Deduction for Natural Language Access to Data

Cleo Condoravdi[1], Kyle Richardson[2], Vishal Sikka[3]†, Asuman Suenbuel[3] and Richard Waldinger[4*]

[1] Department of Linguistics, Stanford University
Stanford, California, U.S.A.
cleoc@stanford.edu
[2] Institute for Natural Language Processing, University of Stuttgart
Stuttgart, Germany
kyle@ims.uni-stuttgart.de
[3] Office of the CTO, SAP Labs
Palo Alto, California, U.S.A.
asuman.suenbuel@sap.com
†now at Infosys
[4] Artificial Intelligence Center, SRI International
Menlo Park, California, U.S.A
waldinger@ai.sri.com

1 Introduction

We outline a general approach to automated natural-language question answering that uses first-order logic and automated deduction. Our interest is in answering queries over structured data resources. We are concerned with queries whose answer is not stored directly in a single database but rather must be deduced and computed from information provided by a number of resources, which may not have been designed to work together. While the obstacles to understanding natural language queries are formidable, we simplify the problem by limiting ourselves to a well-understood subject domain and a known set of data resources. Using domain knowledge, queries in natural language are mapped to a logical representation and interpreted using an automated reasoner over a logical theory with semantic links to target knowledge sources.

Examples are drawn from a prototype system called Quest, which is being developed for a business enterprise question-answering application. Users of such a system can query complex databases without needing to know the structure of the target knowledge sources or to write programs in a database query language.

2 Motivation

Automated question answering has long been a major research topic in artificial intelligence. In particular, there is a large literature on natural language interfaces to databases, which focuses on using ordinary language to query database content (for a review, see [1, 11]). Given a natural language question, the goal of a question-answering system is to find an answer using

*From NLSR2014— 1st International Workshop on Natural Language Services for Reasoners, Vienna, Austria. https://www.cisuc.uc.pt/ckfinder/userfiles/files/cisucTR201402.pdf

a combination of *language processing* and *reasoning*. In the automated reasoning community, the deductive aspects of question answering on structured data have long been understood (for discussion see [19]), but theorem-proving tools not have not been widely used in recent mainstream natural-language processing tasks (some exceptions: [4, 5, 16]), let alone for question answering. In general terms, answering a question in this framework reduces to the problem of proving a theorem about the existence of entities that satisfy the conditions of the question, which are expressed in a formal logic. The role of the language-processing component is therefore to produce such a formal representation.

In natural language processing proper, work on interfaces to databases has traditionally focused on transforming a natural language question to an expression in a formal database query language, such as SQL [8, 9] or SPARQL [18]. The expression is then used by a standard database management system to access the target knowledge source and retrieve the desired answer (which in essence solves the reasoning task). Usually, the query is applied to a single (fixed) database with a predefined semantics. More recently, research has centered around learning the transformation from language to formal queries using various forms of data supervision [20, 12, 9, 2]. In this recent work especially, there has been scant attention to using richer formal representation languages, and the available datasets continue to rely on formal database query languages for representing query semantics. Such approaches tend to ignore complex semantic phenomena (e.g. quantifiers scope, temporal and spatial reasoning), despite the appearance of such constructs in benchmark datasets (see [6] for discussion).

A major drawback to these approaches is that a simple natural language query may be complex when represented in a language like SQL, due to limitations in the expressive power of such languages [1]. Furthermore, the way the user naturally formulates a query may be quite different from the way the database designer chooses to represent information that answers the question. It may require world knowledge to translate the user's formulation of the query into the form preferred by the database. Once the query is reduced to an SQL program, it is hard to apply world knowledge to transform it. The problem is compounded when the answer depends on information obtained from several heterogenous sources, or involves information that changes over time. Each source may have adopted its own representational conventions, and, when sources are to be composed, the information provided by one source may not be in the form required by another.

