Towards a Theory of Natural Language Interfaces to Databases

Ana-Maria Popescu*
University of Washington
Computer Science
Seattle, WA 98195, USA
amp@cs.washington.edu

Oren Etzioni
University of Washington
Computer Science
Seattle, WA 98195, USA
etzioni@cs.washington.edu

Henry Kautz
University of Washington
Computer Science
Seattle, WA 98195, USA
kautz@cs.washington.edu

ABSTRACT

The need for Natural Language Interfaces to databases (NLIs) has become increasingly acute as more and more people access information through their web browsers, PDAs, and cell phones. Yet NLIs are only usable if they map natural language questions to SQL queries *correctly*. As Schneiderman and Norman have argued, people are unwilling to trade reliable and predictable user interfaces for intelligent but unreliable ones. In this paper, we introduce a theoretical framework for reliable NLIs, which is the foundation for the fully implemented PRECISE NLI. We prove that, for a broad class of *semantically tractable* natural language questions, PRECISE is guaranteed to map each question to the corresponding SQL query. We report on experiments testing PRECISE on several hundred questions drawn from user studies over three benchmark databases. We find that over 80% of the questions are semantically tractable questions, which PRECISE answers correctly. PRECISE automatically recognizes the 20% of questions that it cannot handle, and requests a paraphrase. Finally, we show that PRECISE compares favorably with Mooney's learning NLI and with Microsoft's English Query product.

Categories and Subject Descriptors

H.2.3 [**Database Management**]: Query Languages—*SQL*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces , Natural Language

General Terms

Languages, Algorithms, Reliability

Keywords

Natural language interface, database, reliability

*We thank Keith Golden, Dan Weld, Bonnie Weber and Tessa Lau for comments on previous drafts. This research was supported in part by ONR grant N00014-02-1-0324.

IUI '03 Miami, Florida USA

1. INTRODUCTION

Understanding arbitrary natural language sentences is widely regarded as "AI complete". Yet understanding questions such as "What are the Chinese restaurants in Seattle?" seems straightforward even for a machine. While natural language sentences have the potential to be subtle, complex, and rife with ambiguity, they can also be simple, straight forward, and clear. This paper attempts to formalize this intuition by identifying classes of questions that are "easy to understand" in a well defined sense.

Research on Natural Language Interfaces to databases (NLIs) has largely tapered off since the mid 1980's [2]. Yet the need for NLIs has become increasingly acute as more and more nontechnical people access a wide range of databases through their web browsers, PDAs, and cell phones (*e.g.*, accessing services such as `moviefone` and `tellme`). The tiny screen and keyboard of a cell phone or PDA make interaction paradigms such as direct manipulation and browsing far less appealing. Yet these form factors only increase the appeal of NLIs coupled with speech recognition.

In recent years significant advances have been made in dialog management (see work on spoken-language dialog systems [16, 1] and tutorial systems [7]), yet the core problem of reliably understanding a single sentence is far from solved. We anticipate that, due to its reliability PRECISE would be an attractive module for a dialog system.

1.1 Reliable NLIs

To satisfy users, NLIs can only misinterpret their questions very rarely if at all. Imagine a mouse that appropriately responds to a 'click' most of the time, but periodically whisks the user to an apparently random location. We posit that users would have an even worse reaction if they were told that a restaurant was open on a Sunday, but it turned out to be closed. If the NLI does not understand a user, it can indicate so and attempt to engage in a clarification dialog, but to actively misunderstand the user, form an inappropriate SQL query, and provide the user with an incorrect answer, would erode the user's trust and render the NLI unusable.

Consequently, this paper has twin goals. First, we introduce a theoretical framework for analyzing the reliability of an NLI — formally defining the properties of soundness and completeness, and identifying a class of *semantically tractable* natural language questions for which sound and complete NLIs can be built. Second, we show that the theory has practical import by measuring the prevalence of semantically tractable questions, and by measuring the performance of a sound and complete NLI in practice.

