Natural Language Access to Data: It Takes Common Sense!

Cleo Condoravdi [*]
Department of Linguistics,
Stanford University
Stanford, California, U.S.A.

Kyle Richardson
Institute for Natural Language Processing,
University of Stuttgart
Stuttgart, Germany

Vishal Sikka
Infosys Ltd.
Palo Alto, California, U.S.A.

Asuman Suenbuel
SAP Labs
Palo Alto, California, U.S.A.

Richard Waldinger
Artificial Intelligence Center, SRI International
Menlo Park, California, U.S.A.

Abstract

Commonsense reasoning proves to be an essential tool for natural-language access to data. In a deductive approach to this problem, language processing technology translates English queries into a first-order logical form, which is regarded as a conjecture to be established by a theorem prover. Subject domain knowledge is encoded in an axiomatic theory equipped with links to appropriate databases. Commonsense reasoning is necessary to disambiguate the query, to connect the query with relevant tables in the databases, to deal with logical relationships in the query, and to achieve interoperability between disparate databases. This is illustrated with examples from a proof-of-concept system called Quest, which deals with queries over business enterprise data for an industrial QA system.

Motivation

We are interested in a particular 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.
- Answers can be obtained from multiple databases, which may be disparate; they need not be intended to be used together.
- Answers need not be present explicitly in any of the databases; they may be deduced or computed from information supplied by several databases.
- Queries may be expressed incrementally; on seeing the answers from one query, the user may impose restrictions or add follow-on questions that refer back to earlier queries.

[*]Author names are listed in alphabetical order.

To make the problem more manageable, we allow the following restrictions on the problem.

- Questions are limited to a particular subject domain.
- We access only a known set of databases; we do not seek new databases when answering a new question.
- All our data is structured; we do not deal with, say, unstructured English knowledge sources.

In this context, reasoning plays a number of roles:

Natural Language Interpretation: Natural language queries are highly ambiguous. Even for an apparently simple query, a parser may find a large space of analyses, most of which can be discarded with some knowledge of the subject domain. For instance, in our business enterprise domain, if we ask the business enterprise system Quest *Find me a company with a debt. It should be high*, a person will know that *it* is the debt that should be *high*, not the *company*, because in this context we define *high debts* but not *high companies*. Similarly, syntactic ambiguities (e.g. *Companies that have decreasing daily sales outstanding <u>within</u> the last year*) can often be resolved by having background domain knowledge about relations that hold between entities in the target domain.

Finding Answers and Coordinating Databases: 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 data is represented in a database (e.g, Company A has 1,000,000 dollars in the "debt" column,) and the way the query is phrased (*The company owes more than 500,000 euros*). The answer may depend on more than one data source, and there may be a complex inferential link between the information provided by the sources and the query posed by the user. When multiple sources must cooperate, we cannot assume that information provided by one source is in the form required by another (e.g., one may provide a monetary value in dollars, while another requires it in euros.) These connections can all be filled in by reasoning over a declarative knowledge base.

Logical and Commonsense Reasoning: Reasoning also helps to deal with logical connectives, quantifiers, and temporal and spatial operators that may appear in a query. We might want to find a client in Northern Europe that has *not* had a high balance in the last two years. We need temporal reasoning to tell if a debt was contracted within the last two years, 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 be able to deal with quantifiers.

We cannot use conventions such as the unique-name assumption (i.e. that distinct names refer to distinct things) or the closed-world assumption (i.e. that our data sources are complete) blindly in this context. For instance, while we must assume that companies have id numbers that are unique, we must allow the possibility that different people or even cities have the same name. We must assume that our database contains a complete list of clients, but we do not require that our currency conversion table contains all the world's currencies. Thus we must intermix unique-name and closed-world reasoning with ordinary logical inference.