Another obstacle is that natural language queries are hard to parse and interpret, especially when the target representations are richly defined and aim to model deeper aspects of linguistic meaning. Using a well-defined subject domain and set of knowledge resources, one can use this knowledge (e.g. about the entities and types of relations between entities) to help guide the parsing process, an idea which has gained some traction in recent work in NLP on semantic parsing (e.g. [2]). Using a reasoner further simplifies the task of interpreting the language, since assumptions can be associated with symbols in the formal representation rather than explicitly encoded into the language analysis directly.

In the following sections, we outline our approach and show examples from a small prototype question-answering system called Quest, which is being built for the enterprise software company SAP. We focus largely on the representation side of the problem and the general mechanics of how the reasoning works and interacts with the natural language analysis.

User Query (a): (computer naive user)

Pull up a list of large companies that have a debt of more than 5 million euros. In particular, I am interested in the nationalities of these clients.

Sample Databases (b)

Accounting Database 1

company	debt	location	time	employed
SL Foods	$10526...	CH	9.2007	5000
Maria Morgan	$5000	US	8.2006	150

Name Database 2

ID	Name
CH	Switzerland
US	United States

Currency Conversion Database 3

USD(x)=	EU(x)=	CH Franc(x)=	RU Ruble(x)=
x	x * 0.74	x * 0.91	x * 35.81

Figure 1: Illustration of the problem setting: a natural language query (a), along with a set of knowledge resources, or databases (b). See text for full description.

3 Problem Setting

We are interested in questions whose answers are spread over different knowledge sources. Figure 1 provides an illustration of the problem, and includes an English query (a) and a set of (simplified) databases (b). The question is from the domain of *dunning* (or debt collection) being investigated in the Quest project. The colors show the rough correspondence between individual phrases in the English query and values in multiple databases. Some of these correspondences are direct; for example the word *'company'* is directly related to the `company` field in Database 1. Other correspondences are less direct; for example the modifier *'large'* is a *qualitative* notion that relates to the size of the particular item being modified; for companies, this is taken here to mean the number of employees. There are also more abstract relations between the words (e.g. that the *'company'* has the *'debt* as one of its properties), which in general are less direct and require further specification or computation.

Our approach is based on logic, so the meaning of a query is expressed symbolically as a logical formula and answered by proving a theorem in the logic about the existence of entities that satisfy the formula. Figure 2a informally shows how the query in Figure 1 is represented in logic. The starred symbols and indentation show the logical symbols (in this case, quantifiers) and their scope, the words in bold are the types of entities, and the underlined symbols are the relations. These symbols are further defined in a subject domain theory (sketched in Figure 2b), which consists of *axioms*, which define relations and other background assumptions and knowledge (e.g. the meaning of 'large'); and *procedural attachments*, which specify how to link symbols with procedures, such as look-up procedures in a structured database. A semantic parser is used to translate the English queries to first-order logic, and an automated reasoner is used to find proofs of the query, each of which indicates an answer.

When a formula with a linked symbol appears in the search for a proof, the relevant database is consulted, and information from the database is imported into the proof, just as if it had been

represented axiomatically. For example, the procedural attachment in Figure 2b specifies that an existential variable of sort `company` in the linked relation `company-record` can be replaced by an entry from the `company` field in Database 1. In addition, multiple databases might need be consulted in order to solve the conjecture; e.g. knowing if the debt value (in dollars) in Database 1 is above the specified value (in euros) requires doing (a real-time) currency conversion using Database 3, specified in the `euro` procedural attachment. Another database relates the two-character country code `CH` to the name of the country Switzerland. Procedural attachment allows a set of databases to act as a virtual extension of the subject domain theory. Qualitative relations are further defined by ordinary axioms and can be crafted to fit a particular domain or refined by asking follow-up questions (e.g. defining what a *large* company means).

The labor in our system therefore is divided between producing first-order logic representations from natural language questions (the *semantic parsing* component) and using these representations to find answers by proving theorems (the *reasoning* component). In the sections below, we describe the details of each component in the Quest system and show sample representations.