In essence, the paper introduces a new methodology for NLI research, which is loosely analogous to the taxonomic paradigms popular in Knowledge Representation (KR), computational learning theory (COLT) and elsewhere in Computer Science. COLT has a long history or classifying learning problems as "easy" or "hard" (*e.g.*, the concept class k-CNF is learnable in the PAC framework, whereas k-term DNF is not [21]). Similarly, KR researchers have identified a wide range of KR formalisms (*e.g.*, description logics) and analyzed the decidability and complexity of inference within them [3, 9]. By analogy, our taxonomic theory seeks to carve out classes of questions that are semantically tractable in the context of NLIs. We do not claim that NLI users will restrict their questions to a subset of English in practice, but rather that identifying classes of questions as semantically tractable (or not), and experimentally measuring the prevalence of such questions, is a worthwhile avenue for NLI research.

This paper goes beyond pure taxonomic results to describe the fully-implemented PRECISE system, which maps English questions to SQL queries[1]. The paper measures PRECISE's efficacy empirically on an independently gathered sample of several hundred questions posed by university students to three distinct databases in the domains of restaurants, jobs, and geography [19]. We report on two key experimental results. First, semantically tractable questions are in fact quite prevalent (from 77.5% to 97% of the questions in Mooney's data). Second, based on our theoretical analysis we have been able to build an NLI that is 100% accurate on all three of our benchmark databases. It answers semantically tractable questions and requests paraphrases for intractable ones.

1.2 Organization

The remainder of the paper is organized as follows. First, we introduce some basic notation and informally describe the class of "easy questions". Next, we give an example of PRECISE in action and describe each component of the PRECISE NLI. We then make our "easy questions" intuition more precise by specifying a natural subset of English that can be efficiently and accurately interpreted as non-recursive Datalog clauses. We also sketch the formal results on the soundness and completeness of our approach. We then present empirical results that show that PRECISE indeed has high coverage and accuracy over common English questions. We conclude with a discussion of related and future work.

2. DEFINITIONS

This section introduces our vocabulary for analyzing NLIs and semantically tractable questions.

A database is made up of three types of *elements*: *relations*, *attributes* and *values*. Each element is distinct and unique: an *attribute* element is a particular column in a particular *relation* and each *value* element is the value of a particular *attribute*. A value is *compatible* with its attribute and also with the relation containing this attribute. An attribute is *compatible* with its relation. Let a *wh-word* be any word in the set {"what", "which", "where", "who", "when"}. Each database attribute has a set of compatible *wh-values*.

[1]The PRECISE system can be accessed at *http://abstract.cs.washington.edu/~amp/index.html*

A *token* is a set of word stems that *matches* a database element. For instance, {`require,experience`} and {`need, experience`} match the database attribute **Required Experience**. Many different tokens might match the same database element, and conversely, a token might match several different elements (sometimes with different types). Each token has a set of possible types (*e.g. value token*, *attribute token*) corresponding to the types of the database elements it matches. A *syntactic marker* (such as "the") is a token that belongs to a fixed set of database-independent tokens that make no semantic contribution to the interpretation of a question.

2.1 Semantically Tractable Questions

Our definition of semantically tractable questions is based on the observation that many natural questions specify a set of attribute/value pairs as well as free-standing values, where the attribute is implicit. An attribute (or relation) may also be paired with a wh-word (such as "what"). For example consider the question, "What French restaurants are located downtown?" in the context of a database containing the relation **Restaurants** with attributes **Name**, **Cuisine** and **Location**. The word "French" refers to the value **French** of the *implicit* database attribute **Cuisine**, the words "located" and "downtown" refer to the *explicit* attribute **Location** and its value **Downtown**, and the word "restaurant" refers to the relation **Restaurants** and corresponds with the wh-word "what".

The fact that attributes may be implicit allows a form of "ellipsis" that occurs frequently in questions; recovering this missing information is an important challenge for semantic interpretation.

Our definition of a semantically tractable question captures the above intuitions. First, we require that some set of tokens exists such that every word in q appears in exactly one token. We refer to any such token set as a *complete tokenization* of q. PRECISE uses an *attachment* function to model syntactic attachment constraints derived from the question's parse tree. *Attachment* is a function from pairs of tokens to TRUE or FALSE. For instance, in the above example the tokens `located` and `downtown` are attached while the tokens `what` and `downtown` are not.

In order for the sentence to be interpreted in the context of the given database, at least one complete tokenization must *map* to some set of database elements E as follows:

1) each token *matches* a unique database element in E.

This means that there is a one-to-one match between the tokens in the tokenization and E.

2) each attribute token *corresponds* to a unique value token.