Outline of Approach

Below is an outline of the approach we are following:

semantic parsing: Natural language queries are parsed and translated into a formal semantic representation.

logical form: A first-order logical representation is constructed from the semantic representation. The entities to be found are represented by existentially quantified variables. (If the semantic representation has unresolvable ambiguities, several logical forms may be constructed).

playback: The logical form is translated back into a pedantic but less ambiguous English, and displayed to the user for approval or modification.

proof: The selected logical form is submitted as a conjecture to a theorem prover, to show that it is a theorem in a given *subject domain theory*.

procedural attachment: Certain relation (and function) symbols in the theory are linked to tables in selected databases, which indicate what entities satisfy them. When these linked symbols arise in the proof search, existential variables are replaced by the satisfying entities, and the proof search continues.

answer extraction: When the proof is complete, an answer to the original query is extracted. An English explanation of the answer is constructed, which may serve as a partial explanation.

The approach we outline centers around using first-order logic and reasoning. 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. Such an approach is within the broader tradition of deductive question-answering (Waldinger 2007).

Our approach contrasts with approaches that use formal query languages, such as SQL, and database reasoners to represent and execute natural language queries (Giordani and Moschitti 2010). One main drawback to these approaches is that it may be demading to translate a simple natural language query into a language like SQL, due to known limitations in the expressive power of database languages (Androutsopoulos, Ritchie, and Thanisch 1995). Furthermore, the way the user naturally formulates a query may be quite different from the way the answer is elicited from the target databases; It may require considerable background knowledge to connect the formulations. While a full defense of the logical approach is beyond the scope of this paper, we argue that the approach has the following advantages:

extensibility: New symbols and background knowledge can be added on-the-fly using an expressive logical language, independently of the target databases. We intend to allow the domain theory to be extended by translating declarative natural-language exposition into logical axioms.

external grounding: Domain symbols and background knowledge can be arbitrarily linked to database content via *procedural attachments* to symbols in the subject domain theory.

precision: Logic allows for a principled and deep analysis of logical connectives, negation, quantification, temporal operators, and other constructs not found in standard database languages.

We discuss our approach in the context of a prototype system called Quest, which was developed with SAP for business enterprise data access. A similar approach has been pursued in the Quadri system, which was developed with the Stanford Biomedical Informatics Research Program to answer questions about HIV therapy (Richardson et al. 2011).

Lightweight Semantic Parsing

The system's knowledge of the subject domain is represented in an axiomatic theory in first-order logic. The axioms embody the knowledge of our subject domain experts. The theory is sorted; every constant is declared to have a sort; every relation and function symbol is declared to accept arguments of a specified sort. Function symbols are also declared as to the sort of values they return.

Our approach relies on translating each English query into a sentence in the language of the subject domain theory, first by creating a semantic representation of the query and then by translating that representation into a structured logical form. Our system performs this mapping via a simple pattern- or template-matching approach (Unger et al. 2012), and exploits rich subject domain knowledge to guide the analysis. The matching is done by using a set of ordered (local) rewrite rules (Crouch 2005) defined on abstract syntactic patterns and dictated by the background subject domain.

The sample rewrite rule shown in Figure 1.1 encodes a relation `company-has-debt` between entities `company` and `debt`. The rule is run over the output of a syntactic parser, which is rewritten to the *flat* semantic representation

```
1. (NP (NP company_x) (.? .? (... debt_y)))
   =>
   (company-has-debt company_x debt_y)

   i. (NP (NP company_company)
          (PP with_P (NP debt_debt))) ...

   ii.(NP (NP firms_company)
          (WHNP that_WDT
                (S owe_money_debt))) ...

2. (? (NP debt_x) (.? .? (... money_amount_y)))
   =>
   (debt-has-value debt_x money_amount_y)
```

Figure 1: Example rewrite rules for mapping abstract syntactic patterns (on the left of the rule) to flat semantic clauses (on the right). Matching patterns for (1) are shown in (1i-ii.)

on the right when both entities are found inside a matching syntactic configuration. As shown by the regex-like operators *?, ., ...*, such rewrites can be written at varying levels of abstraction, allowing a rule to generalize over various types of syntactic forms (see the matching fragments in 1i-ii) and skip over certain details in the syntactic analysis. In addition to those matching patterns from the subject domain theory, rules can be created for modeling other linguistic constraints.