4 Natural Language Query Processing

The first component of our system translates English questions to a formal meaning representation, a task traditionally known as *semantic parsing*. The goal of the representation is to encode the domain-specific relations that hold, in addition to logical operators (e.g. quantifiers, negations) and their scope. This mapping is done by detecting patterns within the surface syntax, patterns that are defined by a domain expert and based largely on the background subject domain. These rules are used to rewrite *interpretable* syntactic patterns to flat semantic clauses. Below is a fragment of a question (1), along with sample tree mapping rules (a-d) and syntactic fragments (i) that match these patterns.

1. `.. company with a debt of 100 million euros... show the nationality...`

   (a) `(COMPANY,(with(DEBT)`$_{\texttt{NP}}$`)`$_{\texttt{PP}}$`)`$_{\texttt{NP}}$ `=> (COMPANY-HAS-DEBT company debt)`
   (i) [company$_{\mathbf{COMPANY}}$ [with$_{\text{P}}$ [debt$_{\mathbf{DEBT}}$]$_{\text{NP}}$]$_{\text{PP}}$]$_{\text{NP}}$

   (b) `(DEBT,(..?(MONEY_AMOUNT)`$_{\texttt{NP}}$`)`$_{\texttt{..?}}$`)`$_{\texttt{?}}$ `=> (DEBT-HAS-VALUE DEBT MONEY_AMOUNT)`
   (i) [debt$_{\mathbf{DEBT}}$ [of$_{\text{P}}$ [euros$_{\text{MONEY_AMOUNT}}$]$_{\text{NP}}$]$_{\text{PP}}$]$_{\text{NP}}$

   (c) `(NUMBER,(..?(MONEY_AMOUNT)`$_{\texttt{NP}}$`)`$_{\texttt{..?}}$`)`$_{\texttt{NP}}$ `=>(HAS-MONEY-VALUE MONEY_AMOUNT NUMBER)`
   (i) [100 million$_{\mathbf{NUMBER}}$ euros$_{\mathbf{MONEY_AMOUNT}}$]$_{\text{NP}}$

   (d) `(DEFINITE_MARKER,NATIONALITY)`$_{\texttt{NP}}$ `=>(DEFINITE NATIONALITY)`
   (i) [the$_{\mathbf{DEFINITE_MARKER}}$ nationality$_{\mathbf{NATIONALITY}}$]$_{\text{NP}}$

This transduction procedure is implemented in a prototype parser called SAPL, which is built on top of components from the SAP Inxight text analytics engine. At an early stage of processing, the text analytics engine is used to perform entity (bold subscripts) and part-of-speech tagging. SAPL then uses a bottom-up parsing strategy and incrementally matches rules to syntactic patterns annotated with their entity types. For example, in 1a, the words *'company'* and *'debt* are tagged as `COMPANY` and `DEBT` respectively and detected in an NP pattern. The mapping rule specifies that there is a `COMPANY-HAS-DEBT` relation, which holds between a `COMPANY` and `DEBT` in a subordinate PP. Since the pattern matches, it is rewritten into

Conjecture (a):

```
there's some* company
    that company is large
    there's some* debt
        the company has that debt
        there's some* debt amount
            the debt amount is 5 million euros
            the debt is more than the debt amount
            there's some* nationality
            there's some* client
                the client has that nationality
                the client is equal to the company
```

Subject Domain Theory (b):

```
;;axioms
  COMPANY has DEBT <=> exists company-record(?company,?debt,..)
                          AND ?debt > $0.00
  COMPANY is large <=> exists company-record(?company,..,?size)
                          AND ?size > some Y
  X more than Y euros <=> euro(x) > Y

;;procedural attachments
  company-record(?company,?debt,..) :
        for some row in Database 1
         replace ?company with value in companies field
         replace ?debt with value in debt field (DB1)
         replace ?size with value in employed field
  euro(?value) : replace ?value with result of EU(?value)
```

Figure 2: A sketch of the interpretation process for Figure 1, including a logic-like representation of the query (or the theorem to be proved) and a informal description of the associated knowledge. The colored objects in the knowledge point to the database values shown in Figure 1, and the underlined words show the relations.

the corresponding semantic representation. In general, rules can be written at varying levels of detail; for example, the regex-like operator *..?* (zero or one of anything) under-specifies details about the associated syntax. Other rules are linguistic in nature; for example, (1d) is a mapping rule for marking definite descriptions, which is useful for later reasoning about equality and anaphora. Additional heuristic rules for handling scope and other logical operator are also added.

