Advances in Cognitive Systems 6 (2018) 17–29 Submitted 6/2018; published 9/2018

Natural Language Access: When Reasoning Makes Sense

Richard Waldinger WALDINGER@AI.SRI.COM
Artificial Intelligence Center, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025 USA

Cleo Condoravdi CLEOC@CSLI.STANFORD.EDU
Department of Linguistics, Stanford University, Stanford, CA 94305 USA

Kyle Richardson KYLE@IMS.UNI-STUTTGART.DE
Institute for Natural Language Processing, University of Stuttgart, Pfaffenwaldring 5b, 70569 Stuttgart Germany

Asuman Suenbuel ASUMAN.SUENBUEL@SAP.COM
SAP Labs, 3412 Hillview Avenue, Palo Alto, CA 94304 USA

Abstract

Natural language is one of the more appealing ways by which people can interact with computers, but up to now its application has been severely constrained. We argue that to use natural language effectively, we must have both a deep understanding of the subject domain and a general-purpose reasoning capability. We illustrate the issues with SAP-QUEST, a proof-of-concept system for natural language question answering over a set of data sources for business enterprise applications, but the argument can be applied more generally to dialogue-style interfaces over a variety of subject domains.

1. Motivation

While natural language and speech interfaces are playing a larger role in our efforts to interact with machines, current language interfaces have limited capability. Yet older interaction styles, such as programming languages and graphic interfaces, are of value for only a limited class of users, and natural language interfaces will be of increasing importance to a wide variety of applications, including robotics, internet of things, and control of appliances. Kaplan (2013) offers a convincing discussion of the limitations of earlier interface styles. In this paper, we argue for a general reasoning capability operating over a substantial body of subject domain knowledge. We are interested in a style of natural-language question answering in which:

- Questions are in ordinary language, not a controlled sublanguage;
- Answers are precise, not just references to web pages or documents that may or may not answer the question;

- Answers can be obtained from multiple sources, which may be disparate; they may not have been intended for use together and they may not adhere to uniform standards;
- Answers need not appear explicitly in any one source; they may be deduced or computed from information obtained from multiple sources;
- Users can express queries incrementally; on seeing the answers to one question, the user may ask follow-on questions that refer back to earlier ones and that may broaden the class of answers, impose restrictions on answers, or request further information.

In this paper, we argue that question answering requires a general reasoning capability, equivalent in power to a theorem prover. In this context, reasoning plays multiple roles:

- **Logical operators in queries.** Reasoning is necessary when logical operators, including connectives, quantifiers, and spatial or temporal operators, appear explicitly in questions. For instance, in the enterprise domain, we may search for a client in northern Europe who has *not* had a high debt balance within the past two years. We need temporal reasoning to tell if a debt was contracted within that period, spatial reasoning to determine whether a company is in Northern Europe, and ordinary reasoning to handle the negation connective *not*. If we are looking for a company that does not have a Spanish subsidiary, we must deal with quantifiers (e.g., *there exists*). In a domain in which a company may have more than one name, or in which one name may apply to several distinct companies, we must deal with the equality relation.
- **Ambiguity in queries.** Natural language questions are highly ambiguous. Even for an apparently simple query, a parser may find hundreds of analyses, most of which can be discarded immediately with some knowledge of the subject domain. For instance, suppose that we ask, in our business enterprise domain, *Find me a client with a debt. It should be high.* A person will know that *"it"* is the debt that should be *high*, not the *client*, because in this context we define *high debts* but not *high clients*. (Note that here clients are companies.) When we ask for a company with decreasing daily sales within the last 20 weeks, we know that *"it"* is the decreasing daily sales that is within the time period, because here that is a temporal concept and a company is not.
- **Rephrasing.** Once the query has been understood and correctly represented, finding an answer need not be a simple matter of lookup. There may be a discrepancy between the way the question is phrased and the way it is represented in our knowledge sources. For instance, if a query says *Fnd a client that owes a lot of money*, the system must know that this means finding a company with a high debt. Similarly, a query may specify the duration of an event, while a knowledge source knows only its start time and end time.
- **Interoperability.** The answer may depend on more than one data source, and the query and the various sources may not adhere to the same representational conventions. If we seek a company that owes more than 500,000 euros and a database has a company with a debt of more than 1 million US dollars in its debt column, the system must know the exchange rate between euros and dollars at the appropriate time. Performing that conversion is part of the reasoning process that relates the query to the information in the database.
- **Extendability.** Question answering relies on a good deal of domain-specific knowledge represented in logical form. Adding new knowledge can be a demanding process in terms of time