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 performs entity and part-of-speech tagging, and maps each entity token to a variable. SAPL then uses a bottom-up chart parser and incrementally matches rewrite rules to syntactic patterns annotated with their entity types and stores each resulting semantic representation in a chart. During the parsing, knowledge from the theory can be used to guide the analysis and eliminate certain syntactic ambiguities. For example, in the fragment in (1i.), the attachment between `company` and `debt` is licensed by knowledge that companies and debts relate to one another, and this knowledge can be used to block further attachments encountered later in the search. Once the parser is finished, the representation can be extracted from the chart. One can deal with ambiguity by solving for the most probable or k most probable representations (or trees) using standard methods from parsing (Huang and Chiang 2005).

The full output of SAPL applied to a particular example is shown in Figure 2. In this case, the query has multiple questions, and the second question is a follow-up to the first. Entities in the query are expressed as logical variables with appropriate sorts and quantification, and the relations (including logical connectives and scopal relations) among them are expressed in a flat form which mirrors the local syntactic structures from which the patterns are rewritten. In this example, all the quantifiers are existential, because the variables all stand for entities to be found. Some information, such as the coreference between variables and background knowledge, is left underspecified and resolved at a later reasoning stage (described in the subsequent sections).

```
((input Show a company with a debt of
  more than 100 million dollars.
  What is the nationality of the client?)
(top-level company5 0)
(top-level company6 0)
(definite company6)
(definite nationality8)
(desired-answer company5)
(quant exists debt3  sort debt)
(quant exists company5  sort company)
(quant exists nationality8 sort nationality)
(quant exists company6  sort company)
(quant
   (complex-num
      more than 100000000 dollar)
    dollar4 sort money-amount)
(exists-group ex-grp45
   (debt3 nationality8))
(scopes-over nscope company5 ex-grp45)
(in nscope debt3
   (company-has-debt company5 debt3))
(in nscope dollar4
   (debt-has-value debt3 dollar4))
(in nscope nationality8
   (nationality-of
      company6 nationality8)))
```

Figure 2: Output of the SAPL semantic parser for a multi-sentence query. Relations are first expressed in a flat clausal form.

```
(exists ((company5 :sort company))
(exists ((nationality8 :sort nationality))
(exists ((debt3 :sort debt))
(exists ((dollar4 :sort money-amount))
 (and
  (is-debt debt3)
  (is-company company5)
  (nationality-of company5 nationality8)
  (company-has-debt company5 debt3)
  (more-than dollar4 (* 100000000 dollar))
  (debt-has-value debt3 dollar4)))))))
      :answer
(ans
  company5 nationality8 debt3 dollar4)
```

Figure 3: Normalized logical form for the flat semantic representation in Figure 1. The *ans* list shows the variables that, when instantiated during the proof, constitute an answer to the query.

```
there exists a nationality8 such that
there exists a debt3 such that
there exists a dollar4 such that
there exists a company7 such that
  company7 is a company,
  debt3 is a debt,
  the nationality of company7 is nationality8,
  company7 has debt3,
  dollar4 is more-than 100000000 dollars and
  the amount of debt3 is dollar4
```

Figure 4: Playback for the logical form in Figure 3, expressed in quasi-English for the user to inspect.

The SAPL parser is based on earlier work that used the PARC XLE parser and the packed rewrite system (Crouch 2005; Bobrow et al. 2007), for parsing questions about HIV treatment (Richardson et al. 2011; Bobrow et al. 2011). The overall method bares resemblance to recent work (Purtee and Schubert 2012) on using tree-to-tree transducers for modeling syntax-semantics correspondences, as well as recent work in NLP on data-driven semantic parsing. In the later case, such work has focused on learning interpretation rules from parallel data (Mooney 2007), often by employing techniques and formal models from statistical machine translation (Wong and Mooney 2006) and statistical parsing (Zettlemoyer and Collins 2007). Given enough data, such techniques could be used to learn the rules we describe, cutting out the need for hand-coding some of the rewrite rules.