Technically, the SAPL parser builds on earlier work that uses the PARC XLE system [15], and MT-style transfer rules for mapping syntactic structures to semantic clauses [7]. Rewrite rules of this kind are commonplace in computational semantics and can range from rather simple pattern or template matching ([18]) to the use of formal systems such as tree transducers [14]. In the latter case, recent work on data-driven semantic parsing has focused on learning these transformation rules from data [12], often by employing techniques and formal models from statistical machine translation [20] and statistical parsing [2]. Given enough data, such

```
((input Show a company with a debt of more than 100 million dollars.<end>
        What is the nationality of the client ?)
  (top_level company_5 0)
  (top_level company_6 0)
  (definite company_6)
  (definite nationality_8)
  (desired_answer company_5)
  (quant exists debt_3 sort debt)
  (quant exists company_5 sort company)
  (quant exists nationality_8 sort nationality)
  (quant exists company_6 sort company)
  (quant (complex_num more than 100000000 dollar) dollar_4 sort money_amount)
  (exists_group ex_grp_45 (debt_3 nationality_8))
  (scopes_over nscope company_5 ex_grp_45)
  (in nscope debt_3 (company-has-debt company_5 debt_3))
  (in nscope dollar_4 (debt-has-money-value debt_3 dollar_4))
  (in nscope nationality_8 (nationality-of company_6 nationality_8)))
```

Figure 3: Output of the SAPL semantic parser. Relations are expressed in a flat clausal form. See text for more details, as well as a description of the mapping rules in (1) used for producing these representations.

learning methods could be used to learn the patterns shown in (1), cutting out the need for writing rules.

Most of the mapping rules encode information directly from the subject domain theory used by the automated reasoner, and provides an overall conceptualization of the target domain. This knowledge guides the parsing and interpretation of the language, in the sense that constraints on the types of semantic relations that hold make it easy to eliminate certain structural ambiguities. For example, the PP *'of 100 million dollars'* syntactically can attach to either *'company'* or *'debt'*, but only the second attachment is allowed by our knowledge of the domain (similarly for *'with debt'* attaching to *'company*). The idea of using knowledge like this has been applied at a larger scale for semantic parsing, e.g. in [2], which uses knowledge from the large FreeBase ontology [3] (as well as answers) for representations for open-domain questions.

The full output of SAPL applied to a particular example is shown in Figure 3. In this case, the query has multiple sentences, and the second question is a follow-up on the first. Similarly to the representations used in [7], the entities in the patterns are expressed as skolem terms, and the relations (including scopal relations) are expressed in a flat clausal form, which mirror the local syntactic structures that the patterns are rewritten from. This representation is then unpacked to have a more conventional first-order-logic form (shown in Figure 4), which is passed to the automatic reasoner. The reasoning also fills in more details missing from the analysis, e.g. it can deduce the equality of *'company'* and *'client'* given that the clue that *'client'* is definite in the representation.

5 Reasoning

SAPL produces an intermediate semantic representation. This representation provides a description of the query, including the logical relations and the connectives and quantifiers that relate them. A component called SAPL-to-SNARK converts this into logical form; this form is

regarded as a conjecture that is submitted to the theorem prover SNARK [17].

The proof is conducted within the subject domain theory, whose axioms embody the knowledge of our subject domain experts. They define the meaning of the concepts in the query, specify the capabilities of the data sources, and provide the background knowledge necessary to relate them. The logical form is decomposed and transformed according to these axioms.

