

Deduction for Natural Language Access to Data

Cleo Condoravdi¹, Kyle Richardson², Vishal Sikka³, Asuman Suenbuel³ and
Richard Waldinger^{4*}

¹ Department of Linguistics, Stanford University
Stanford, California, U.S.A.
`cleoc@csli.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.

`vishal.sikka@sap.com`, `asuman.suenbuel@sap.com`

⁴ Artificial Intelligence Center, SRI International
Menlo Park, California, U.S.A
`waldinger@ai.sri.com`

1 The Problem

Our interest is in natural language access to 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 reasoning over an axiomatic subject domain theory provides a handle on some of the natural-language issues. The users of such a system would not need to resort to a graphical user interface and would not have to know which databases were available or how they were organized. They would certainly not need to write programs in a database query language.

2 Obstacles

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. Given the obvious practical applications of such research, interest in the topic has not been limited to the research community but rather has been investigated for industrial applications. In the automated reasoning community, the basic techniques for question-answering on structured data have long been understood, but these tools have not been widely used in recent mainstream natural language processing (some exceptions: [2]), let alone for work on natural language question answering. While we do not have the space for a full survey here, there is a literature discussion in [5].

In natural language processing proper, work on interfaces to databases has traditionally focused on transforming a natural language question directly to an expression in a formal database query language, such as SQL [3]. The expression is then used by a standard database management system to access the target knowledge source and retrieve the desired answer. Usually, the query is applied to a single database with a predefined semantics. More recently, research has centered around learning the transformation from language to formal queries using collections of parallel data (for a review, see [6]).

*From NLSR2014— the workshop on Natural Language Services for Reasoners

In this recent work especially, there has been scant attention on using richer formal representation languages, and the available datasets continue to rely on formal database query languages for representing query semantics.

A major drawback to these approaches is that a simple natural language query may be complex 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 the same question. It may require world knowledge to translate the user's formulation of the query into that preferred by the database; it is hard to integrate world knowledge into the query interpretation process. The problem is compounded when the answer depends on information obtained from several heterogeneous sources. 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 languages are famously ambiguous. While a human user can get the intended meaning quickly, a parser can find hundreds of possible meanings for a relatively simple English query. The person can quickly discard the unintended interpretations through knowledge of the context and the intended subject domain.

3 Our Approach

In the Quest project of the SAP business enterprise software company, English queries are understood by combining natural language technology and automatic deduction. Unlike the traditional approaches that rely on formal database-query languages, Quest allows users to define subject domain knowledge and axiomatic theories using a rich and customizable first-order language. In this case, the subject domain is limited to a business application and provided by a domain expert. Symbols in the axiomatic theory are semantically linked to external structured data resources, and particular fields within these resources, using a procedural attachment mechanism. Such a mechanism also makes it possible to coordinate knowledge and reason across multiple disparate resources. English questions are translated by a semantic parser into first-order logic, which captures the intended meaning of the query. This logical form is treated as a theorem and submitted to an automatic theorem prover, which tries to show that it follows from axioms in the subject domain theory. An answer to the question is extracted from a proof of the theorem. Typically there are many proofs, each yielding a different 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. Procedural attachment allows a database to act as a virtual extension of the subject domain theory. Rather than forcing the user to come up with a single question that captures everything, Quest allows a query to be expressed as a sequence of questions, each of which may extend or restrict previous ones. The user may examine a query and then add new constraints or conditions. For instance, a user might ask a query such as *Show clients with a high outstanding debt*, view the answers, and then add the condition *The debt should be short term* to restrict the class of answers. We can extend the question by asking *What is the nationality of the company?* as a follow-up. (Note that in this context the clients are companies.)

Terms in the subject domain theory have sorts, and function and predicate symbols are declared as to what sort of terms they will accept as arguments. Thus, a relation **company-has-debt** takes two arguments, whose sorts are **company** and **debt**, respectively. This information helps us discard parses that make no sense in the subject domain. For example, the query sequence *Show a client with a debt. Require it to be high* is syntactically ambiguous, in the sense that *it* could refer to the client or the debt. However, in our subject domain we define high debts but not high clients, so the former interpretation can be excluded by sort information.

In translating a query from English to logical form, there is always the chance that we have misunderstood the user's intention. For this reason, we translate the logic back into a new English version, the *playback*, that is cumbersome but presumably unambiguous. A user who is not satisfied with the way the query has been understood can rephrase it. If the query is ambiguous and the system fails to

disambiguate it, we can generate playback and answers for the two or more logical forms, and present them to the user, who may discard those that mistook the intention of the query.

Important features of our approach are the use of the playback device to ensure that the user's question has been correctly understood; the answer extraction mechanism; the use of theorem proving to disambiguate queries; the use of a procedural-attachment mechanism to link symbols in our theory to external databases; the use of temporal inference; and a natural-language explanation facility.