The idea of using semantic knowledge to disambiguate natural language and guide interpretation is not a new one, and has been explored in a variety of different settings (Allen, Swift, and De Beaumont 2008). It is a key component to the classic work by (Woods 1978) on the Lunar QA system. Such an idea has also gained some traction in recent NLP work on semantic parsing. For example, (Berant et al. 2013) look at using knowledge from the large Freebase ontology (Bollacker et al. 2008), as well as answers, to learn representations for open-domain questions.

Logical Form

The resulting semantic representation produced in the previous step is flat, but it contains the information necessary to construct a nested logical form. Quantifiers in the formula are put at the outermost level, and variables from follow-up questions are captured by quantifiers from the original question. A normalized logical form constructed from the representation in Figure 2 is shown in Figure 3.

Answering this question involves finding a company that has a debt of more than 100 million dollars, and returning the nationality of this company. Note that the condition of the second question:

```
(nationality-of company5 nationality8).
```

falls within the scope of the company quantifier from the first question. In fact, the logical form for each question in the sequence falls within the scope of the same quantifiers. This convention allows us to refer in later questions to entities that occur in earlier questions in the sequence, and resolve certain types of anaphora.

For the logical form in Figure 3 we generate the playback in Figure 4, which is shown to the user for approval. Mistranslations may be evident when the query is presented in this way.

Theorem Proving

The logical form is submitted to a theorem prover to be proved. The proof is conducted in the same axiomatic subject-domain theory that was used in the semantic parsing phase. The axioms of the theory define the meaning of the concepts in our query, specify the capabilities of our data sources, and provide the background knowledge necessary to relate them. The logical form is decomposed and expanded according to these axioms.

We use SRI's open-source first-order reasoner SNARK (Stickel, Waldinger, and Chaudhri 2000). SNARK is fully automatic and uses machine-oriented inference rules, such as resolution for general-purpose reasoning and paramodulation for reasoning about equality. It has special facilities for accelerated reasoning about space and time; a sort mechanism for internalizing the reasoning about sorts or types; a procedural-attachment mechanism for linking selected relation and function symbols to external procedures, especially accessing an external database; and an answer-extraction mechanism for constructing an answer to a query from a completed proof. Although SNARK is ideally suited and has been successfully used in other deductive question-answering tasks (Shrager et al. 2007), any theorem prover with comparable features could be used.

SNARK is a refutation procedure, it looks for inconsistencies in a set of logical sentences. When trying to prove that a conjecture holds in a given theory, it negates the conjecture, adds it to the set of axioms of the theory, and tries to find a contradiction in the resulting set. For this purpose, it will apply inference rules, which deduce new sentences, and adds those to the set. The process continues until no further inferences are possible or a contradiction is obtained—this shows that the negated conjecture is inconsistent with the axioms of the theory and, hence, the conjecture itself is valid in the theory.

The order in which inference rules are applied depends on a set of heuristic principles which are under the control of the developer of the theory. Weights can be assigned to symbols, which causes SNARK to focus attention on "lighter" sentences in its set. Also, an ordering is applied to symbols, which causes SNARK to focus on particular parts of of a sentence; typically, symbols will be replaced by other symbols that follow them in the ordering. Since first-order theorem proving is undecidable and the search can go on forever, we give SNARK a time limit on proving a given theorem. This is set empirically—for the examples in this paper, a limit of six seconds was sufficient, although most of the theorems were proved much more quickly.

The answer-extraction mechanism, developed for program synthesis applications (Manna and Waldinger 1980), associates with each sentence in the set an answer term, such that if the sentence can be falsified, the corresponding instance of the answer term will satisfy the original query. When a contradiction is obtained, the corresponding answer term will be an answer to the query. Typically there are many proofs of a given theorem, each leading to a possibly different answer. These can be assembled and presented to the user.

