consists of the query *person(X)*. This query would have many solutions (with abductive cost 0), namely all the entities of type *person* in the story. Likely, one of these would be the correct answer, however now the problem is reduced to simply choosing among all the *person* entities in the story. On the contrary, queries that are too specific and/or rules that are too restrictive are more likely to result in proofs with fewer solutions at the minimum cost. However,

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press

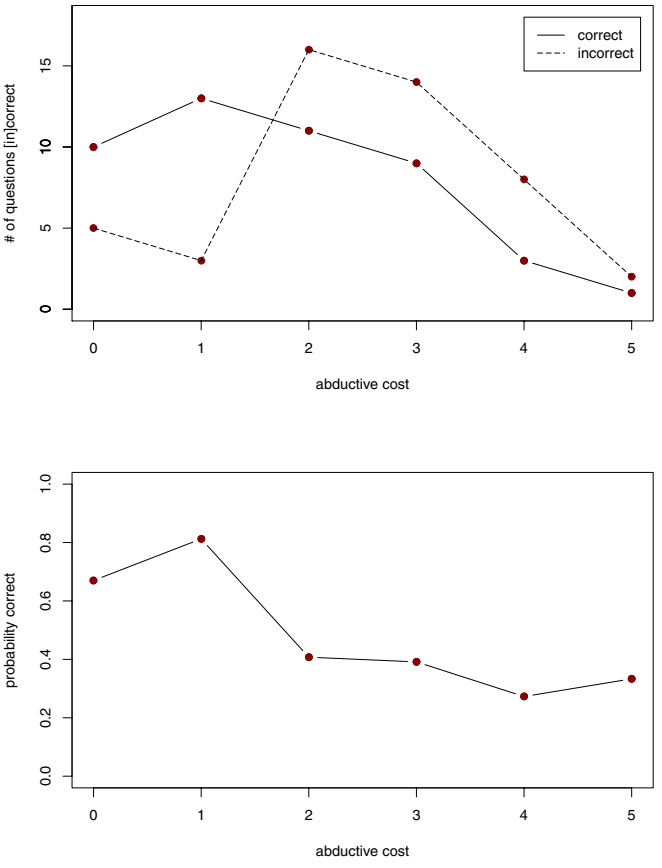


Fig. 10. Number of questions answered correctly and incorrectly (upper graph) and the corresponding probability of correctness (lower graph) at a given abductive cost.

these proofs will likely have a higher minimum cost and higher probability of finding an incorrect solution.

Figure 11 summarizes the analysis of the effect of multiple solutions on system performance. All numbers shown here are calculated using the coreference chain evaluation method described above. The *max* bar shows the percentage of questions answered correctly, broken down by question type, when one or more of the solutions found at the minimum abductive level is correct. It is the score the system would obtain it were to always choose correctly among competing solutions. The *min* bar shows the score obtained where the system only chooses correctly when all of the options are correct. It is the worst score possible. The *expected* bar is the expected score when selection among the solutions at the minimum abductive level is random. Finally, the *actual* bar gives the actual system scores obtained by choosing the first solution. These results are obtained by looking at all the *solutions* at the minimum abductive level for each question, not the *proofs*. The difference is that there may

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press

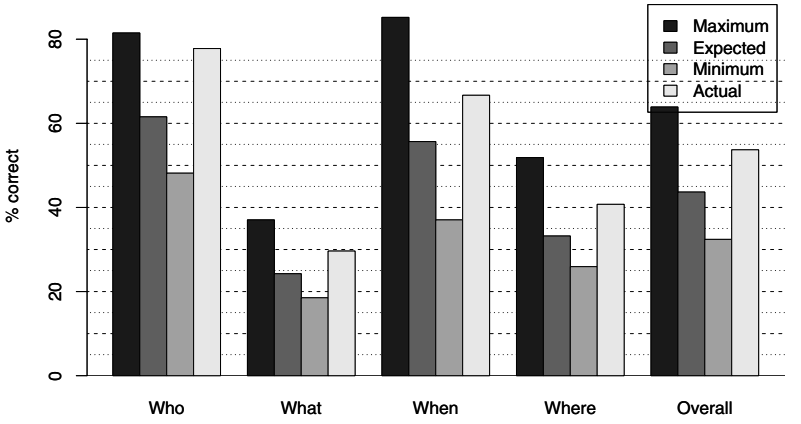


Fig. 11. Max, min, expected and actual scores considering multiple solutions.

be multiple proofs for a particular solution. Since we're interested in solutions (and ultimately answers) and not proofs, however, it seems appropriate to ignore alternate proofs for the same solution.

