C-Phrase: A System for Building Robust Natural Language Interfaces to Databases

Michael Minock
Department of Computing Science Umeå University, Sweden
Phone: +46 90 786 6398
Fax: +46 90 786 6398
Email: mjm@cs.umu.se

Abstract
This article presents C-PHRAKE, a natural language interface system that can be configured by normal, non-specialized, web-based technical teams. C-PHRAKE models queries in an extended version of Codd's tuple calculus and uses synchronous context free grammars with lambda-expressions to represent semantic grammars. Given an arbitrary relational database, authors rapidly build an NLI using what we term the name-tailor-define protocol. We present a small study demonstrating the effectiveness of this approach for the GEO corpus and we introduce the evaluation metric of willingness that complements the standard metrics of precision and recall. However our true evaluation comes as we open-source C-PHRAKE.

1. Introduction

Although the advantages of interacting with computers in natural language (e.g., Swedish or English) are easy to recount (e.g., humans already know natural language, natural language is capable of expressing nuanced semantics, speech interfaces are eyes-free/hands-free, etc.), in practice such systems have enjoyed only limited success [3, 6]. Typically such systems are characterized by being brittle and unpredictable [21]. Slight re-phen clos command result in failures, users have difficulty comprehending system coverage, etc. Although NLIs to databases have fallen out of focus, many developments over the last 20 years make another pass at the general problem promising (e.g., maturation of relational technology and theory [1], availability of extensive lexical resources [13], high performance theorem provers [25], advances in machine learning, improvements in speech recognition [10], etc.).

One factor that has blocked the uptake of natural language interfaces (NLIs) to databases has been the economics of configuring such systems [3, 6]. Typically configuration requires high levels of linguistic expertise 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 requirements necessary to build them.
Given the limited linguistic expertise possessed by most technical teams, modern approaches to configuring NLIs to standard relational 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 [4, 20].

2. Offer a GUI based tool where an author builds a semantic grammar\(^1\) that interpret natural language queries over databases.

3. Use machine learning to induce a semantic grammar from a corpus of natural language query/correct logical interpretation pairs [24, 11, 7, 26, 27].

Since our ultimate goal is the delivery of a high impact NLI system, it should not be surprising that we primarily adopt an authoring approach. 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.

There are three main assumptions that underlie C-Phrase. The first is that standard, off-the-shelf relational databases are sufficiently expressive to back NLIs. The second is that any approach must include a generation component that can accurately paraphrase database queries back to the user in natural language\(^2\). Such a capability lets the user decide ambiguous analyzes and gives them a feeling of control, reducing the anxiety of being being misunderstood by the system. The third assumption is that while users can restrict their inquiries to the closed domain of a given database, they will issue queries and commands of a sort that is highly elliptical, idiosyncratic, with spelling (or speech recognition) errors, often lacking proper syntactic structure (e.g., “energy stocks down last year up 10% this year”). Thus analysis components must be robust, seeking out near misses when input is less than ideal and the system must be adaptable, making it easy for authors to patch running systems to catch unanticipated phrasings\(^3\).

This article looks at the total problem of configuring and evaluating NLIs to databases from the system author’s perspective. Although the primary effort goes into configuring the parser and generator to associate phrases with database elements, there are other tricky issues with respect to problems such as spell-checking and, more generally the leveraging of additional linguistic resources for enhanced robustness.

### 1.1. Organization of this article

This article is an extension of an earlier conference paper [17] and holds a similar structure, but presents at greater depth. Section 2 lays the foundation of concepts necessary to understand our approach. This includes a formalization of naming databases elements, queries of the sort we handle and finally some basic material on synchronous

---

\(^1\) Semantic grammars [10] map natural language phrases directly to the tables in the underlying database. As such they are not transportable between databases. Such approaches are common in industrial products such as Microsoft’s ENGLISHQUERY, Progress Software’s EASYASK, Elfsoft’s ELF, etc.

\(^2\) A further economy of representation assumption is that the generation and analysis component share the same lexicon.