and expertise. By incorporating a reasoning capability, we can extend the system to incorporate new knowledge by representing it in declarative natural language sentences. We can translate the sentences into a semantic representation using the same technology by which we translate queries. In this way, a domain expert can extend the system with no knowledge of programming or the representation of knowledge.

In the next section we outline a deductive approach to question answering. We translate natural language queries to a symbolic, logical representation and then employ deductive technology, or theorem proving, to answer the question.

2. Knowledge and Reasoning in Question Answering

Our points extend beyond question answering to more general issues about natural language interaction with computers. For definiteness, we draw many of our examples from our work on SAP QUEST (Condoravdi, Richardson, Sikka, Suenbuel, & Waldinger, 2015), a project initiated and funded by SAP Labs, QUEST originally used PARC natural language technology (Bobrow, Cheslow, Condoravdi, Karttunen, King, Nairn, de Paiva, Price, & Zaenen, 2007) but eventually developed its own; it employs SNARK (Stickel, Waldinger, & Chaudhri, 2004) as its reasoning engine. But we do not intend this paper it to serve as a discussion of the QUEST natural language or reasoning implementation.

In the general approach we propose combines natural language technology with automatic deduction over a knowledge base represented as a logical theory of the subject domain. This theory consists of a vocabulary of logical symbols (constants, functions, and relations) plus a collection of axioms that express the meaning of these symbols. The axioms define the concepts in queries, specify the capabilities of the relevant data resources, and provide the background knowledge necessary to relate them. The vocabulary classifies all entities according to a taxonomy. The logic is sorted, and the sorts (or types) correspond to classes in the taxonomy.

For example, in the enterprise theory, we distinguish between such sorts as companies, nationalities, money, and dates. For each relation in the theory, a declaration specifies the sorts of arguments it accepts. For instance, the `company-has-debt` relation has two arguments, the first of sort `company` and the second of sort `debt`. The declarations of a function symbol specifies the sorts of the arguments it expects and the value it yields. The sort structure is used both to control the parsing of the query and to accelerate the reasoning in searching for answers. Axioms in a sorted language are also more concise and readable.

What follows is a bald outline of the way a query is handled. We indicate how features of a theorem prover come into play.

- **Semantic parsing.** Natural language queries are translated into a formal semantic representation. The semantic representation is translated into a *logical form*, a sentence in the language of mathematical logic that expresses what has been understood from the query in the vocabulary of the subject domain theory. Entities that constitute answers to the query are represented as variables surrounded by existential quantifiers `(exists (?x) ...)`, with variables indicated by question marks. If the question is a sequence of queries, the logical form is a conjunction of the translations of each query. Syntactic entities that have no representation in the theory are pruned away (Richardson, Bobrow, Condoravdi, Waldinger, & Das, 2011). If the question has ambiguities that are unresolved by the translation into logical form, the parsing may produce multiple logical forms.

