Towards Building Robust Natural Language Interfaces to Databases

Michael Minock, Peter Olofsson, Alexander Näslund

Department of Computing Science
Umeå University, Sweden
Phone: +46 90 786 6398
Fax: +46 90 786 6398
Email: {mjm,c02pon,c02nar}@cs.umu.se

Abstract. We seek to give everyday technical teams the capability to build robust natural language interfaces to their databases, for subsequent use by casual users. We present an approach to the problem which integrates and streamlines earlier work based on light annotation and authoring tools. We model queries in a higher-order version of Codd's tuple calculus and we use synchronous grammars extended with lambda functions to represent semantic grammars. The results of configuration can be applied directly to SQL based databases with general n-ary relations. We have fully implemented our approach and we present initial empirical results for the GEOQUERY 250 corpus.

1 Introduction

One factor that has blocked the uptake of natural language interfaces (NLIs) to databases has been the economics of configuring such systems [2, 5]. Typically configuration requires high levels of knowledge and long time commitments. The typical work environment has neither of these in great supply and thus when presented with the choice of building a common forms based interface versus an NLI, organizations typically opt for forms. Our work seeks to make NLIs a more attractive option by reducing the time and expertise requirement necessary to build them.

Given the limited linguistic knowledge possessed by most technical teams, modern approaches to configuring NLIs to standard databases come down to one of three approaches:

1 Let authors only lightly *name* database elements (e.g. relations, attributes, join paths, etc.) and reduce query interpretation to graph match[3, 14].
2 Offer a GUI based authoring interface where one grunts out a *semantic grammar* that interprets user queries over their database. Such approaches are common in industrial products such as Microsoft's ENGLISHQUERY, Progress Software's EASYASK, Elfsoft's ELF, etc.
3 Use machine learning to induce a semantic grammar from a corpus of natural language query/correct logical interpretation pairs [16, 8, 6, 17, 18].

Since the ultimate goal of our project is the delivery of a high impact NLI to database tool, it should not be surprising that we primarily adopt an authoring approach. The contribution of the present paper is to present our formally rooted *name-tailor-define* authoring approach and to show that it may be effectively employed by normal technical personnel. That said, aspects of the first approach are deeply integrated into our work and we have laid the ground for an integration of machine learning techniques to help achieve highly robust NLIs 'in the limit' after experiencing large volumes of user queries.

1.1 Organization of this paper

Section 2 lays the foundation of concepts necessary to understanding our approach. Section 3 presents our formal approach to analyzing noun phrases, what we consider to be the primary challenge of NLIs to databases. Section 4 presents our GUI-based administration tool in which one populates the system with the formal elements described in sections 2 and 3. Section 5 presents an initial experiment that shows that reasonably skilled subjects can effectively use our authoring tool. Section 6 compares our approach with other approaches to building NLIs to databases. Section 7 concludes.

2 Foundations

2.1 Database elements, namings and the dictionary

While database design may initially involve UML or ER modeling, most ongoing work is at the *representational level* with the underlying relations, views, attributes, tuples and values of the working database. Assume the set of relations **REL** (e.g. `CITY`), attributes **ATT** (e.g. `CITY.NAME`) and data values **VAL** (e.g. `'Chicago'`). An additional element class that we include here are join conditions **JOIN**. These specify conditions commonly appearing in queries (e.g. `CITY.STATE = STATE.NAME`). Collectively the set of elements **ELEMENTS** = **REL** ∪ **ATT** ∪ **VAL** ∪ **JOIN**.

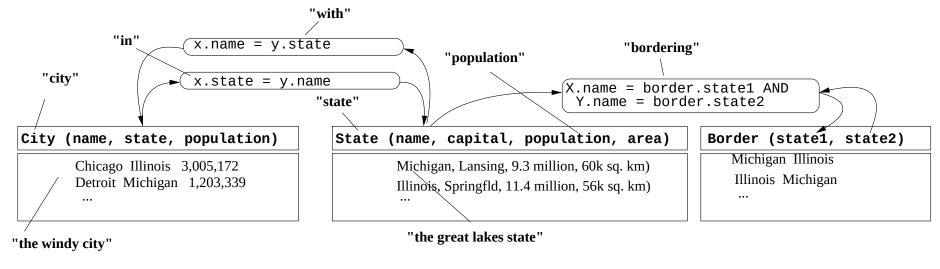

"with"
"in"
x.name = y.state
x.state = y.name
"city"
"state"
"population"
"bordering"
X.name = border.state1 AND
Y.name = border.state2
City (name, state, population)
State (name, capital, population, area)
Border (state1, state2)
Chicago Illinois 3,005,172
Detroit Michigan 1,203,339
...
Michigan, Lansing, 9.3 million, 60k sq. km)
Illinois, Springfld, 11.4 million, 56k sq. km)
...
Michigan Illinois
Illinois Michigan
...
"the windy city"
"the great lakes state"