\(^3\) The name C-Phrase is derived from ‘catch phrase’, which we interpret literally as ‘catching phrases’.
context-free grammars enriched with lambda expressions. Section 3 presents our formal approach to analyzing noun phrases, what we consider to be the primary challenge of NLIs to databases. The treatment is based on a limited derivative of X-bar theory [9] encoded in synchronous context-free grammars [2] augmented with lambda calculus expressions (λ-SCFG). Section 4 presents our GUI-based authoring tool through which an author populates the system with the formal elements described in sections 2 and 3. Section 5, which is new in this article, provides a detailed description of the processing steps that C-PHRASE goes through to respond to user’s typed input. Section 6 describes the overall methods of evaluating NLIs to databases and presents a small study confirming that reasonably skilled subjects can effectively use C-PHRASE’s authoring tool. Section 6 also introduces a new evaluation metric termed willingness that complements the classic notions of precision and recall. Section 7 compares our approach with other approaches to building NLIs to databases and section 8 concludes.

2. Foundations

At it’s base, this work rests on relational databases – assuming authors (and readers) to have a knowledge of primary and foreign keys and Codd’s tuple calculus. There are many excellent sources covering these concepts, ranging from detailed theoretical treatments [1] to more conceptual undergraduate textbooks. Perhaps due to substantial growth within the database field, many recent textbooks have skipped Codd’s tuple calculus. We consider this to be unfortunate with respect to NLIs to databases. For SQL is too informal, standard first order logic (i.e. domain calculus) is unnecessarily verbose, relational algebra is too procedural and DATALOG spreads queries over multiple rules in complex ways. The tuple calculus is a compact, declarative formalism in which many user queries can be expressed as single formulas.

2.1. Relations, attributes, values, join paths, and namings

While database design may initially involve UML or ER modeling, most on-going work is at the representational level with the underlying relations, views, attributes, tuples and values of the working database. The process of naming associates words or phrases with such database elements. Assume the set of relations \( R \) (e.g. \( \text{CITY} \in R \)), attributes \( A \) (e.g. \( \text{CITY.NAME} \in A \)) and data values \( V \) (e.g. ‘Chicago’ \( \in V \)). An additional class of elements that we consider are equi-join paths \( J \) over the underlying schema. Collectively the set of database elements is \( E = R \cup A \cup V \cup J \). Elements are named by associating various words or phrases with them. In figure 1 we see elements being named with various phrases. If we consider the set of all alphanumeric strings to be \( \Sigma^* \), this may be captured by a naming relation \( N \subset E \times \Sigma^* \) (e.g. \( \text{CITY,“city”} \in N \)). The creation 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’).

Although relations, attributes and values may be immediately comprehended, equi-join and, more generally, paths of equi-joins, deserve a bit more explanation. These specify conditions commonly appearing in queries that span more than one relation (e.g. \( \text{CITY.STATE} = \text{STATE.NAME} \)). Normally such joins are formed via foreign key definitions in the underlying schema. So, for example, if the attribute \( \text{CITY.STATE} \) is a foreign
key to the `STATE` table, and `STATE` has the primary key `STATE.NAME`, then normally the equi-join `CITY.STATE = STATE.NAME` ∈ J. Note that for purposes of naming, order matters. Thus the equi-join `CITY.STATE = STATE.NAME` is not the same as the equi-join `STATE.NAME = CITY.STATE` (see figure 1). This accounts for the difference between phrases such as “cities in Michigan” and “states with cities with more than 2 million people”.

We generalize these notions to consider paths of equi-joins as forming multi-table complex joins. For example if we have a table that expresses a binary bordering relation between two states, then `STATE.NAME = BORDER.STATE1` followed by `BORDER.STATE2 = STATE.NAME` is such a join path that expresses ‘bordering’. Of course if one were to instantiate such a join, then it would be important to alias separate variables to state in the first as well as the second equi-join. Formally an equi-join path of length n is the sequence \[ R_{\phi(1)}A_{\theta(1)} = R_{\phi(2)}A_{\theta(2)} = \ldots = R_{\phi(n)}A_{\theta(n)} \] where \( \phi \) and \( \varphi \) map to relation identifiers, \( \theta \) and \( \varphi \) map to attribute identifiers, and \( \varphi(i) = \phi(i+1) \) for \( 1 \leq i < n \). Currently C-PHRASE includes in J all join paths of length 3 or less that are built up from primitive foreign key based equi-joins. The naming process enables system authors to attach phrases to such join paths.