- **Playback.** The logical form is translated back into pedantic but less ambiguous natural language, to reassure the user the question has been correctly understood. If necessary, the user may rephrase the question to correct misunderstanding. If the semantic parsing has produced more than one logical form, the user may indicate which of them they intend.
- **Proof.** The logical form is regarded as a conjecture and is submitted to a theorem prover to be shown valid. Since the entities to be found are represented by existentially quantified variables, the proof establishes that such entities do indeed exist.
- **Procedural attachment.** Certain relation symbols in the theory are linked to tables in external databases, which indicate what entities satisfy them. Similarly, certain function symbols in the theory are linked to external procedures, which tell us how to compute them. When a linked relation symbol appears in the proof search, the corresponding table in the database is consulted, and existentially quantified variables in the proof search are replaced by concrete entities, which may participate in an answer to the query. When a linked function symbol appears in the proof search, the external procedure may be invoked to compute it, and the resulting value may be provided to the proof search. (Thus, if the expression `(+ 2 2)` appears in the proof search, and the symbol `+` has been linked to a procedure that computes the addition function, the expression may be replaced by the value `4`. Note that we employ a LISP-like syntax for applying an operator to arguments.)
- **Answer extraction.** In the course of the proof, existentially quantified variables, including those indicating desired answers to the query, are replaced by concrete expressions. By keeping track of these replacements, we may extract from the completed proof an answer to the query.
- **Explanation.** The proof of the conjecture contains an explanation of how the answer was obtained and a justification of its validity. A natural language explanation of the answer is constructed from the proof.

In the balance of this paper we illustrate the approach in some detail and show how it helps in resolving some natural language issues. In our selected domain, we are given a database of SAP clients, each of which has a current debt balance in its payments to SAP; the database contains information about each client, such as its number of employees, its country code, debt and so forth. The country code is a two-character encoding of the location of the company; the debt record includes the balance, start date, and a payment history.

The approach, however, is domain independent: For instance, it has been applied previously in the system QUADRI (Bobrow, Condoravdi, Richardson, Waldinger, & Das, 2011), for answering questions about HIV drug therapy; and in the system QUARK (Waldinger et al., 2004), for geographic question answering. The NASA system Amphion (Stickel, Waldinger, Lowry, Pressburger, & Underwood, 1994) has a similar structure, although the queries there were expressed in a graphical notation rather than in a textual language. Some researchers (e.g., Forbus 1984, McDonald 2014, and Nirenburg 2001) employ a comparable approach; they do not use theorem provers, but they use other mechanisms to achieve the same effect.

3. Proving the Logical Form

The query is represented by a conjecture, a logical sentence whose proof is conducted in the same axiomatic theory that the system used to help disambiguate the query. This logical form is decomposed and transformed according to the axioms of the theory, which embody the knowledge extracted from our subject domain experts.

The query requires us to find entities that satisfy a complex relation, which may be a logical combination of many conditions. The entities to be found are represented by variables that are existentially quantified in the logical form; that is, each such variable `?x` is bound by a quantifier `(exists(?x) ...)`. In proving the theorem, the theorem prover replaces these variables by more complex terms; variables in those terms are replaced in turn by other terms, and so on. When the proof is complete, the composition of these replacements indicates an answer that can be extracted from the proof. As we have mentioned, there are typically many proofs for each theorem, each of which may yield a different answer. The theorem prover will seek as many answers as possible within a specified time limit, which it assembles for presentation to the user.

For such applications, a theorem prover must be capable of general reasoning from axioms and dealing with the equality relation. For a commonsense example, suppose a system knows that George is married to Martha and is asked if George is married to Honey. To answer negatively, it must be able to represent the knowledge that the society in question is monogamous and that Martha is not the same person as Honey, and to perform the requisite deductions.

During the proof, the existentially quantified variables of the conjecture are replaced by possibly more complex terms. A *sort mechanism* allows handling of sorts or types; ensuring that a variable cannot be replaced by a term of inconsistent sort. Thus `?company1` will not be replaced by the constant `swiss`. (Here the name of the variable indicates its sort.) Thus the sort mechanism blocks some fruitless branches in the search for a proof.