Fig. 1. A fragment of the GEO250 schema with some namings

In figure 1 we see these elements being named with various phrases. If we consider the set of all alphanumeric strings to be Σ^*, this may be captured by a naming relation $\mathbf{NAMING} \subset \mathbf{ELEMENTS} \times \Sigma^*$ (e.g. $(\mathsf{CITY}, \text{“}\texttt{city}\text{”}) \in \mathbf{NAMING}$). The population of the naming relation may be accomplished by a mix of automated and manual methods, but a substantial amount of it is simply materialized from the underlying database (e.g. “Chicago” is a name for the database value `'Chicago'`.).

2.2 Queries in an extended tuple calculus

Because of the formal difficulties of working directly with SQL expressions, we represent queries as expressions in Codd's Tuple Calculus. The tuple calculus is a widely known syntactic sugar developed over standard first order logic where variables range over tuples within database relations, not over the individual places in n-ary relations. For example the query that expresses the cities in the states with more than 10 million people is $\{x|City(x) \wedge (\exists y)(State(y) \wedge x.state = y.name \wedge y.population > 10,000,000)\}$. Tuple calculus is covered in the majority of introductory database textbooks and may be directly mapped to standard SQL queries or expressions in first-order logic.

To handle various ranking and aggregation capabilities of SQL, we extend the tuple calculus with several higher-order capabilities. For example to support ranking and thus superlatives, we use higher-order predicates.

Example 1. (“cities of over 100,000 people in the largest area mid-western state”)

$$\begin{aligned}&\{x|City(x) \wedge x.population > 100000\wedge \\ &\quad (\exists y)(State(y) \wedge x.state = y.name\wedge \\ &\qquad LargestByArea(y, State(y) \wedge y.name \in \{'\text{Indiana}', ..., '\text{Wisconsin}'\})\}\end{aligned}$$

The two place predicate *LargestByArea* is true for the tuple y that has the greatest area that satisfies the supplied formula, $State(y) \wedge y.name \in \{'\text{Indiana}', ..., '\text{Wisconsin}'\}$ in the case here. We assume a set of built-in predicates to express superlative conditions over the various relations and numerical attributes (e.g. There is a higher order predicate $LargestByHeight(y, \phi(y))$ which is true for that value of y with the largest height that satisfies $\phi(y)$. Naturally this predicate is only meaningful for mountain tuples.) These expressions have straightforward mappings to SQL for database systems that robustly support sub-selects.

Example 2. (Example 1 translated to SQL for PostgreSQL)

```
SELECT *
FROM City AS x
WHERE x.population > 100000 AND
  EXISTS (SELECT * FROM
           (SELECT * FROM State AS z
            WHERE z.name = 'Indiana' OR ... OR z.name = 'Wisconsin'
            ORDER BY area DESC LIMIT 1) AS y
      where x.state = y.name);
```

2.3 Semantic grammars in λ-SCFG

Following [17], we model our semantic grammars as *synchronous context-free grammars* [1] augmented with lambda calculus expressions (λ-SCFG). Our λ-SCFG rules define two 'synchronous' trees derived from the same start symbol S. The yield of the first tree is natural language, and the yield of the second tree is a formal language expressing the semantics of the natural language yield of the first tree. The requirement of having variables within the semantic formulas necessitates the use of λ expressions and in turn (slightly) complicates the notion of what a yield is for the second tree – yields are calculated bottom up and involve the well known alpha conversion and beta reduction operations of lambda calculus.