Figure 1: A fragment of the GEOQUERY Corpus schema with some namings

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 well 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) ∧ (∃y)(State(y) ∧ x.state = y.name ∧ y.population > 10,000,000)\}. Tuple calculus may be directly mapped to 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")

\{x|City(x) ∧ x.population > 100000 ∧ (∃y)(State(y) ∧ x.state = y.name ∧ LargestByArea(y, State(y) ∧ y.population ∈ \{'Indiana', 'Michigan', 'Wisconsin'\}))\}
The two-place predicate \( \text{LargestByArea} \) is true for the tuple \( y \) that has the greatest area that satisfies the supplied formula, \( \text{State}(y) \land \text{y.name} \in \{ \text{Indiana}', ..., \text{'Wisconsin}' \} \) in the case here. We stipulate 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 \( \text{LargestByHeight}(y, \phi(y)) \) 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.) Each numeric attribute automatically induces such a higher-order superlative predicates. These expressions have straightforward mappings to SQL for database systems that robustly support sub-selects.

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

```sql
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);
```

We model aggregation and projection as functions that apply to tuple sets. Thus we denote the total area of all states as \( \text{sumArea}\{x|\text{State}(x)\} \). We denote the names and areas of the states as \( \text{proj}\{\text{name, area}, \{x|\text{State}(x)\}\} \). Note that we use this notation, rather than the normal dot notation of standard tuple calculus.

### 2.3. Semantic grammars in \( \lambda \)-SCFG

Following [26], we model our semantic grammars as synchronous context-free grammars [2] augmented with lambda calculus expressions (\( \lambda \)-SCFG). Our \( \lambda \)-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 formula 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 \( \lambda \) 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 \( \lambda \)-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 \( \alpha \) is a sequence of terminal and non-terminal symbols where the terminals are words or word sequences in natural language. \( \beta \) 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 [26], we shall at times pass lambda functions in as arguments\(^4\). In such case we shall use the symbol \( f_i \) in the place of \( x_i \). The terminals within \( \beta \) consist of raw material used to build up

\(^4\text{We restrict such lambda functions to single applications to avoid expressing the Y-combinator.}\)
formal expressions. Each use of a non-terminal symbol in $\alpha$ is paired with the use of the same symbol in $\beta$ (and vice versa). In cases that there are multiple uses of the same non-terminal in $\alpha$, we use indexes to specify which non-terminal in $\alpha$ corresponds to which in $\beta$. The example in the next section gives a more complete characterization of how such $\lambda$-SCFG rules are used to translate a natural language input to a formal semantic expression.

3. Focusing on Noun Phrases

```
QUERY
 "list the" NP

PRE
 "big"
 NP
 NP1
 HEAD
 "cities"

POST
 "in the" NP

PRE
 "largest"
 NP
 NP1
 HEAD
 "state"
 "mid-western"
```

... is used simply to save space in this diagram

Figure 2: An example parse of “list the big cities in the largest mid-western state.”

Through earlier and ongoing work [14, 16], we posit that the main analysis task for natural language interfaces to databases is the analysis of noun phrases, often with modifying adjectival phrases, propositional phrases, relative clauses, and modifying gerund constructions. 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, essentially basing our linguistic formalism on a limited derivative of X-bar theory [9]. In the notation of $\lambda$-SCFG some of the general rules that guide this process are:

\[
\begin{align*}
QUERY & \rightarrow \{ \text{“list the”} \cdot NP, \text{answer} \{ \{ x \mid NP(x) \} \}\} \\
NP & \rightarrow \{ \text{PRE} \cdot NP, \text{PRE}(NP) \}
\end{align*}
\]
NP → (NP1,NP1)
NP1 → (NP1 · POST, POST(NP1))
NP1 → (HEAD,HEAD)