A system will benefit from accelerated treatment of important concepts, such as space and time. A facility for *procedural attachment* allows external procedures to be invoked, including database access, during the proof search, so that not all reasoning must proceed axiomatically. The facility links selected relation and function symbols in the theory with external procedures, which indicate how to compute the linked symbol. Typically these procedures will access an external database. In the QUEST theory, for example, the symbol `company-record` is linked to an external database of companies, so if the name of a client company is given, the company's other attributes (e.g., size, country code, and debt) can be looked up. When `company-record` occurs in the proof search and a concrete company name is supplied, the other arguments of the relation can be filled in by database lookup, and the proof continues. Other symbols can be linked to other databases, such as those for currency conversion. The procedural-attachment mechanism lets a theorem prover access not only knowledge that resides in its axioms, but also the much larger repository of information stored in external sources.

An *answer extraction* facility allows answers to a query to be obtained from a completed proof. Developed for program synthesis applications (e.g., Green, 1969; Manna & Waldinger, 1980), this facility associates with each sentence in the set an *answer term*, such that if the sentence can be made true, the corresponding instance of the answer term will satisfy the original query. As variables in goal sentences are replaced by terms, the answer term becomes more concrete. When a proof is obtained, the corresponding answer term will be an answer to the query. Typically there are many proofs of a given theorem, each of which may lead to a different answer; these are assembled and presented to the user.

Although general theorem proving is an undecidable problem and the search for a proof can in principle require an unbounded amount of time, the theorems required for question answering are usually not a challenge to contemporary theorem-proving technology.

4. Answering Follow-up Questions

If there is a follow-up question, the system constructs a new logical form; new questions at the end of a query sequence add conditions to the conjecture, which is reproved from scratch. Some of the answers to the original query may not satisfy the new conditions, so in general it cannot salvage old proofs. Logical forms are always transformed so that quantifiers are brought to the top level; new conditions are always added within the scope of the quantifiers for the first question. This offers us a way to deal with otherwise problematic *donkey anaphora*, in which a variable appears to escape the quantifier that should have bound it (King & Lewis, 2017).

For example, suppose our first question is *Show a company with a debt of more than 100 million dollars.* The logical form will include a subcondition

```
(company-has-debt ?company1 ?debt2) .
```

bound by an existential quantifier `(exists (?company1) ...)`. As we will see, a procedural attachment will search a database and come up with a collection of companies with such a high debt. These will be presented to the user, who may decide to discover the nationality of each discovered company by adding the new question *What is the nationality of the company?* to the sequence. The logical form will be amended to have the new condition

```
(company-has-nationality ?company1 ?nationality4) .
```

Note that the variable `?company1` in the new condition is the same as in the original logical form; it has (appropriately) slipped under the same existential quantifier

The theorem will be reproved with the new condition, and the answer extraction mechanism will find the nationality of the returned companies. In this case, the new condition does not force the theorem prover to discard any proofs, because all companies have nationalities; the collection of companies is the same, but the answer includes this information. Had the new condition been "Oh, the company should be Swiss!" the set of answers would have been reduced.

We can now consider in more detail how the above sequence of queries is dealt with in QUEST. We begin with the first query, *Show a company with a debt of more than 100 million dollars.* The conjecture contains the condition:

```
(and
  (company-has-debt ?company1 ?debt2)
  (debt-has-value ?debt2 ?dollar3)
  (> ?dollar3  (* 100 million dollar))) .
```

In the proof, an axiom causes the subcondition `(company-has-debt ?company1 ?debt2)` to be expanded into the conjunction

```
(and
  (company-record  ?company1 ?dollar3 ?country-code5 ...)
  (positive ?dollar3)) .
```

Here `company-record` is a relation symbol with many arguments, one for each attribute of a company we wish to consider. This symbol has a procedural attachment to a table in the SAP database. When it appears in the search space, the variables `?company1`, `?dollar3`, and `?country-code5` are instantiated with concrete company names, balance, and country-codes, respectively, all from the same row. A company is taken to have a debt if its balance of dollars owed is a positive number.