Formally, each λ-SCFG rule has the form:

$$A \rightarrow \langle \alpha, \lambda x_1, ..., \lambda x_k \beta \rangle \tag{1}$$

where A is a single non-terminal symbol and α is a sequence of terminal and non-terminal symbols where the terminals are words or word sequences in natural language. β consists of a sequence of terminals, non-terminals and formal argument terms which compose arguments $x_1, ..., x_k$. We say compose here, because in contrast to [17], we shall at times pass lambda functions in as arguments. In such case we shall use the symbol f_i in the place of x_i. The terminals within β consist of raw material used to build up formal expressions. Each use of a non-terminal symbol in α is paired with the use of the same symbol in β (and vice versa). In cases that there are multiple uses of the same non-terminal in α, one must include index terms to specify which non-terminal in α corresponds to which in β. The paper [17] gives a more complete characterization of how such λ-SCFG rules are used to translate a natural language input to a formal semantic expression.

3 Focusing on Noun Phrases

Through earlier and ongoing work [10, 12], we posit that the main analysis task for natural language interfaces to databases is the analysis of noun phrases, often with relative clauses. In fact more experienced users often type just noun phrases to describe the information they would like retrieved. In addition our experience tells us that we can model noun phrases as coordinated pre-modifiers and post-modifiers around head nouns. In the notation of λ-SCFG some of the general rules that guide this process are:

$QUERY \rightarrow \langle \text{“}\texttt{list the}\text{”} \cdot NP, answer(\{x | NP(x)\}) \rangle$
$NP \rightarrow \langle PRE \cdot NP, PRE(NP) \rangle$
$NP \rightarrow \langle NP1, NP1 \rangle$
$NP1 \rightarrow \langle NP1 \cdot POST, POST(NP1) \rangle$
$NP1 \rightarrow \langle HEAD, HEAD \rangle$

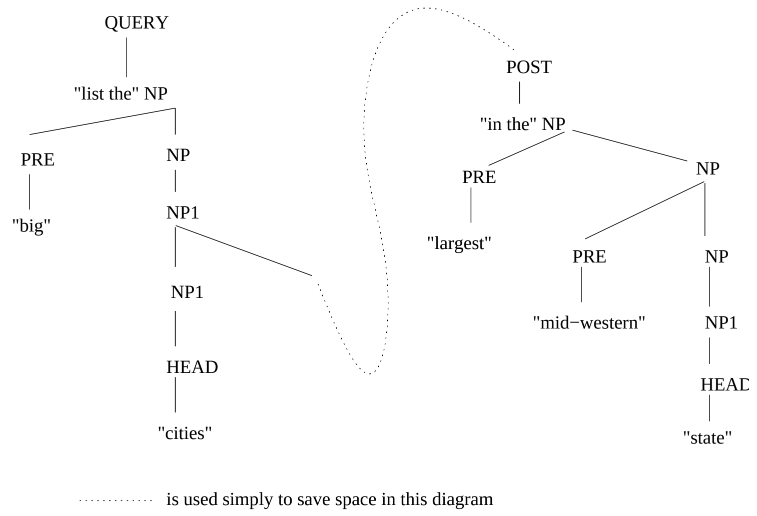


Fig. 2. An example parse of "list the big cities in the largest mid-western state."

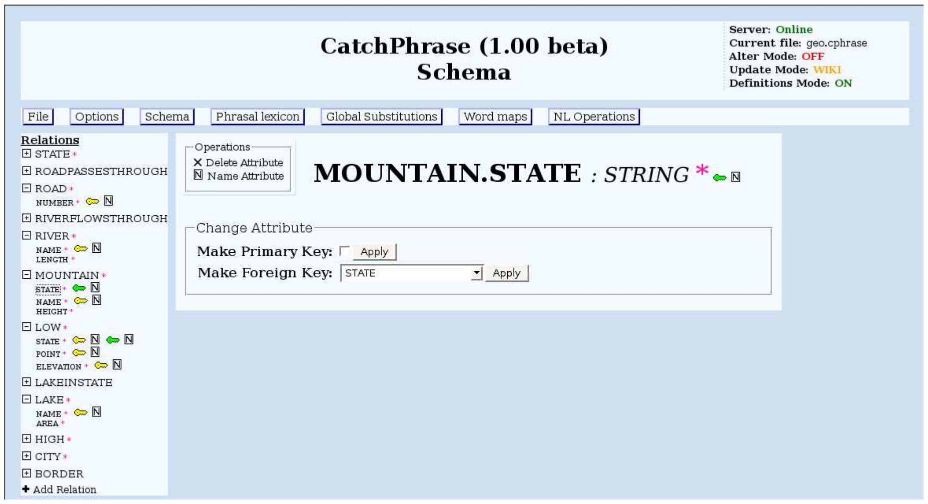
The first rule expresses what we call a *sentence pattern*, of which there are many variants. This particular rule enables users to type "list the X" where X is a noun phrase. There are many other sentence patterns that let the user type things like "give me the NP" or "what are the NP?", etc. In total the system has around 75 manually defined sentence patterns and rarely do we find users typing expressions not covered by these basic patterns. Of course as we discover new ones, we, as system designers, simply add them in. The last four rules above are more interesting and enable noun phrases such as those in figure 2.