In proving the logical form, SNARK uses machine-oriented inference rules, such as resolution (for general-purpose reasoning) and paramodulation (for reasoning about equality). It has special facilities for reasoning about space and time. Its procedural-attachment mechanism allows us to link the axiomatic theory to structured data and its answer-extraction mechanism constructs an answer to the query from the completed proof. Although SNARK is ideally suited, QUEST could use any theorem prover that has comparable facilities.

SNARK is a refutation procedure; it looks for inconsistencies in sets of logical sentences. In trying to prove a new conjecture, SNARK will negate the conjecture, add it to the set of axioms in the theory, and try to show that the resulting set is inconsistent. For this purpose, it will apply inference rules to the negated conjecture and the axioms. The resulting inferred sentences are added to the set of sentences. Inference rules are than applied to the new sentences and the axioms. The process continues until a contradiction is obtained. Because the inference rules are all logically sound, all the sentences obtained follow from the negated conjecture and the axioms. Assuming that the axioms are all true, this shows that the negated conjecture must be false, i.e., the conjecture itself must be true.

The decision in which order the axioms are considered is controlled by a set of heuristic principles, which are built into SNARK. We supply orderings and weights on the symbols of the theory; this information focuses the attention of the system. For example, subgoals of lower weight are given higher priority than sentences of higher weight. Within a given logical sentence, relation symbols that are higher in the ordering are attended to before relation symbols that are lower.

Since our theorem proving is undecidable, we give SNARK a time limit (currently six seconds); if the theorem is not proved in that time, we cannot give an answer. None of the examples in this paper required more than that time; most were done in less than a second.

The answer-extraction mechanism was originated for program synthesis applications. The query requires us to find entities that satisfy a complex relationship, which may have many conditions. The entities to be found are represented by variables that are existentially quantified in the logical form. 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. Typically, there are many proofs for each theorem; each proof may yield a different answer. QUEST assembles all these answers for presentation to the user.

As we have seen, using the procedural-attachment mechanism, symbols in the theory may be linked to tables in the database (or indeed to arbitrary procedures). The database can be accessed as the proof is under way. When a formula with a linked symbol appears in the search for a proof, the relevant database is consulted, and information from the database is imported into the proof, just as if it had been represented axiomatically. For example, a procedural attachment might specify that an existentially quantified variable `company` can be replaced by to an entry from the `company` field in the Company Record database, and the variable `debt` can be replaced by an entry from the `debt` field). Other symbols can be linked to other databases, e.g. the Currency Conversion database. Mathematical functions and relations are linked directly to arithmetic procedures, so that these operations do not need to be performed

by the theorem prover.

This means that the theorem prover can access not only the knowledge that resides in its axioms, but also the contents of available data resources. The procedural-attachment mechanism also allows us to mimic some higher-order-logic inference steps.

If, after the user has viewed the answers, there is a follow-up question, the logical form is modified to incorporate the new conditions, and the theorem is proved again.

6 Example

As a specific example, we consider the query *Show a company with a debt of more than 100 million dollars. What is the nationality of the client?* Although for a human being this is straightforward, for the parser it is highly ambiguous. A person knows that the word "with" pertains to the company, because companies may have debt. The parser, however, also considers the possibility that a debt could be a way of showing something, like *pictures* in *Show your ideas with pictures*. By using subject domain knowledge, the parser concludes that a debt cannot be a way of showing a company and so "with" must apply to the company itself.

Also, the subject domain knowledge tells us that in this context the word "client" is synonymous with "company." Hence, "client" in the second sentence is the same thing as "company" in the first. Consequently, we replace all occurrences of "client" with "company." The full semantic representation for this query is shown in Figure 3.

SAPL introduces a quantified variable for each named entity in the query and indicates what kind of quantifier is to be used and what scoping is to be employed. In our example, all the quantifiers are existential, because they are entities to be found. The rows in (1) indicate that the variables `company_5` and `debt_3` are to be existentially quantified in the logical form, and these variables have sorts `company` and `debt`, respectively.