The theorem prover must still establish the remaining two conditions of the logical form,

```
(and
  (debt-has-value ?debt2 ?dollar3)
  (> ?dollar3  (* 100 million dollar))) ,
```

with the variables `?debt2` and `?dollar3` replaced by the actual debt and dollar value for each company. If the dollar value is less than one hundred million, the condition

```
(> ?dollar3
 (* 100 million dollar)))
```

is not satisfied, and that branch of the proof search does not lead to an answer. At the end, we have a collection of companies, each of which has a very large debt.

Now we suppose that the user needs to see the nationality of each answer. The new question in the sequence is *What is the nationality of the company?* The amended question is reparsed, and, as mentioned above, the resulting logical form contains the additional condition

```
(company-has-nationality ?company1 ?nationality4) ,
```

where `?nationality4` now appears in the answer term to be extracted from the proof. An axiom of the theory causes this condition to be replaced by

```
(and
  (company-record  ?company1 ?dollar3 ?country-code5 ... )
  (country-code-vs-nationality ?country-code5 ?nationality4)) .
```

As before, the `company-record` symbol has a procedural attachment that leads the proof search to consult a table in the database. When it retrieves this information about a company, it discovers its country-code. As we shall see, axioms and procedural attachments then let it uncover the corresponding nationality, which it enters into the answer term. The same collection of companies is discovered, but the answer term for each includes its nationality. For example, one answer term will contain the company `SL-Foods`, which has a debt of $105,263,552 and a country code of `CH`, which coincides with the nationality `swiss`.[1]

1. Actually, to preserve the confidentiality of SAP clients, the data we were given was scrambled.

5. Handling Logical Operators

Reasoning is clearly necessary when the query contains explicit logical connectives or quantifiers. For example, the disjunction (*or*) operation lets us answer queries such as *What clients have a high debt or a long-term debt?* Axioms define concepts such as high, low, long-term, and short-term. The query *Does no company have a low high debt?* can be answered immediately in the affirmative, because the definitions of low and high are contradictory; the system does not need to consult the database.

In addition to disjunction, the question *What companies do not have a low debt?* requires reasoning about quantifiers, negation, and equality. The logical form contains the disjunction

```
(or (not (company-has-debt ?company1 (some-debt ?company1))
      (not (low (some-debt ?company1))) .
```

Here `(some-debt ?company1))` is a "skolem term" that denotes an arbitrary debt. We may think of a skolem in a goal as a "spoiler"; it stands for an entity that will make the conjecture false, if anything does. (Skolem functions such as `some-debt` are introduced when a process called *skolemization* removes universal quantifiers,)

The proof relies on the axiom *uniqueness of debt*, which asserts that debts of companies are unique:

```
(implies
 (and
   (company-has-debt ?company ?debt1 ... )
   (company-has-debt ?company ?debt2 ... ))
 (= ?debt1 ?debt2)) .
```

In other words, if a company has two debts, they must be equal. This holds because in this context debt is always the company's current balance.

To prove a disjunction we need prove only one of the disjuncts. Suppose the first disjunct is false; that is

```
(company-has-debt ?company1 (some-debt ?company1))
```

is true. The procedural attachment searches through the database and finds a company, `company1`, that has a high debt `debt-high`, which means it cannot be low. If companies could have more than one debt, that company could also have a low debt and would not satisfy our query. However, in the QUEST domain the *uniqueness* axiom then implies that such a debt must equal `(some-debt company1)`; hence the skolem term must satisfy the second disjunct

```
(not (low (some-debt company1))) .
```

Any system that can carry out the above reasoning must have some facility for dealing with equality and logical operators.