The authoring process provides the lexical entries that define pre-modifiers, heads and post-modifiers of noun phrases for the given database. Assume the following entries are built over our specific geography database:

$HEAD \rightarrow \langle \text{``}\texttt{cities}\text{''}, \lambda x.City(x)\rangle$
$HEAD \rightarrow \langle \text{``}\texttt{state}\text{''}, \lambda x.State(x)\rangle$
$PRE \rightarrow \langle \text{``}\texttt{big}\text{''}, \lambda f.\lambda x.f(x) \wedge City(x) \wedge x.population > 100,000\rangle$
$POST \rightarrow \langle \text{``}\texttt{in the}\text{''} \cdot NP,$
$\quad \lambda f.\lambda x.f(x) \wedge x.City(x) \wedge (\exists y)(State(y) \wedge x.state = y.name \wedge NP(y)\rangle$
$PRE \rightarrow \langle \text{``}\texttt{largest}\text{''},$
$\quad \lambda f.\lambda x.\mathbf{LargestByPop}(x, f(x) \wedge State(x))\rangle$
$PRE \rightarrow \langle \text{``}\texttt{largest}\text{''},$
$\quad \lambda f.\lambda x.\mathbf{LargestByArea}(x, f(x) \wedge State(x))\rangle$
$PRE \rightarrow \langle \text{``}\texttt{mid-western}\text{''}, \lambda f.\lambda x.$
$\quad f(x) \wedge State(x) \wedge x.name \in \{'\text{Indiana}', ..., '\text{Wisconsin}'\}\rangle$

The reader may wish to verify that "list the big cities in the largest mid-western state" parses to the tree in figure 2, which evaluates to the expression *answer*(Q) where Q is the expression in example 1, which in turn is automatically translated to the SQL of example 2.

4 Our GUI-based Authoring Tool


CatchPhrase (1.00 beta)
Schema

Server: Online
Current file: geo.cphrase
Alter Mode: OFF
Update Mode: WIKI
Definitions Mode: ON

Relations
STATE *
ROADPASSESTHROUGH
ROAD *
NUMBER *
RIVERFLOWSTHROUGH
RIVER *
NAME *
LENGTH *
MOUNTAIN *
STATE *
NAME *
HEIGHT *
LOW *
STATE *
POINT *
ELEVATION *
LAKEINSTATE
LAKE *
NAME *
AREA *
HIGH *
CITY *
BORDER
Add Relation

MOUNTAIN.STATE : *STRING* *

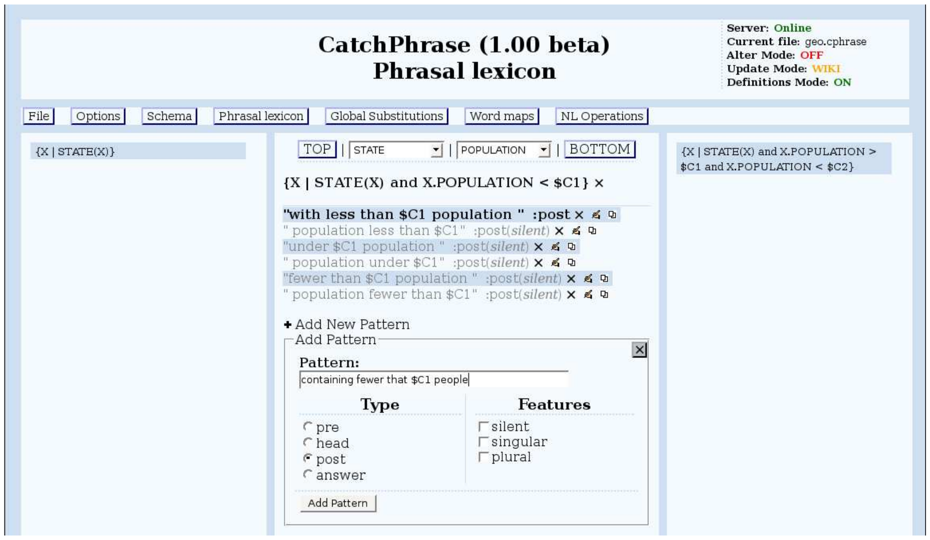
Change Attribute
Make Primary Key: ☐
Make Foreign Key: STATE