The procedural-attachment mechanism links selected relation and function symbols in the theory with external procedures, which indicate how the linked symbol can be computed. Typically those procedures will involve accessing an external database. In the QUEST theory, the symbol `company-record` is linked to an external database of companies, so if the name of a client company is given, the other attributes of the company (e.g. the size, the nationality, and the amount the client owes) can be looked up. When the `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 currency conversion. The procedural-attachment mechanism allows the theorem prover to access not only the knowledge that resides in its axioms, but also information that is stored in external procedures.

If, after the question is answered, there is a follow-up question, the query sequence is reparsed, a new logical form is constructed, and a new proof is sought. New questions at the end of a query add conditions to the conjecture, which can force us to discard proofs that satisfied the original query, so earlier proofs cannot be reused.

;; expansion axiom
```
(<=> (company-has-debt ?company ?debt)
(exists (?country-code ...)
 (and
  (company-record
   ?company ?debt ?country-code ...)
  (positive ?debt)))).
```
;; nationality axiom
```
(iff
 (nationality-of ?company ?nationality)
 (exists
  (?country ?country-code ?debt ...)
  (and
   (nationality-for-country
    ?nationality ?country)
   (country-code-of-country
    ?country ?country-code)
   (company-record
    ?company ?debt ?country-code ..)))
```
;; uniqueness axiom
```
(implies
 (and
   (company-record ?company ?debt1 ...)
   (company-record ?company ?debt2 ...))
 (= ?debt1 ?debt2))
```
;; duration axiom (time)
```
(=
 (duration
  (make-interval
   ?time-point1 ?time-point2))
 (minus-time ?time-point2 ?time-point1)
```
Figure 5: Example Snark axioms from the Quest business domain. See text for details

Proof

As an example, we walk through the steps involved in proving the logical form in Figure 3. The negated conjecture contains (after the quantifiers are removed by skolemization) the following subcondition:

```
(company-has-debt ?company7 ?debt3)
```

The variables, indicated by the prefix `?`, are implicitly quantified existentially; they can be replaced by terms during the proof. The axiomatic subject-domain theory contains the *expansion* axiom shown in Figure 5 which expands the above condition to a `company-record` symbol.

The resulting `company-record` relation symbol has a procedural attachment to the database of companies; we omit most of its many arguments. (Note that the data, based on actual SAP data, has been garbled to protect the privacy of clients.) While the database does not contain the nationality of each company, it does contain a two-character country code. For instance, for the company `SL Foods Inc.`, the code is `CH`, which stands for Switzerland. We distinguish between countries such as Switzerland and nationalities such as Swiss. We require that for a company to have a debt, the amount owed must be positive.

Applying the resolution rule to this axiom and the negated conjecture causes a new sentence to be deduced, containing the subcondition

```
(company-record
 ?company7 ?debt3 ?country-code8 ...).
```

Because the relation symbol `company-record` has a procedural attachment, SNARK can at this point consult the linked database of companies. Of course, many companies will match this. One of them is the company `SL Foods Inc.`, which has a debt of around 105 million dollars.

Two other conditions in the conjecture that relate to the value of the target debt are

```
(debt-has-value ?debt3 ?dollar5).
(> ?dollar5 100000000).
```

In a way similar to the transformation of the relation `company-has-debt`, the relation `debt-has-value` is expanded to a formula involving the relation `company-record`; the company database is consulted, and the actual debt of each company in the database is retrieved and entered into the proof search, replacing the variable `?dollar5`. It remains to prove the second greater-than condition above using the value of `?dollar5` obtained from the database. Because a hundred million is quite a large debt, all companies but `SL Foods Inc.` fail to satisfy the inequality.

It also remains to establish the nationality of the company. The logical form contains the condition shown below, which is transformed into the *nationality axiom* shown in Figure 5.

```
(nationality-of company5 nationality8).
```

The relations `nationality-for-country` and `country-code-of-country` that result from the axiom are equipped with procedural attachments that allow them to consult external websites which link `Swiss` with `Switzerland` and `Switzerland` with `CH`, respectively. This allows us to answer that the nationality of our company `SL-Foods-Inc.` is `Swiss`.