6. Dealing with Ambiguity

One obstacle in question answering is that natural language sentences are ambiguous. If we view the query sequence *Get a client with a debt. It should be high* purely syntactically, the pronoun "It" can refer to either the client or the debt. However, in the QUEST theory, debts can be high but clients cannot. In other words, the logical form

```
 (and (company-has-debt ?company1 ?debt2)
      (high ?debt2)) ,
```

in which "It" refers to ?debt2, is properly sorted. However, the logical form

```
(and (company-has-debt ?company1 ?debt2)
     (high ?company1)) ,
```

in which "It" refers to ?company1, is not properly sorted, because the relation high applies to debts, not companies. Thus, the natural language component will produce the first logical form but not the second one.

Similarly, in the query sequence *Get a client with a debt. It should be Swiss* viewed syntactically, the "It" could be either the company or the debt. However, in the QUEST theory clients can be Swiss but debts have no nationality. As a result, the only logical form produced is

```
(and (company-has-debt ?company1 ?debt2)
     (company-has-nationality ?company1 swiss)) .
```

Finally, the sequence *Get a client with a debt. It should be large* is truly ambiguous: The word "It" can refer to either the company or the debt, because either can be large. Hence we will obtain two logical forms,

```
(and (company-has-debt ?company1 ?debt2)
     (large ?company1)) ,
```

in which "It" coresponds to ?company1, and

```
(and (company-has-debt ?company1 ?debt2)
     (large ?debt2)) ,
```

in which "It" refers to ?debt2. QUEST then gives the user the opportunity to chose between the two alternatives.

Resolving ambiguity plays a large role in facing the *Winograd Challenge Problems*, which Morgenstern and Ortiz (2015) posed to sharpen discussions of the Turing test. We believe introducing deductive inference into natural language understanding can settle many of the obstacles that arise in these problems.

7. Interoperability and Reasoning about Time and Space

Many questions require temporal and spatial reasoning. For example, in the query *Show companies with a high debt within the last two years.*, the logical form will refer to a temporal interval that represents the last two years. This has a duration of two years and ends at the time point `(now)`, when the question is being asked. The duration of a time interval is defined by the axiom

```
(=
 (duration
  (make-interval
   ?time-point1 ?time-point2))
 (minus-time ?time-point2 ?time-point1)) .
```

In other words, the duration of a time interval is the difference between its end points. The theorem prover will then seek companies whose debt lies within this two year interval. The implementation of the Allen (1983) interval calculus in QUEST's reasoning engine SNARK can reason about relations between time intervals, such as `after` and `within`.

Similarly, an implementation of the regional connection calculus (Cohn & Hazarika, 2001) lets SNARK (or other theorem provers) reason efficiently about spatial regions. For instance, if we are seeking southern European companies with decreasing sales, we may use spatial reasoning to rule out companies that are not within the region of southern Europe.

When handling a query using multiple information sources, the system must contend with the fact that the user may choose terminology that differs that of these sources. Furthermore, since these were not necessarily designed to work together, they may themselves have different notations and representations. For example, in the query *What is a Swiss company that has a debt of more than 10 million euros within the last 100 weeks?* the user asks for a company whose nationality is Swiss. However, as we have seen, the database of companies represents the location of a company by a two-letter code, such as "CH" for Switzerland. The system invokes procedural attachments to two external geographical tables, one that relates the nationality `swiss` to the country `switzerland`, and another that relates `switzerland` to the country code "CH". Similary, it uses an online currency table to relate the *euros* in the query to the US dollars used in the database on a given date. The currency table is extended every day to include the new rates of exchange. The system looks for the exchange rate that held when the question was asked. Similarly, we use a procedural attachment to date arithmetic to relate the *weeks* of the query with the dates used by the database; the database does not store durations explicitly.

8. Extending the Subject Domain Theory

Many question-answering systems, including QUEST and the medical system QUADRI, operate on a fixed subject domain theory. But perhaps the most persuasive argument for using a logical representation is the ease of extending the theory by adding new declarative knowledge. This can be provided by a domain expert who needs no knowledge of the theory's logical notation. The same technology we use to translate natural language queries into logical form can translate declarative assertions. Instead of being treated as a conjecture, the resulting logical sentence becomes a new axiom of the theory. Deductive technology will detect if this axiom contradicts what is already known. The domain expert can test the adequacy of the new axiom by asking test questions. If the system fails to answer questions that depend on the new information, the assertion may need to be strengthened or new assertions added.