Fig. 3. Naming database elements over the schema.

Our AJAX based administrator tool gives an integrated GUI through which one may import a schema from any ODBC accessible database and commence what we term the *name-tailor-define* cycle of authoring an NLI.

Naming actions provide simple text names for the relations, attributes and join paths of the database schema as well as operations to provide additional names to values materialized from the underlying database. In short one populates the **NAMING** relation of section 2.1. Figure 3 shows the schema browser in our tool where the user has the option of exploring and naming the various database elements. Note the option here of naming the attribute `MOUNTAIN.STATE`. Note also that in this image the foreign key joins may be clicked on to name join elements. Naming actions trigger the definition of default head, pre-modifier and post-modifier lexical rules. These default rules are formed by coupling logical expressions over the join graph of the database, with associated terms in the **NAMING** relation. The default glosses of these elementary logical expression include function words (e.g. 'of','and',etc.) from the underlying language, English in our case.

To manage comprehensibility and navigation, default rules are gathered into a collection of *entries* that couple a single elementary conceptual expression with a set of m-patterns. Under the hood this expresses m lexical rules, but to the user of the author there is one entry with m patterns. Using a theorem prover these entries are sorted into a subsumption hierarchy for ease of navigation. See [10, 12] for a detailed description of this process.



CatchPhrase (1.00 beta)
Phrasal lexicon

Server: Online
Current file: geo.cphrase
Alter Mode: OFF
Update Mode: WIKI
Definitions Mode: ON

{X | STATE(X)}

STATE | POPULATION

{X | STATE(X) and X.POPULATION < $C1}

"with less than $C1 population " :post
" population less than $C1" :post(*silent*)
"under $C1 population " :post(*silent*)
" population under $C1" :post(*silent*)
"fewer than $C1 population " :post(*silent*)
" population fewer than $C1" :post(*silent*)

Pattern: containing fewer that $C1 people

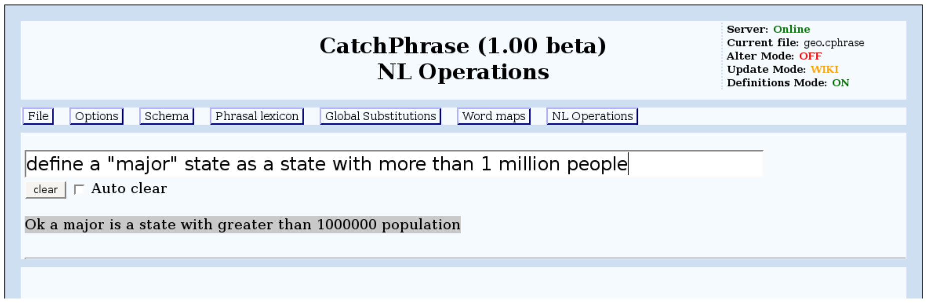
Type	Features
pre	silent
head	singular
[✓ post]	plural
answer	

{X | STATE(X) and X.POPULATION > $C1 and X.POPULATION < $C2}

Fig. 4. Tailoring entries.

During *Tailoring* one works with patterns that are associated with parameterized concepts expressed in tuple calculus. For example in figure 4 the currently selected entry corresponds to the concept of a state under a certain population. Note that in this case an additional linguistic pattern is being associated with this concept, for a total of seven patterns. Note also that the patterns may be used in the reverse direction to paraphrase queries to clarify to users that the system understood (or misunderstood) their questions. See [11] for an in-depth discussion of this paraphrase generation process.