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. The system has a series of context-free rules that manually define sentence patterns and rarely do we find users typing expressions not covered by these rules. 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:

\[
\text{HEAD} \rightarrow \langle \text{“cities”}, \lambda x.\text{City}(x) \rangle \\
\text{HEAD} \rightarrow \langle \text{“state”}, \lambda x.\text{State}(x) \rangle \\
\text{PRE} \rightarrow \langle \text{“big”}, \lambda f.\lambda x.f(x) \land \text{City}(x) \land x.\text{population} \geq 100,000 \rangle \\
\text{POST} \rightarrow \langle \text{“in the”} \cdot \text{NP}, \\
\lambda f.\lambda x.f(x) \land x.\text{City}(x) \land (\exists y)(\text{State}(y) \land x.\text{name} = y.\text{name} \land \text{NP}(y)) \rangle \\
\text{PRE} \rightarrow \langle \text{“largest”}, \\
\lambda f.\lambda x.\text{LargestByPop}(x, f(x) \land \text{State}(x)) \rangle \\
\text{PRE} \rightarrow \langle \text{“largest”}, \\
\lambda f.\lambda x.\text{LargestByArea}(x, f(x) \land \text{State}(x)) \rangle \\
\text{PRE} \rightarrow \langle \text{“mid-western”}, \lambda f.\lambda x. \\
f(x) \land \text{State}(x) \land x.\text{name} \in \{\text{‘Indiana’}, \ldots, \text{‘Wisconsin’}\} \rangle
\]

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 \(\text{answer}(Q)\) 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

We have developed an AJAX based authoring tool that gives an integrated GUI through which an author 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. In short one populates the naming relation \((N)\) 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. The attributes and relations with asterisks have already been named, and the user is in the process of naming the join from \text{CITY} to \text{STATE}. Naming actions trigger the definition of default head, pre-modifier and post-modifier lexical rules (exactly the types of rules presented in section 3). These default rules are formed by coupling logical expressions over the database elements, with associated words and phrases from the naming relation. The default glosses of these
elementary logical expression include closed category words (e.g. ‘of’, ‘and’, etc.) from the underlying language, English in our case. They also, depending on matches, may include alternate phrasings obtained through synonym searches using additional lexical resources (e.g. [13]) or stemming routines. Note we posit that naming time, not query time, is the most appropriate time at which to utilize such domain independent resources and techniques; inevitable erroneous generalizations are better reflected in structures that the author can view, control and correct.

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 (see figure 4). In the underlying formalism this expresses \( m \) lexical rules, but to 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 [14, 16] for a detailed description of this process. 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 with an area greater than a certain amount. In this case the additional linguistic pattern “states covering more than \( X \) square kilometers,” is being associated with this concept. Patterns may be used in the reverse direction to paraphrase queries to clarify to users that the system understood (or misunderstood) their questions. See [15] for an in-depth discussion of this paraphrase generation process.

Once a fair bit of structure has been built up, definitions 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 1 million is defined. One continues with the name-tailor-define process over the lifetime of the interface. Our authoring 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.

One final bit of configuration that should be mentioned is the specification of substitutions. Substitutions are domain specific and should always be applied to the user’s input string. Figure 6 shows the screen in which such substitutions are specified. This includes phrases such as “in the country”. Since such a phrase is always redundant in the given context, it is important to be able to blot it out and not expect that the parser will be able to meaningfully parse it.

5. Processing

Now that we have described the main representations underlying C-PHRASE, we shall describe the processing steps taken when a user inputs a string. Figure 7 shows the flow of processing. This figure shows a cascade of operations that transform the user’s input string, ultimately into a system response, report of failure or a request for the user to resolve ambiguity.
5.1. Input string to token sequence

The input string is the user’s raw text string that they type into C-Phrase. The first processing step transforms this raw string into an initial, non-tagged token sequence. The processing is straightforward. First all of the global substitutions (see section 4) are applied in the order of their definition. After this all illegal characters are removed. Finally the string is separated into tokens where whitespace is a separator and punctuation symbols (commas, colons, periods, currency symbols, etc.) are always tokens of length 1. Any matching double quotes delimit text that should be interpreted as a single quoted material token.