For instance, in QUEST's enterprise theory the notion of a *high* debt is defined to be any debt whose value is greater than one million dollars. But it makes little sense to assume this concept is built in; different users may have different notions of a high debt, and a single user may have different notions in different contexts.

If the system accepts new declarative knowledge, we can provide the English sentence *A debt is high if and only if it has a value greater than a million dollars.* This translates into the logical form

```
(<=>
   (high ?debt)
   (exists (?dollar)
      (and (debt-has-value ?debt ?dollar)
           (> ?dollar (* million dollar))))) .
```

This would be consistent with the theory unless someone had previously introduced a different notion of high debt. In that case, the system would need to resolve the inconsistency be resolved by extra-logical means, perhaps by intervention of a human moderator. This means that the theory could grow rapidly, with many domain experts working independently to add content and a moderator dealing with conflicts as they arise.

9. Concluding Remarks

Many problems that arise in natural language access to computer systems are alleviated if the conversational interface understands the subject domain. An effective approach is via a reasoning system equipped with an appropriate axiomatic subject-domain theory and procedural attachment to external knowledge sources. Among the issues that reasoning can address are

- Questions can include explicit logical operators, such as connectives, quantifiers, and spatio-temporal relations.
- There are many ambiguities in natural language queries.
- Queries may be phrased in a vocabulary that differs from that adopted by the available knowledge resources.
- Sources not intended to work together may have different representational conventions.
- Follow-on questions may include references to previously mentioned quantified entities (*donkey anaphora*).
- It may require domain knowledge to identify the entities referred to in follow-on questions (*Turing challenge questions*).
- A system may need to be extended by a domain expert who is ignorant of programming or logic.

We have phrased these issues in terms of question-answering applications, but they clearly come into play whenever we use natural language to communicate with computer systems. And we have argued that any intelligent natural language interface would have reasoning capabilities comparable to a theorem prover.

Despite the benefits of logical reasoning, existing deductive question-answering systems show only rudimentary ability and, for the most part, remain far from practical application. In the rest of this section we hint at some areas in which further research would be fruitful.

The system Amphion (Stickel, Waldinger, Lowry, Pressburger, & Underwood, 1994) used program synthesis technology to compose software from NASA routines to access data from interplanetary missions; it was used to generate efficient software for analyzing photographs from the Cassini mission. But, overall, little attention has been paid to applying deductive methods to achieve efficient database access. Ideally, the system should use the capabilities of database languages like SQL to search for answers efficiently.

Another issue is that existing systems do not fail gracefully. When faced with a question it cannot answer, a system should tell us whether the desired answer does not exist, the answer is too hard to find, or the question is outside of its capabilities. The system should also be able to ask the user questions during its search for a proof, without becoming intrusive.

The knowledge bases for question answering systems tend to be ad hoc, developed for particular applications, and with little uniformity among them. Ontologies such as Cyc (Lenat, 1995) or SUMO (Reed & Pease, 2015) contain axiomatic definitions for all the concepts in the online dictionary WordNet (Miller, 1995) and might be a foundation for a broader question answering system, not restricted to a single domain. We cannot expect, however, that any of these ontologies contain all the axioms we need for a useful system. Building a general axiomatic knowledge base is a daunting undertaking, much like building a dictionary or thesaurus, or even a cathedral. But that doesn't mean that we shouldn't do it.

Acknowledgements

We dedicate this paper to the memory of our colleague, collaborator, and friend Danny Bobrow. We thank Vishal Sikka for initiating and providing direction and suggestions for the SAP QUEST project, SAP Labs for its support, and many of its members for lending their technical expertise and providing access to data. We are indebted to Pat Langley for careful reading and insightful suggestions and to Steve Greidinger for many relevant discussions.