1.
```
(quant exists company_5 sort company)
(quant exists debt_3 sort debt)
```

SAPL introduces appropriate semantic relations between these variables. For instance, in the representation, the row in (2) indicates that the variable `company_5` is related to the variable `debt_3` by the relation `company-has-debt`. Furthermore the notation requires that this relation be within the scope of the variable `debt_3`.

2.
```
(in nscope debt_3 (company-has-debt company_5 debt_3))
```

The logical form obtained from this representation is shown in Figure 3. In other words, we are seeking a company that has a debt whose value is more than a hundred million dollars; we are also to find the nationality of this company.

7 Proof

As part of the theorem proving process, the logical form is negated, its quantifiers are removed by skolemization, and its variables are renamed. The expression is expanded according to the following axiom `expand-company-has-debt`:

```
(exists ((company_5 :sort company))
 (exists ((nationality_8 :sort nationality))
  (exists ((debt_3 :sort debt))
   (exists ((dollar_4 :sort money_amount))
     (and (is-debt debt_3)
          (is-company company_5)
          (nationality-of company_5 nationality_8)
          (company-has-debt company_5 debt_3)
          (more-than dollar_4 (* 100000000 dollar))
          (debt-has-money-value debt_3 dollar_4)))))))
:answer
(ans company_5 nationality_8 debt_3 dollar_4)
```

Figure 4: Reformatted representation used by the theorem-prover for query in Figure 3.

```
(==>
  (company-has-debt ?c.company ?d.debt)
  (&
    (company-record ?c.company ?d.debt ?dso.dso ?l.location
                    ?size.number (now))
    (> ?d.debt 0)))
```

Here `?c.company` is a variable of sort `company`; it can be replaced only by terms of sort `company` (or subsorts of this sort) during the proof. The relation `company-record` holds if its arguments are linked to a row in the database of companies; `?dso.dso` is a detailed record of a company's debt; `?size.number` is the size of the company; `(now)` is the current date. Note that the axiom requires that, for a company to be said to have a debt, its entry in the debt column of the database must be nonzero. A company with zero debt (or negative debt) is not thought of as having a debt at all.

Applying this axiom to the negated logical form results in a sentence that contains the negation of `(company-record ?c.company ... (now))`. The symbol `company-record` has a procedural attachment which can access rows in the database as the proof is underway. The variables in the expression will be replaced by concrete terms from the database. For instance, one such row refers to the Swiss company `SL Foods Inc.`, which has a debt of 105,263,552 dollars. (The data, based on actual SAP data, has been garbled to protect the privacy of clients.) Consequently a new sentence will be obtained in which `?c.company` has been replaced by `SL Foods Inc.`, `?d.debt` has been replaced by a debt of 105,263,552 dollars, and `?l.location` has been replaced by a two-letter code for Switzerland. The new sentence will contain the condition that the debt is greater than 100 million dollars.

Other sentences will correspond to other companies in the database, with other debts and nationalities. These turn out to have debts of less than 100 million dollars. The only proof, then, will yield the answer SL Foods Inc. of Swiss nationality.

8 Other Examples

In this section we mention some examples that illustrate other capabilities.

What clients have a high debt or a long-term debt? Concepts such as high and low, and

long-term and short-term debts are defined by axioms. Having a theorem prover allows us to deal with questions with logical operators, such as "or", and quantifiers.

Temporal reasoning allows us to determine the duration of a debt. For example, in *Show clients with a high debt within the last two years*, the logical form will posit the existence of a temporal interval that represents the notion of the last two years. The duration of this interval is two years and its finish-point is the current time "now." Relevant axioms for these concepts include the axiom `duration-of-time-interval`,

```
(=
   (duration (make-interval ?t1.time-point ?t2.time-point))
   (minus-time ?t2.time-point ?t1.time-point)
```

and the axiom `last-of-time-interval`,

```
(iff
   (last ?t.time-interval)
   (= ,now (finish-time ?t.time-interval).
```

(In other words, the duration of a time interval is the difference between its end-points, and an interval is a "last" interval if its finish-time is "now." Debts are themselves temporal intervals, and the specified debt is restricted in the logical form to be within the posited two-year temporal interval.