Once a fair bit of structure has been built up, definition actions may be performed in which the author creates conceptually more specific entries. Figure 5 shows an example of an action in which a new concept that corresponds to states with a population over 10 million is defined. One continues with the *name-tailor-define* process over the lifetime of the interface. Our administrator tool includes special viewers and editors that assist in analyzing the logs of casual user interactions and to patch leaks in the configuration on an on-going basis.


CatchPhrase (1.00 beta)
NL Operations

Server: Online
Current file: geo.cphrase
Alter Mode: OFF
Update Mode: WIKI
Definitions Mode: ON

define a "major" state as a state with more than 1 million people

Ok a major is a state with greater than 1000000 population

Fig. 5. Defining additional concepts via natural language.

5 Experiments

The experiments we present here are based on the Geoquery 250 test corpus provided by Raymond Mooney's group at the University of Texas, Austin. The corpus is based on a series of questions gathered from undergraduate students over an example US geography database originally shipped with a Borland product. In addition to the raw geography data, the corpus consists of natural language queries and equivalent logical formulas in Prolog for 250 queries.

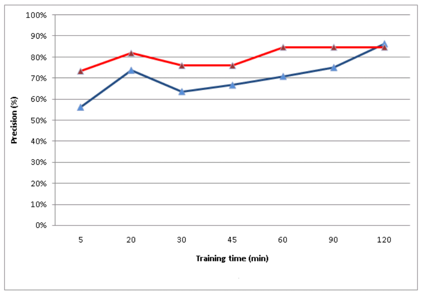
Although we have informally verified that our authoring interface is usable for normal technical users, unfortunately we have only run our full experiment with two subjects. The subjects were two under-graduate computer science students that had recently taken an introductory relational database course. The subjects were instructed to read the user's manual of our administration tool and were trained on its use. The training examples were over several mocked up canonical examples and were in no way related to the Geoquery 250 corpus.

After being trained on the system, subjects were presented with a random 'training set' of 100 of the 250 Geoquery 250 queries. Subjects were told that they needed to author the system to cover such queries. Of the remaining 150 queries in the corpus, 33 queries were selected to be in our test set. For these 33 queries correct logical queries were manually constructed in our extended tuple calculus representation. As a side note, this process took slightly over 1 hour, thus our first finding is that on a corpus of the complexity of the Geoquery 250 corpus, a skilled worker will take approximately 2 minutes per natural language query to write and test the equivalent logical query.

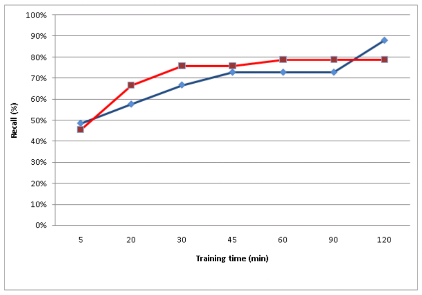
As our subjects worked to cover the 100 queries of the training set, their configurations were saved off at intervals for future analysis. This continued for a two hour time span. Afterwards we automatically tested the resulting sequences of configuration files against the 33 queries of the test set. Exploiting our capability to determine logical query equivalence, queries were marked as correct if their parse to logical form was equivalent to the manually constructed correct result. In the rare case of ambiguity (e.g. "largest state") the answer was marked correct if one of its results was equivalent to the manually constructed correct result. Each configuration file evaluation yielded a measure of *precision*, *recall*[1] and *accuracy* where:

$$\text{precision} = \frac{\#\text{ of correct queries}}{\#\text{ of parsed queries}};\ \text{recall} = \frac{\#\text{ of parsed queries}}{\#\text{ of queries}} \tag{2}$$

$$\text{accuracy} = \frac{\#\text{ of correct queries}}{\#\text{ of queries}} \tag{3}$$



(a) Precision



(b) Recall

Fig. 6. Precision and Recall measures for our initial two trials.

Figure 6 shows the resulting precision and recall measures through time.

6 Related Work

Due to the public availability of GEOQUERY 250 corpus, we can compare our initial results to several machine learning approaches [16, 8, 6, 17, 18], an approach based on light annotation [14] and an authoring approach over Microsoft's EnglishQuery product (described in [14]).