One interesting observation from Table 11 is that the *actual* score is notably higher than the *expected* score. Recall that the *actual* score is derived by simply taking the first solution found and that the order of the solutions is determined by the order in which rule and fact clauses appear in the knowledge base. The most likely explanation for the higher *actual* score is that clauses were ordered in such a way that better solutions are found first. Two ordering biases might help explain this. Firstly, the facts derived from the story itself are positioned in the knowledge base in roughly the order they appear in the story. This provides a bias towards selecting proofs with answers closer to the beginning of the story. It may well be the case that answers tend to be found closer to the beginning of a story. Secondly, the manually constructed inference rules were derived from looking at the data. The tendency would be to place rules capturing more common phenomena first. Latter rules capture more specific cases that may also have exceptions. We have carried out some experiments re-ordering both the rule clauses and the fact clauses derived from the story which tend to result in notable drops in actual performance. This observation indicates the need to either find good orderings among clauses or to develop a mechanism for selecting among multiple proofs.

6 Discussion and further research

This paper has presented ABCs, a system for taking reading comprehension exams. ABCs has been designed with the goal of understanding the natural language components involved in answering reading comprehension questions, and the interdependencies of these components. The system uses an abductive inference

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press

engine that reasons to the lowest cost solution. This allows assumptions to be made for missing information, when searching for a solution to the query derived from the question. The nature and quantity of such abductions contribute to the cost of the solution. The overall abductive process makes the system's reasoning process more transparent, by keeping a record of abduced predicates and missing information. Ablation experiments indicate a need for more sophisticated lexical classification, as well as the need to independently evaluate sub-components such as parsing.

One area for future work is based on the analysis of multiple solutions indicated in section 5.2. Our results showed that there is significant potential to improve performance by choosing the correct proof and corresponding solution among the many that may reside at the minimum abductive cost level. An interesting and elegant approach to discriminate among multiple proofs is Stochastic Logic Programming (SLP) described in (Muggleton 1996). SLP incorporates probability distributions over clauses. Inference in SLP amounts to finding the proof with highest probability. Methods exist for finding the parameters of SLP from training data (Cussens 2001). This approach fits within our ongoing research objective of combining abductive and probabilistic inference. We hypothesize that we can improve performance and adaptability through statistical means, while maintaining the transparency of a rich symbolic representation.

We have also used ABCs as the basis for an interactive RC system in which abduced predicates are converted into questions presented to a user. The user must validate the proposed abduced predicate, e.g. *climber is_a_kind_of person*, for the proof to proceed. When a user rejects a proposed abduction, the system is forced to backtrack through alternative answers and hypothesize new assumptions in order complete the proof. This approach allows the system to acquire useful knowledge during the course of answering a question – namely those assumptions which the user *does*, in fact, verify. We also plan to experiment with an extension of this paradigm, where the system tries to answer questions through the use of automatic question answering, information retrieval, machine readable dictionaries and thesauri. We believe that the abductive framework of ABCs has the potential to support a system that learns through its interaction with expert users and external resources, allowing it to acquire (and even "learn") new material by reading with understanding.

References

ANLP/NAACL (2000) *Workshop on Reading Comprehension Tests as Evaluation for Computer-Based Understanding Systems*, May.

Brill, E., Lin, J., Banko, M., Dumais, S. and Ng, A. (2001) Data-intensive question answering. *Proceedings of the 10th Text Retrieval Conference (TREC-2001)*, Gaithersburg, MD.

Charniak, E. *et al.* (2000) Reading comprehension programs in a statistical-language-processing class. *Workshop on Reading Comprehension Tests as Evaluation for Computer-Based Understanding Systems: ANLP/NAACL 2000)*, pp. 1–5.

Chinchor, N. (1998) Muc-7 named entity task definition (version 3.5).

Cussens, J. (2001) Parameter estimation in stochastic logic programs. *Machine Learning*. **44** (3): 245–271.

Enfron, B. and Tibshirani, R. (1993) *An Introduction to the Bootstrap*. Chapman & Hall.

Fellbaum, C. (editor) (1998) *WordNet: An Electronic Lexical Database*. MIT Press.

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press

Ferro, L. (2001) Instruction manual for the annotation of temporal expressions. Technical Report MTR 01W0000046, The MITRE Corporation, McLean, VA.

Gerber, L., Ferro, L., Mani, I., Sundheim, B., Wilson, G. and Kozierok, R. (2002) Annotating temporal information: From theory to practice. *Proceedings of the 2002 Conference on Human Language Technology*, San Diego, CA.

Green, B., Wolf, A., Chomsky, C. and Laughery, K. (1986) Baseball: An automatic question answerer. In: Grosz, B. J., Jones, K. S. and Webber, B. L., editors, *Readings in Natural Language Processing*. Morgan Kaufmann, pp. 545–550.

Greenwood, M. A., Roberts, I. and Gaizauskas, R. (2002) University of Sheffield TREC 2002 Q&A System. *Proceedings of the Eleventh Text Retrieval Conference (TREC-2002)*.

Grinberg, D., Lafferty, J. and Sleator, D. (1995) A robust parsing algorithm for link grammars. *Proceedings of the Fourth International Workshop on Parsing Technologies*, Prague, Czech Republic.