5.2. Token sequence to tagged token sequence

Given a token sequence, the first process is to apply a spell-check. The token sequence is scanned from left to right. A token with quoted material is accepted, as is the largest prefix that is within what we term the domain dictionary. For example if the current token sequence is ("population" "Des" "Moines"), then "Des" and "Moines" will be accepted without spell correction. However in the token sequence ("Des" "the" "mississippi" "flow" "through" "Michigan" "?"), the "Des" will be corrected to "Does". Later in processing “mississippi” will be corrected to “Mississippi”. These corrections are achieved by running a spell check on an unrecognized token, returning a ranked list of suggestions. The first suggestion within the domain dictionary is selected. In the case that no match is found, the token is simply dropped.

A record of the spell correction is packed in the structure that records the assigned tags so that the user may be presented with a “did you mean X?” phrase where X is the original sentence with the spell check applied, exactly as web search engines typically present spelling corrections.

After the tokens have been spell checked, the resulting token sequence is lexically analyzed, resulting in a set of possible tags being associated with tokens or spans of tokens. For example for the token sequence ("population" "of" "Ann" "Arbor" "," "please") the sub-sequence ("Ann" "Arbor") will be recognized as a legitimate string value via consulting the

---

5. The domain dictionary includes the entire set of alphanumeric strings in the naming relation $N$ as well as all the phrases appearing in the rules of section 3 which typically also includes the closed category words (e.g. prepositions, articles, wh-words, modal verbs, etc.) of the language. Matching is case insensitive.
domain dictionary. To enhance robustness all possible taggings will be considered. Thus if the user happened to type in the query “length of the Mississippi River”, then both ("Mississippi") and ("Mississippi" "River") would be recognized as possible string values (one referring to the `STATE.NAME` and `RIVER.NAME` respectively). Thus in general because of the possibility of sub-sequences spanning a set of tokens, the lexical analysis phase will have to apply over all $O(n^2)$ sub-sequences of a input sequence of length $n$. In addition, in seeking out valid string values, stemming is performed to find matches. The current stemming method is simply Porter’s method.

There are a plethora of independent processes that can recognize lexical types such as `STRING_VALUE`, `NUMERIC_VALUE`, `DATE_VALUES`, `URL_VALUE`, `QUOTED_MATERIAL`, etc. In principle each recognizer runs independent of the others, although certain optimizations are currently coded for. Before a fuzzy chart parser is engaged, rules are added to the stock of language and authored rules that reflect the tags obtained during lexical analysis. These rules are built in a standard way from tags.

5.3. Tagged token sequence to system response

A standard chart parser that performs semantic composition is applied to the tagged token sequence. Each rule has a plausibility measure between 0 and 1 and the overall plausibility of a parse is simply the product of these measures for all the rules employed. A tuple calculus expression is obtained for each parse in the chart whose plausibility
meets or exceeds a failure threshold (e.g. a plausibility above \(0.5\)). Given a set of such queries, equivalence checks are performed to filter out any redundancies.

Given a set of such resulting tuple calculus expressions, there are two other parameters that then determine the system’s response. The first is the immediate application threshold. All formulas that meet or exceed this threshold are able to be applied without a confirmation step being taken with the user. Finally there is the selection gap measure. In the case that there are a set of parse results that meet the failure threshold, then all the parses that are within the selection gap distance of the highest scoring parse will be added to the set of ambiguous parses that must be resolved by the user.

In the case that there is only one semantically distinct query that meets the immediate application threshold and for which there are no other competing parses within the selection gap, the system converts the query to an equivalent SQL query, evaluates this query over the database to obtain answers. Finally these answers and a paraphrase of the tuple calculus query is presented as a response to the user. In the case of ambiguity, or confirmation, queries are paraphrased via a paraphraser (not shown in diagram) and the user must select, or confirm, one interpretation. Once the user picks their intended interpretation, the system evaluates the query as above. When no parse is above the failure threshold we have the case of utter failure. In such a case the system will have to come up with a suitable message. Although the current C-Phrase system does nothing other than report requesting the user to rephrase their query, future work will investigate the best type of response in such situations and base responses on at least some island parse of the users input string.

6. Evaluation

6.1. Benchmark studies

Benchmark studies rely on a human built ‘gold standard’ of natural language sentence/logical query pairs, possibly obtained through wizard-of-Oz studies. The advantage of such benchmarks is that they set a bar on the expressivity and types of queries that a system must be able to handle in a given domain. For example systems incapable of handling negation or ambiguity (for example [20]), would be flagged as inadequate by such benchmarks.