In comparing our results with machine learning approaches, we focus on results obtained after 120 minutes of effort. Since our informal finding of 2 minutes preparation time for each query in the training set, we thus focus on results with training sets of size 60. The results for such small training sets are not very strong. For example the accuracy of λ-WASP, the latest and currently best performing system developed by the group at the University of Texas, appears to be slightly under 50% with 60 queries in the test set (precision was slightly under

[1] We adopt the definition of recall presented in [14]. Unfortunately terms have not been consistently used across the literature on NLI to database evaluation. For example our measure of accuracy corresponds to recall as defined in the UT group's results. We adopt the definition of terms in [14], because they are reminiscent of the trade offs in standard information retrieval between recall and precision. In any case as we compare our results to others, we shall present their results in our terms.

80%, thus in our terminology recall was approximately 60%). In our experiments subjects average slightly under 80% correctness after 120 minutes of authoring with an average precision of 86%. Also of interest is to look at asymptotic results when training samples grow to essentially unbounded size. In these cases, machine learning results are much stronger. The asymptotic precision of λ-WASP appears to be approximately 91.95% with an accuracy of 86.59%, yielding, in our terminology a recall of 94%. Another machine learning experiment over a relaxed-CCG approach obtained similar results [18].

Our comparisons with machine learning approaches highlight a bootstrapping weakness that if overcome would probably make machine learning approaches dominant. The weakness is of course the cost of obtaining the corpus of natural language/logical expression pairs. Mooney talks briefly about an approach to this problem for simulated environments where descriptions in natural language are paired with formal representations of objects and events retained from the simulation [13]. He suggests simulated RoboCup where a commentator describes game events as an interesting test-bed. Our proposed approach, focussed as it is on just NLIs to databases, envisions authors making equality statements between natural language queries. For example one may assert that "What are the states through which the Longest river runs" means "states with the longest river". If the system is able to obtain a correct parse of the second query, it can associate that with the earlier natural language question and use this as a basis to induce extra lexical rules that make the NLI more robust. Although our approach always requires some initial bootstrapping before such machine learning can engage, this paper has shown that the labor involved in such bootstrapping can be of reasonable cost. The thesis is that in the long run this will lead to systems approach 100% precision and recall for the queries that are issued to the system[2]

PRECISE[14] is a system based on light annotation of the database schema that was tested over the Geoquery 250 corpus. PRECISE reduces semantic analysis to a graph matching problem after the schema elements have been named. Interestingly the system leverages a domain independent grammar to extract attachment relationships between tokens in the user's requests. The PRECISE work identifies a class of so called *semantically tractable queries*. Although the group did not publish the actual configuration times, they presumably corresponded to the naming phase and thus were rather short durations. We will forgo a discussion of the so called *semantically tractable queries* class and take at face value the claim that they achieved 100% precision and 77.5% recall, yielding a correctness of 77.5%. For such little configuration this is an impressive result and over very simple databases with a stream of very simple queries this may be adequate. However experience tells us that users do actually ask somewhat complex queries at times and they will be frustrated if told that their queries are not semantically tractable and must be rephrased or abandoned.

The PRECISE group reported a side experiment in which a student took over 15 hours to build a Geoquery 250 NLI using Microsoft's EnglishQuery

[2] This does not address larger issues such as the limitations of the underlying formal language nor users querying for information outside the scope of the database.

tool. The resulting system achieved rather poor results for such an expensive effort – approximately 80% precision and 55% recall, yielding a correctness of approximately 45%. Our limited experience with the Microsoft English query tool was rather frustrating as well, but it would be interesting to repeat this experiment and also see how other commercial systems such as Progress Software's EasyAsk and Elfsoft's ELF fare.

It should be noted that historically, transportable systems (e.g.,[7]) arose as a proposed solution to the high configuration costs of NLIs to databases. The idea is to use large scale domain-independent grammars, developed in the linguistics community, to map user requests to intermediate *logical form*. Using *translation knowledge*, the logical form is then translated to a logical query expressed in the vocabulary of the relations of the actual database. Hence building an interface over a new database requires supplying a set of domain-specific lexical entries and the specification of translation knowledge, but does not require new linguistic syntax rules to be defined. A serious problem with the transportable approach however, is that it requires a deep understanding of the particular logical form employed to specify working and robust translation knowledge to the actual database tables in use. Considering the knowledge and tastes of the average technical worker, one wonders how well this can be supported. This said, some very interesting work has recently taken up the transportable approach for authoring NLIs over OWL and F-logic knowledge-bases [4]. Like this work, that work assumes very little computational linguistics knowledge on the part of the person who builds the natural language interface.