Grosz, B. J., Jones, K. S. and Webber, B. L. (1986) Introduction to Chapter 6. In: Grosz, B. J., Jones, K. S. and Webber, B. L., editors, *Readings in Natural Language Processing*. Morgan Kaufmann, pp. 539–544.

Grosz, B. J., Jones, K. S. and Webber, B. L., editors, *Reading in Natural Language Processing*. Morgan Kaufmann.

Harabagiu, S., Maiorano, S. J. and Pasca, M. A. (2003) Open-domain textual question answering techniques. *Natural Language Engineering* **9**: 231–267.

Harabagiu, S., Pasca, M. and Maiorano, S. (2000) Experiments with open-domain textual question answering. *Proceedings of International Conference on Computational Linguistics (COLING-2000)*. Morgan Kaufmann.

Hendrix, G., Sacerdoti, E., Sagalowicz, D. and Slocum, J. (1986) Developing a natural language interface to complex data. In: Grosz, B. J., Jones, K. S. and Webber, B. L., editors, *Reading in Natural Language Processing*. Morgan Kaufmann, pp. 563–584.

Hirschman, L. (19997) MUC-7 Coreference Task Definition, Version 3.0. *Proceedings of the Seventh Message Understanding Conference (MUC-7)*.

Hirschman, L. and Gaizauskas, R. (2001) Natural language question answering: The view from here. *Natural Language Engineering*, **7**(4): 275–300.

Hirschman, L., Light, M., Breck, E. and Burger, J. D. (1999) Deep read: A reading comprehension system. *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pp. 325–332.

Hobbs, J. R., Stickel, M. E., Appelt, D. A. and Martin, P. (1990) Interpretation as Abduction. Technical Report 499. AI Center, SRI International.

ISO-8601 (1997) ftp://ftp.qsl.net/pub/g1smd/8601v03.pdf.

Kakas, A., Kowalski, R. A. and Toni, F. (1993) Abductive Logic Programming. *J. Logic & Computation*, **2**(6): 719–770.

Lehnert, W. (1986) A conceptual theory of question answering. In: Grosz, B. J., Jones, K. S. and Webber, B. L., editors, *Reading in Natural Language Processing*. Morgan Kaufmann, pp. 651–658.

Light, M., Mann, G., Riloff, E. and Breck, E. (2001) Analyses for elucidating current question answering technology. *Natural Langauge Engineering* **7**(4): 325–342.

Mardis, S., Burger, J., Anand, P., Anderson, D., Griffith, J., Light, M., McHenry, C., Morgan, A. and Ponte, J. (2001) Qanda and the catalyst architecture. *Proceedings of the Tenth Text Retrieval Conference (TREC-10)*, Gathersburg, MD.

Moldovan, D., Harabagiu, S., Girju, R., Morarescu, P., Lacatusu, F., Novischi, A., Badulescu, A. and Bolohan, O. (2002) LCC Tools for Question Answering. *Proceedings of the Eleventh Text Retrieval Conference (TREC 2002)*.

Muggleton, S. (1996) Stochastic logic programs. In: L. de Raedt, editor, *Advances in Inductive Logic Programming*, pp. 254–264. IOS Press.

Muggleton, S. (2002) Learning structure and parameters of stochastic logic programs. *Linkoeping University Electronic Press*.

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press

Riloff, E. and Thelen, M. (2000) A rule-based question answering system for reading comprehension tests. *Workshop on Reading Comprehension Tests as Evaluation for Computer-Based Understanding Systems (ANLP/NAACL 2000)*, pp. 13–19.

Schank, R. and Abelson, R. (1977) *Scripts, Plans, Goals and Understanding: an Inquiry into Human Knowledge Structures*. Lawrence Erlbaum.

Vilain, M. (1995) Semantic inference in natural language: Validating a tractable approach. *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, Montreal, Canada.

Voorhees, E. (1998) Variations in relevance judegements and the measurement of retrieval effectiveness. *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Melbourne, Australia.

Voorhees, E. (2002) Overview of the TREC 2002 question answering track. *Proceedings of the Eleventh Text Retrieval Conference (TREC 2002)*.

Wang, W., Auer, J., Parasuraman, R., Zubarev, I., Brandyberry, D. and Harper, M. P. (2002) A question-answering system developed as a project in a natural language processing course. *Workshop on Reading Comprehension Tests as Evaluation for Computer-Based Understanding Systems (ANLP/NAACL 2002)*, pp. 28–32.

Webber, B. L. (1986) Questions, answers and responses: interacting with knoweldge-base systems. *On Knowledge Base Management Systems: Integrating Artificial Intelligence and Database Technologies*, Springer-Verlag, pp. 365–402.

Wellner, B. (1999) An abductive-inductive learning framework for logic-based agents. Master's thesis, Imperial College of Science, Technology and Medicine, University of London.

https://doi.org/10.1017/S1351324905004018 Published online by Cambridge University Press