For *What companies do not have a low debt?* the following axiom `uniqueness-of-debt` implies that the debt a company owes SAP at a given time is unique.

```
[(company-record ?c.company ?d1.debt ?dso1.dso ?l1.location ?size1.number
   ?t.time) &
 (company-record ?c.company ?d2.debt ?dso2.dso ?l2.location ?size2.number
   ?t.time)] =>
 (= ?d1.debt ?d2.debt)
```

Thus, when Quest finds a company with one debt that is not low, it reasons that the company could not have a low debt.

In *Every company does not have a low high debt,* Quest can answer immediately in the affirmative because the definitions of low and high are contradictory; it does not need to consult the database.

Can you show me Swiss companies with a debt? Show only those that owe more than 50 million euros. Quest knows that debt means the same as money owed. Currency conversion rates for 35 currencies for the date in question are obtained from the European Central Bank [`www.ecb.europa.eu`]. Two-letter country codes in the SAP database are associated with country names in an ISO database [`www.iso.org/iso/country_codes.htm`]; another table associates country names (e.g. Switzerland) with nationalities (e.g. Swiss).

What country had the lowest debt within the last two years? Show only a client who owes more than 50 dollars. What is the nationality of the company? The database gives the date each debt was contracted. Quest uses temporal reasoning to make sure that date is within two years of "now," the date the question is being asked. Companies that owe less than 50 dollars are not accepted as answers. Of all the remaining proofs, Quest will select the one yielding the lowest debt as the answer.

9 Future Work and Summary

There is much to do in the business enterprise domain that we have not yet approached. While we can find the largest or smallest item from a set of answers, we cannot find their sum or average. We don't do amalgamation questions such as *What percentage of companies have a high debt?* which are quite useful.

While we find answers in a few seconds, we could greatly improve the efficiency of Quest by making more use of the facilities of a database query system such as SQL. We would need to prove a more complex theorem, which would result in the synthesis of a more complex but faster database query.

We have found that knowledge and reasoning in a specific subject domain can greatly assist in natural language processing. However, the task of developing an axiomatic subject domain theory for the subject of interest can be very labor intensive. In the next phase of our research, we plan to use natural language to assist in the formation or extension of an axiomatic knowledge base.

Quest translates a query into a logical form that captures its intended meaning. We can apply the same techniques to translate declarative sentences into axioms. Supplying this technology to a subject domain expert allows an axiomatic theory to be constructed or extended, without the expert's needing to know any logic. Playing back the axiom into English, as Quest does with queries, allows us to ensure that the correct meaning of the original English sentence has been captured. Theorem proving techniques can also be brought to bear to detect if any inconsistencies have been introduced into the theory. The domain expert may also be alerted to any surprising consequence of the new axioms. For instance, the system may warn the expert if a new axiom simplifies or subsumes a great many other axioms in the theory; this can be a sign that the axiom is stronger than expected.

Rather than starting to develop axioms from scratch, we can use an existing ontology, such as SUMO [13] or Cyc [10]. SUMO gives us some 80,000 axioms for common concepts. Every word in the online thesaurus Wordnet maps into some SUMO concept. While we usually need to introduce more axioms, to fill in gaps in the understanding of our subject domain, we can use a standard ontology as a starting point. Adapting a standardized representation allows us to take advantage of other efforts in axiomatizing the same subjects.

We can also partially automate the formation of new procedural attachments. A domain expert who is also familiar with the structure of a relevant database can assist in linking a new relation symbol with a database table. Each column in the database (e.g. size, location) must be associated with a concept in the ontology. We may then automatically generate code that accesses that table when the linked relation symbol occurs in the search for a proof. The domain expert may also provide declarative sentences that relate common relations in the subject domain with database concepts. This is an iterative process—when the system fails to answer a question, the domain expert may need to provide a new sentence, which corresponds to a missing axiom. At no point does the domain expert need to understand the underlying logical language.