A widely used benchmark for NLIs to databases is the GEOQUERY Corpus provided by Professor 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 883 queries. There are several other benchmarks available. For example there are the RESTAURANT and JOBS corpora also developed by Mooney’s group as well as the ATIS corpus. Finally the author has also built a very difficult corpus based on a very close reading of [6]. This corpus, called the COPESTAKE corpus sets very high semantic demands indeed.

When one conducts a benchmark evaluation for NLIs to databases one can use the familiar notions of precision and recall. Still their definition is somewhat complicated by the fact that for each query we have three possibilities: 1. the query is correctly answered; 2. the query is incorrectly answered; 3. the query is unanswered. As an aside, in the case of ambiguity, we count a query as correctly answered
if it is among the interpretations recognized by the system. Given this, \(# \text{ queries} = \# \text{ correctly answered} + \# \text{ incorrectly answered} + \# \text{ unanswered}\). The standard definitions of precision and recall follow.

\[
\text{precision} = \frac{\# \text{ correctly answered}}{\# \text{ correctly answered} + \# \text{ incorrectly answered}} \tag{2}
\]

\[
\text{recall} = \frac{\# \text{ correctly answered}}{\# \text{ queries}} \tag{3}
\]

We introduce an additional metric of what we call \textit{willingness}.

\[
\text{willingness} = \frac{\# \text{ correctly answered} + \# \text{ incorrectly answered}}{\# \text{ queries}} \tag{4}
\]

The intuition behind willingness is that it represents the tendency of a system to actually offer answers to queries. A system that always parses to some, possibly incorrect logical query, would have complete willingness. A system that very often gave the message, “sorry your request can not be processed” would in contrast have very low willingness. In classical information retrieval it is recall and precision that trade-off directly against one another. In contrast, for NLIs to databases it is willingness and precision which have this relationship. Also of note, \(\text{recall} = \text{precision} \times \text{willingness}\).

6.2. A small authoring study

Given that many systems have been evaluated using the GEOQUERY Corpus, we decided to see how well our authoring approach would work. Though this study is small, it does at least establish that our authoring system is usable for normal technical users and does not require specialized linguistic expertise. The subjects were several undergraduate computer science students that had recently taken an introductory relational database course. The subjects were instructed to read the user’s manual of our authoring tool and were trained on its use. Training times varied from practically none, to around an hour over a mocked up example student database in no way related to the GEOQUERY Corpus. Not surprisingly the two subjects that received the least training did the poorest.

After being trained on the system, subjects were presented with a random ‘training set’ of 100 of the GEOQUERY Corpus queries. Subjects were told that they needed to author the system to cover such queries. Among the remaining queries in the corpus, 33 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 Corpus 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 and willingness. Figure 7 shows the resulting precision and willingness measures through time.

6.3. Fielded systems studies through open-sourcing C-PHRA

Fielded systems studies put an actual system in a real world environment to see how users actually interact with it. The survey paper [19] presents qualitative results for NLI systems deployed in corporate environments up to the mid-1990’s. The web opens up many new possibilities for such studies. The advantage of web-based studies is that one may observe a large amount of real user behavior at relatively low cost. The interface may be run from a web-page and user sessions logged for later analysis. There is also the possibility that such fielded systems go beyond the category of ‘exotic toy’ and actually become dominating solutions for particular application niches. ‘Dominating’ means that a significant number of people prefer to use the NLI to access some pertinent information over all of the other common techniques (e.g., forms-based interfaces, hypertext navigation, key-word search, point-and-click query builders, formal query languages, etc.). If such a case is found, and there is strong demand, then it will give birth to a new interface paradigm in the domain and it will attract a lot of attention to NLIs to databases.

The disadvantage to fielded system studies is that it requires product level robustness and a keen sense of what is relevant to some segment of the browsing public. For C-PHRA, we have worked very hard to stabilize the system into near product level robustness. However the system still has a way to go. Additionally although we have informally conducted fielded system studies through present a geography interface for general querying by the public, what is needed is for teams to attach C-PHRA to their real databases to see how the system performs in the real world.