The system DUDE[9] presents an easy to use authoring interface to build dialogue systems. The back-end database is essentially a universal relation with relatively simple slot filling queries, but the authoring method is elegantly direct and results are sufficient for many practical applications.

7 Conclusions

This paper has presented a state-of-the-art authoring system for natural language interfaces to relational databases. Internally the system uses semantic grammars encoded in λ-SCFG to map typed user requests to an extended variant of Codd's tuple calculus which in turn is automatically mapped to SQL. The author builds the semantic grammar through a series of naming, tailoring and defining operations within a web-based GUI. The author is shielded from the formal complexity of the underlying grammar and as an added benefit, the given grammar rules may be used in the reverse direction to achieve paraphrases of logical queries (see [11]).

Using the Geoquery 250 corpus, our results are compared with contemporary machine learning results and approaches based on light annotation. Our initial experimental evidence shows quick bootstrapping of the initial interface can be achieved. Future work will focus on more complete evaluation and experimentation with more comprehensive grammatical frameworks (e.g. CCG [15]), especially under regimes that enable enhanced robustness [18]. Future work will also explore hybrid approaches using initial authoring to bootstrap NLIs, followed by interactive machine learning that, applied in the limit, will edge such NLIs toward 100% precision and recall.

References

1. A. Aho and J. Ullman. *The Theory of Parsing, Translation and Compiling*, volume 1. Prentice-Hall, Englewood Cliffs, NJ, 1972.
2. I. Androutsopoulos and G.D. Ritchie. Database interfaces. In R. Dale, H. Moisl, and H. Somers, editors, *Handbook of Natural Language Processing*, pages 209–240. Marcel Dekker Inc., 2000.
3. W. Chu and F. Meng. Database query formation from natural language using semantic modeling and statistical keyword meaning disambiguation. Technical Report 990003, UCLA Computer Science Department, June 1999.
4. P. Cimiano, P. Haase, and J. Heizmann. Porting natural language interfaces between domains: an experimental user study with the orakel system. In *Intelligent User Interfaces*, pages 180–189. ACM, 2007.
5. A. Copestake and K. Sparck Jones. Natural language interfaces to databases. *The Natural Language Review*, 5(4):225–249, 1990.
6. R. Ge and R. Mooney. A statistical semantic parser that integrates syntax and semantics. In *Proceedings of the Ninth Conference on Computational Natural Language Learning*, pages 9–16, 2005.
7. B. Grosz, D. Appelt, P. Martin, and F. Pereira. Team: An experiment in the design of transportable natural-language interfaces. *AI*, 32(2):173–243, 1987.
8. R. Kate and R. Mooney. Using string-kernels for learning semantic parsers. In *Proc. of COLING/ACL-2006*, pages 913–920, 2006.
9. O. Lemon and X. Liu. DUDE: a Dialogue and Understanding Development Environment, mapping Business Process Models to Information State Update dialogue systems. In *Proceedings of EACL (demonstration systems)*, 2006.
10. M. Minock. A phrasal approach to natural language access over relational databases. In *Proc. of Applications of Natural Language to Data Bases (NLDB)*, pages 333–336, Alicante, Spain, 2005.
11. M. Minock. Modular generation of relational query paraphrases. *Research on Language and Computation*, 4(1):1–29, 2006.
12. M. Minock. A STEP towards realizing Codd's vision of rendezvous with the casual user. In *33rd International Conference on Very Large Data Bases (VLDB)*, Vienna, Austria, 2007. Demonstration session.
13. R. Mooney. Learning language from perceptual context: A challenge problem for ai. In *Proceedings of the 2006 AAAI Fellows Symposium*, 2006.
14. A. Popescu, O. Etzioni, and H. Kautz. Towards a theory of natural language interfaces to databases. In *Intelligent User Interfaces*, 2003.
15. M. Steedman. *The Syntactic Process.* The MIT Press, 2001.
16. L. Tang and R. Money. Using multiple clause constructors in inductive logic programming for semantic parsing. In *Proc. of ECML*, 2001.
17. Y. Wong and R. Mooney. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL-2007)*, pages 960–967, 2007.
18. L. Zettlemoyer and M. Collins. Online learning of relaxed ccg grammars for parsing to logical form. In *Proceedings of the 2007 Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2007.