References

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM*, *26*, 832–843.

Bobrow, D. G., Cheslow, B., Condoravdi, C., Karttunen, L., King, T. H., Nairn, R., de Paiva, V., Price, C., & Zaenen, A. (2007). PARC's bridge and question answering system. *Proceedings of the GEAF 2007 Workshop*. Stanford, CA: CSLI Publications.

Bobrow, D. G., Condoravdi, C., Richardson, K., Waldinger, R., & Das, A. (2011). Deducing answers to English questions from structured data. *Proceedings of the Sixteenth International Conference on Intelligent User Interfaces* (pp. 299–302). Palo Alto, CA: ACM.

Cohn, A. G., & Hazarika, S. M. (2001). Qualitative spatial representation and reasoning: An overview. *Fundamenta Informatica*, *46*, 1–29.

Condoravdi, C., Richardson, K., Sikka, V., Suenbuel, A., & Waldinger, R. (2015). Natural language access to data: It takes common sense! *Proceedings of the Twelfth International Symposium on Logical Formalizations of Commonsense Reasoning*. Stanford, CA: AAAI Press.

Forbus, K. D. (1984). Qualitative process theory. *Artificial Intelligence*, *24*, 85–168.

Green, C. (1969). Application of theorem proving to problem solving. *Proceedings of the First International Joint Conference on Artificial Intelligence* (pp. 219–239). Washington, DC: Morgan Kaufmann.

Kaplan, R. (2013). The conversational user interface. *Proceedings of the Nineteenth Nordic Conference of Computational Linguistics* (p. 1). Oslo, Norway: Linkoping University Electronic Press.

King, J. C., & Lewis, K. S. (2017). Anaphora. In E. N. Zalta (Ed.), *The Stanford encyclopedia of philosophy*. From `https://plato.stanford.edu/archives/sum2017/entries/anaphora/`.

Lenat, D. B. (1995). CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, *38*, 32–38.

Manna, Z., & Waldinger, R. (1980). A deductive approach to program synthesis. *ACM Transactions on Programming Langanges and Systems*, *2*, 90–121.

McDonald, D., & Pustejovsky, J. (2014). Representing inferences and their lexicalization. *Advances in Cognitive Systems*, *3*, 143–162.

Miller, G. A. (1995). WordNet: A lexical database for English. *Communications of the ACM*, *38*, 39–41.

Morgenstern, L., & Ortiz, C. L. (2015). The Winograd schema challenge: Evaluating progress in commonsense reasoning. *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence* (pp. 4024–4025). Austin, TX: AAAI Press.

Nirenburg, S., & Raskin, V. (2001). Ontological semantics, formal ontology, and ambiguity. *Proceedings of the International Conference on Formal Ontology in Information Systems* (pp. 151–161). Ogunquit, Maine: ACM.

Reed, S. K., & Pease, A. (2015). A framework for constructing cognition ontologies using WordNet, FrameNet, and SUMO. *Cognitive Systems Research*, *33*, 122–144.

Richardson, K., Bobrow, D. G., Condoravdi, C., Waldinger, R. J., & Das, A. (2011). English access to structured data. *Proceedings of the Fifth IEEE International Conference on Semantic Computing* (pp. 13–20). Palo Alto, CA: IEEE Computer Society.

Stickel, M., Waldinger, R., & Chaudhri, V. (2004). *A guide to SNARK*. Technical report, SRI International, Menlo Park, CA.

Stickel, M. E., Waldinger, R. J., Lowry, M. R., Pressburger, T., & Underwood, I. (1994). Deductive composition of astronomical software from subroutine libraries. *Proceedings of the Twelfth International Conference on Automated Deduction* (pp. 341–355). Nancy, France: Springer.

Waldinger, R. J., et al. (2004). Deductive question answering from multiple resources. In M. Maybury (Ed.), *New directions in question answering*, 253–262. Palo Alto, CA: AAAI Press.