Other Capabilities

In this section we show other ways in which reasoning is essential for natural language database access.

disambiguating queries: The query sequence *Get a company with a debt. It should be high.* will yield companies with high debts. The sequence *Get a company with a debt. It should be Swiss.* will yield a Swiss company. The subject domain theory defines high debts but not high companies; it defines Swiss companies but not Swiss debts. In each case, subject domain reasoning allows us to discard interpretations that are plausible syntactically but make no sense semantically. This sort of processing is related to some of the problems in the Winograd Schema Challenge, [1] an elaboration of the Turing Test.

logical operators: Reasoning is necessary when the query contains logical connectives or quantifiers. For example, the disjunction operation allows us to answer queries such as *What companies have a high debt or a long-term debt?* Concepts such as high and low or long- or short-term are defined by axioms.

In *Every company does not have a negative debt,* Quest can answer immediately in the affirmative because the definition of debt requires that debts be positive; Quest does not need to consult the database.

The question *What companies do not have a low debt?* requires reasoning about negation. The logical form, after negation and skolemization, contain the *skolem* condition

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

for all companies `?company1`. Here

```
(debt-sk ?company1)
```

is a skolem term that stands for an arbitrary debt of the company. We also assume (after negation and skolemization) the *lowness condition* `(low (debt-sk ?company1))`. The *uniqueness* axiom shown in Figure 5 implies that debts of companies are unique. The procedural attachment searches through the database and finds companies that have high debts, e.g.,

```
(company-record
 beck-sports <some-high-debt> ...).
```

Also, by the skolem condition and the expansion axiom, we have

```
(company-record
 beck-sports (debt-sk ?company1) ...).
```

The uniqueness axiom can then be applied to show that `<some-high-debt>` is equal to `(debt-sk company1)`, contradicting the lowness of the skolem term. Hence the company Beck Sports will be among the answers.

temporal reasoning: In the query *Show companies with a high debt within the last two years*, the logical form will demand the existence of a temporal interval that represents the last two years; it has a duration of two years and ends at the time point `(now)`, the time at which the question is being asked. The duration of a time interval is defined by the *duration* axiom in Figure 5. In other words, the duration of a time interval is the difference between its end points. SNARK uses a procedural attachment to a date arithmetic procedure to compute the difference between two time points.

[1]http://commonsensereasoning.org/winograd.html

interoperability The subject domain theory provides a lingua franca between the query and the various data bases and information sources. Terms from any of these sources will be translated into whatever representation is preferred by the subject domain theory. For instance, if we ask *What is a Swiss company that has a debt of more than 10 million euros within the last 100 weeks*, SNARK can use external tables to translate from Swiss to Switzerland and then to the two letter code CH; it will translate the 100 week duration to seconds to compare with the debt of the selected companies; and it will use a currency conversion website to translate euros, the currency of the query, into dollars, the currency of the database.

amalgamation questions: Some queries require us to find more than one answer and then perform a deduction or computation on the answers. For instance, the query *What company has the lowest debt within the last two years* requires us to find all the companies that have debts within the last two years and then select the company with the lowest. We may then add additional conditions to the query, such as *Consider only companies with a debt of more than 50 dollars. What is the nationality of the company?* Note that the additional condition causes the earlier company to be replaced, if it has a debt of less than 50 dollars.

Current and Future Work

Our treatment of amalgamation questions is currently ad hoc and is done procedurally outside of the subject domain theory. To get the full advantage of a theorem prover, we would be better off being able to represent the full set (or list) of answers to the query within the theory itself. We would then be able to use axioms as well as procedures to answer amalgamation questions. Although in our work up to now first-order logic has been adequate, it would be advantageous to have higher-order logic to characterize the set of all answers to a query.

We are currently looking at question answering in an entirely different domain, that of Linux programming. This involves reasoning about operations, like program instructions, that can cause changes of state. There is a program synthesis aspect—how to construct a program to cause certain specified effects. And there is also a more standard question-answering aspect, such as searching documentation to diagnose programming errors in Linux scripts.