10 Acknowledgements

The authors would like to thank Matthias Anlauf, Butch Anton, Danny Bobrow, Ray Perrault, the late Mark Stickel, James Tarver, Mabry Tyson, and Michael Wessel for discussions that helped formulate the ideas in this paper and get them to work.

References

[1] Ion Androutsopoulos, Graeme D. Ritchie, and Peter Thanisch. Natural language interfaces to databases - an introduction. *Natural Language Engineering*, 1(1):29–81, 1995.

[2] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, pages 1533–1544, 2013.

[3] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250, 2008.

[4] Johan Bos. Three stories on automated reasoning for natural language understanding. In *Proceedings of ESCoR (IJCAR Workshop)*, pages 81–91, 2006.

[5] Johan Bos. Wide-coverage semantic analysis with boxer. In *Proceedings of the 2008 Conference on Semantics in Text Processing*, pages 277–286, 2008.

[6] Philipp Cimiano and Michael Minock. Natural language interfaces: what is the problem?–a data-driven quantitative analysis. In *Natural language processing and information systems*, pages 192–206, 2010.

[7] Richard Crouch. Packed rewriting for mapping semantics to kr. In *Proceedings of the 6th International Workshop on Computational Semantics*, pages 103–14, 2005.

[8] Anette Frank, Hans-Ulrich Krieger, Feiyu Xu, Hans Uszkoreit, Berthold Crysmann, Brigitte Jörg, and Ulrich Schäfer. Querying structured knowledge sources. In *Proceedings of AAAI-05. Workshop on question answering in restricted domains*, pages 10–19, 2005.

[9] Alessandra Giordani and Alessandro Moschitti. Semantic mapping between natural language questions and sql queries via syntactic pairing. In *Natural Language Processing and Information Systems*, pages 207–221, 2010.

[10] Douglas Lenat. Cycorp: Home of smarter solutions. online, 2014. `http://www.cyc.com`.

[11] Mark T. Maybury, editor. *New Directions in Question Answering*. AAAI Press, 2004.

[12] Raymond J Mooney. Learning for semantic parsing. In *Proceedings of Computational Linguistics and Intelligent Text Processing*, pages 311–324. Springer, 2007.

[13] Adam Pease, Ian Niles, and John Li. The suggested upper merged ontology: A large ontology for the semantic web and its applications (2002). online, 2002. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.6817`.

[14] Adam Purtee and Lenhart Schubert. Ttt: A tree transduction language for syntactic and semantic processing. In *Proceedings of the EACL Workshop on Applications of Tree Automata Techniques in Natural Language Processing*, pages 21–30, 2012.

[15] Kyle Richardson, Daniel G. Bobrow, Cleo Condoravdi, Richard J. Waldinger, and Amar Das. English access to structured data. In *Proceedings of IEEE International Conference on Semantic Computing*, pages 13–20, 2011.

[16] Lenhart Schubert. From treebank parses to episodic logic and commonsense inference. In *Proceedings of ACL Workshop on Semantic Parsing*, pages 55–60, 2014.

[17] Mark E. Stickel, Richard J. Waldinger, and Vinay K. Chaudhri. A guide to SNARK, 2000.

[18] Christina Unger, Lorenz Bühmann, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, Daniel Gerber, and Philipp Cimiano. Template-based question answering over rdf data. In *Proceedings of the 21st international conference on World Wide Web*, pages 639–648, 2012.

[19] Richard Waldinger. Whatever happened to deductive question answering? In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 15–16, 2007.

[20] Yuk Wah Wong and Raymond J Mooney. Learning for semantic parsing with statistical machine translation. In *Proceedings of HTL-NAACL*, pages 439–446, 2006.