The need to stabilize C-PHRA along with the need to encourage others to conduct fielded system studies over their own databases, has thus led us to the decision to open-source the system. To encourage the widest possible use, we shall release it under the BSD type license, allowing unrestricted use and adaptation of the system. C-PHRA consists of approximately 12k lines of LISP code plus another 5k lines of code for an AJAX-based authoring interface to author the system over arbitrary relational databases. The
7. Related Work

Due to the public availability of GEOQUERY Corpus, we can compare our initial results to several machine learning approaches [24, 11, 7, 26, 27], an approach based on light annotation [20] and an authoring approach over Microsoft’s EnglishQuery product (described in [20]).

Figure 9: Comparison of our authoring results to machine learning.

Figure 9, adapted from [26], displays the precision and recall measures for all the systems machine learning based systems [24, 11, 7, 26, 27] over the GEOQUERY Corpus. We have added results for our authoring study (for the well trained authors) as well as the results reported in [20]. 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% recall with 60 queries in the test set (precision was slightly under 80%, thus in our terminology willingness was approximately 60%). In our experiments subjects average around 80% willingness after 120 minutes of authoring with an average precision of a little over 80%. 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 a recall of 86.59%, yielding, in our terminology an asymptotic willingness of 94%. Another machine learning experiment over a relaxed-CCG approach obtained similar results [27].

Our comparisons with machine learning approaches highlight a bootstrapping weakness that if overcome would probably make machine learning approaches dominant. The complete system with source code may be downloaded at http://code.google.com/p/c-phrase/.
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 [18]. 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 article 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. 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.

PRECISE [20] is a system based on light annotation of the database schema that was tested over the GEOQUERY Corpus corpus. PRECISE reduces semantic analysis to a graph matching problem after the schema elements have been named. Interestingly the system (as well as [23]) 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% willingness, yielding a recall 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 Corpus NLI using Microsoft’s EnglishQuery tool. The resulting system achieved rather poor results for such an expensive effort – approximately 80% precision and 55% willingness, yielding a recall 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 might also be interesting to investigate the methods the Wolfram Alpha team uses to cover new domains.

It should be noted that historically, transportable systems (e.g.,[8]) 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 [5]. 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[12] 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.

8. Conclusions

This article has presented C-Phrase, a state-of-the-art system for natural language interfaces to relational databases. Internally the system uses X-bar theory[9] inspired semantic grammars encoded in \( \lambda \)-SCFG to map user requests to an extended variant of Codd’s tuple calculus which in turn is automatically mapped to SQL. The NLI 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 [15]).

Using the GEOQUERY Corpus 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 [22]), especially under regimes that enable enhanced robustness [27]. 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.

Sapir’s proverb ‘all grammars leak’ rings especially true in the case of grammars for NLIs to databases. In any operational system a grammar that has been built up to cover a domain will be leaky in the sense that there will be turns of phrase that user’s will employ that the system is unable to parse. When a system is unable to understand a user’s request, and must respond with some variant of “Sorry, your request can not be understood”, the user, understandably becomes irritated. We see no solution other than for a human author to closely monitor the NLI continually patching the system to minimize the occurrences of such failures.

The central hypothesis of this work is that the robustness and usability of NLIs to databases can be enhanced to such a degree so as to overturn Shneiderman’s objection [21] that such systems are not “comprehensible, predictable and controllable”. The ultimate confirmation of this hypothesis would be the identification of a useful, though not necessarily broad class of interfaces where closed domain phrasal search of databases dominate all other alternatives (e.g., forms-based interfaces, hyper-text navigation, keyword search, point-and-click query builders, formal query languages, etc.).
9. Acknowledgements

I would like to acknowledge the hard work of Peter Olofsson and Alexander Näslund for building the AJAX authoring interface. I would like to also acknowledge Philipp Cimiano and Myra Spiliopoulou for a very productive discussion on precision and recall measures for NLIs to databases. Additionally thanks are due to Bart Massey for convincing me to open source C-Phrase on Google Code and for in fact helping me to do so (http://code.google.com/p/c-phrase/). Finally I would like to acknowledge Bonnie Webber for reminding me to reach back to standard chart parsing.

References

[16] 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.