In all of our applications of deductive question answering we have relied on a hand-crafted subject domain theory. Developing and testing such a theory is labor intensive and requires specialized expertise. We hope to facilitate this task by allowing a subject domain expert to extend an axiomatic theory, or develop a new one, by writing new axioms in natural language and having them translated into logical form, much as we do now with queries. This would mean that the subject domain expert could extend the theory without having any knowledge of logic or programming. Theorem proving tools could be brought to bear to detect inconsistencies in the expanded theory.

Acknowledgements

Thanks to Matthias Anlauff, Butch Anton, Danny Bobrow, Ray Perrault, the late Mark Stickel, James Tarver, Mabry Tyson, Michael Wessel, and David Wilkins for providing expertise and ideas. Research in this paper was supported by a contract from SAP.

References

Allen, J. F.; Swift, M.; and De Beaumont, W. 2008. Deep semantic analysis of text. In *Proceedings of the 2008 Conference on Semantics in Text Processing*, 343–354. Association for Computational Linguistics.

Androutsopoulos, I.; Ritchie, G. D.; and Thanisch, P. 1995. Natural language interfaces to databases - an introduction. *Natural Language Engineering* 1(1):29–81.

Berant, J.; Chou, A.; Frostig, R.; and Liang, P. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of EMNLP*, 1533–1544.

Bobrow, D. G.; Cheslow, B.; Condoravdi, C.; Karttunen, L.; King, T. H.; Nairn, R.; de Paiva, V.; Price, C.; and Zaenen, A. 2007. PARCs bridge and question answering system. In *Proc. of the GEAF 2007 Workshop*.

Bobrow, D. G.; Condoravdi, C.; Richardson, K.; Waldinger, R.; and Das, A. 2011. Deducing answers to English questions from structured data. In *Proceedings of the 16th international Conference on Intelligent User interfaces*, 299–302. ACM.

Bollacker, K.; Evans, C.; Paritosh, P.; Sturge, T.; and Taylor, J. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, 1247–1250.

Crouch, R. 2005. Packed rewriting for mapping semantics to KR. In *Proceedings of the 6th International Workshop on Computational Semantics*, 103–14.

Giordani, A., and Moschitti, A. 2010. Semantic mapping between natural language questions and sql queries via syntactic pairing. In *Natural Language Processing and Information Systems*, 207–221.

Huang, L., and Chiang, D. 2005. Better k-best parsing. In *Proceedings of Workshop on Parsing Technology*, 53–64. Association for Computational Linguistics.

Manna, Z., and Waldinger, R. 1980. A deductive approach to program synthesis. *ACM Trans. Program. Lang. Syst.* 2(1):90–121.

Mooney, R. J. 2007. Learning for semantic parsing. In *Proceedings of Computational Linguistics and Intelligent Text Processing*. Springer. 311–324.

Purtee, A., and Schubert, L. 2012. 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*, 21–30.

Richardson, K.; Bobrow, D. G.; Condoravdi, C.; Waldinger, R. J.; and Das, A. 2011. English access to structured data. In *Proceedings of IEEE International Conference on Semantic Computing*, 13–20.

Shrager, J.; Waldinger, R.; Stickel, M.; and Massar, J. 2007. Deductive biocomputing. *PloS One* 2(4):e339.

Stickel, M. E.; Waldinger, R. J.; and Chaudhri, V. K. 2000. A guide to SNARK.

Unger, C.; Bühmann, L.; Lehmann, J.; Ngonga Ngomo, A.-C.; Gerber, D.; and Cimiano, P. 2012. Template-based question answering over rdf data. In *Proceedings of the 21st International Conference on World Wide Web*, 639–648.

Waldinger, R. 2007. Whatever happened to deductive question answering? In *Logic for Programming, Artificial Intelligence, and Reasoning*, 15–16.

Wong, Y. W., and Mooney, R. J. 2006. Learning for semantic parsing with statistical machine translation. In *Proceedings of HTL-NAACL*, 439–446.

Woods, W. A. 1978. Semantics and quantification in natural language question answering. *Advances in computers* 17(3).

Zettlemoyer, L. S., and Collins, M. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of EMNLP-CoNLL.*