4 Details of Approach

Existing approaches to interpreting language queries focus on very basic question types, and ignore the wider variety of complex linguistic factors that are involved. For example, phenomena such as scope and quantification, temporal reasoning, disjunction, negation, and world knowledge are often ignored, in part because the target representations are limited to the constructs available in weak languages such as SQL and the concepts and symbols in the target databases. Using the rich first-order language provided in Quest, domain experts can define new predicate and relation symbols on-the-fly that satisfy various linguistic and domain constraints.

We have developed a prototype parser called SAPL, which is built on top of the SAP Inxight text analytics engine. At an earlier stage of processing, the text analytics engine is used to perform entity and part-of-speech tagging. SAPL then uses domain knowledge, defined by a domain expert, on top of a bottom-up chart parser to find syntactic patterns that match entity pairs defined in the knowledge base, and maps these pairs to their corresponding relations. This work builds on earlier work that uses the PARC XLE system [8] and MT-style transfer rules for mapping syntactic structures to semantic clauses.

The axiomatic theory classifies all entities according to the sort structure. For example, in the Quest theory we distinguish between such entities as companies, nationalities, debts, dates, and quantities of money. For each relation in the theory, a declaration specifies the sorts of its arguments; e.g., as we have mentioned, the relation `company-has-debt` has a company as its first argument and a debt as its second. The parser can discard interpretations that violate such sort constraints.

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 the logical form; this form is regarded as a conjecture that is submitted to the theorem prover SNARK [9].

The proof is conducted within the theory, which consists of a vocabulary of symbols and a set of axioms, i.e., declarative sentences in a logical language. These 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. Some of the symbols in the theory have procedural attachments to data sources; these are the lowest level symbols. Others are higher level, and are related to the lower-level symbols by axioms. The logical form is decomposed and transformed according to the axioms of the theory.

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. A procedural-attachment mechanism allows us to link the axiomatic theory to structured data, and an answer extraction mechanism to construct an answer to the query from the completed proof. 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 then 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 by hand; 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.

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.

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. 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.

5 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 *examples* in *Show your ideas with examples*. 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 omitted here. 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. There are two rows:

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

that 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.

SAPL introduces appropriate semantic relations between these variables. For instance, in the representation, a row

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

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`.

The logical form obtained from this representation is as follows:

```
(exists ((company_48 :sort company))
  (exists ((nationality_47 :sort nationality))
    (exists ((debt_45 :sort debt))
      (exists ((dollar_46 :sort money_amount))
```

```

      (and (is-debt debt_45)
            (is-company company_48)
            (nationality-of company_48 nationality_47)
            (company-has-debt company_48 debt_45)
            (qlabel-n dollar_46 (* 100000000 dollar) more-than)
            (debt-has-money-value debt_45 dollar_46))))))
:answer
(ans
 (tuple company_48 nationality_47 debt_45 dollar_46))

```

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.

6 Proof

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

```

(<=
 (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 suggests 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 ?d.debt ?dso.dso ?l.location ?size.number (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 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`.

7 Future Work and Summary

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 deductive methods to assist in the formation or extension of an axiomatic knowledge base.

Quest translates queries 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.

Rather than starting developing axioms from scratch, we can use an existing ontology, such as SUMO [7] or Cyc [4]. SUMO gives us some 80,000 axioms for common concepts. Every word in the online thesaurus Wordnet maps into some SUMO concept. While we still need to introduce more axioms, to get a deeper 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 predicate symbol with a database table. Each column in the database must be associated with a concept in the ontology. We may then automatically generate code that accesses that table when the predicate symbol occurs in the search for a proof. The domain expert may also provide declarative sentences that link 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.

8 Acknowledgements

The authors would like to thank Matthias Anlauf, Butch Anton, Danny Bobrow, Ray Perrault, the late Mark Stickel, Mabry Tyson, James Tarver, 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] Johan Bos. Three stories on automated reasoning for natural language understanding. In *Proceedings of ESCoR (IJCAR Workshop)*, pages 81–91, 2006.
- [3] 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.
- [4] Douglas Lenat. Cycorp: Home of smarter solutions. online, 2014. <http://www.cyc.com>.
- [5] Mark T. Maybury, editor. *New Directions in Question Answering*. AAAI Press, 2004.
- [6] Raymond J Mooney. Learning for semantic parsing. In *Computational Linguistics and Intelligent Text Processing*, pages 311–324. Springer, 2007.
- [7] 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>.
- [8] Kyle Richardson, Daniel G. Bobrow, Cleo Condoravdi, Richard J. Waldinger, and Amar Das. English access to structured data. In *ICSC*, pages 13–20. IEEE, 2011.
- [9] Mark E. Stickel, Richard J. Waldinger, and Vinay K. Chaudhri. A guide to snark, 2000.