This means that (a) the database attribute matching the attribute token and the database value matching the value token are compatible and (b) the attribute token and the value token are attached.

For instance, consider the tokens `locate` (which matches the **Location** attribute) and `downtown` (which matches the value **Downtown**). **Downtown** is compatible with **Location** and the two given tokens are attached. Thus, `locate` *corresponds to* `downtown`.

3) each relation token *corresponds to* either an attribute token or a value token.

This means that (a) the database relation matching the relation token and the database element matching the attribute or value token are compatible and (b) the relation token is

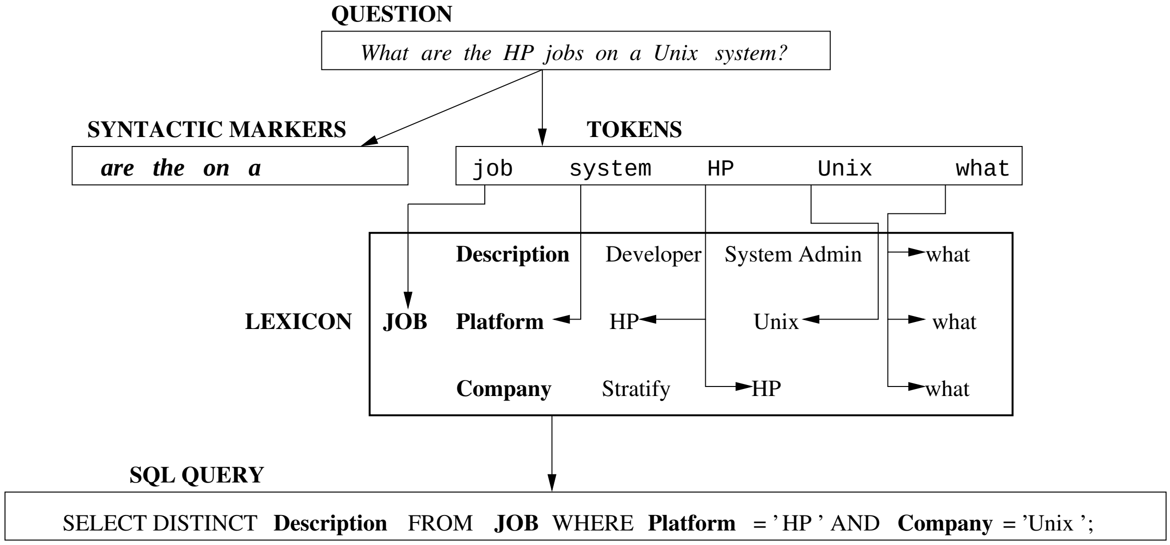

Figure 1: The transformation of the question *"What are the HP jobs on a Unix system?"* **to an SQL query, in the context of a database containing a single relation, JOB, with attributes Description, Platform and Company.**

.

attached to the corresponding attribute or value token.

For instance, the relation token `restaurant` (which matches the relation **Restaurants**) corresponds to the attached value token `French` (which matches the value **French** corresponding to the attribute **Cuisine**). Note that a value token need not have a corresponding attribute token in the sentence — this is the form of ellipsis mentioned earlier.

A mapping from a complete sentence tokenization to a set of database elements such that conditions 1 through 3 are satisfied is a *valid mapping* (see Section 5). If the sentence tokenization contains only distinct tokens and at least one of its value tokens matches a wh-value, we refer to the corresponding sentence as *semantically tractable.*

3. THE PRECISE SYSTEM

Given a question q, PRECISE determines whether it is semantically tractable and if so, it outputs the corresponding SQL query (queries). The problem of finding a mapping from a complete tokenization of q to a set of database elements such that the semantic constraints imposed by conditions 1 through 3 are satisfied is reduced to a graph matching problem. PRECISE uses the max-flow algorithm to efficiently solve this problem. Each max-flow solution corresponds to a possible semantic interpretation of the sentence. PRECISE collects max-flow solutions, discards the solutions that do not obey syntactic constraints, and retains the rest as the basis for generating SQL queries corresponding to the question q.

This section begins with an example of PRECISE in action. Subsequently, we describe each of PRECISE's modules in more detail.

3.1 PRECISE in Action

To illustrate PRECISE's behavior consider how it maps the example question "What are the HP jobs on a Unix system?" to an SQL query. This question is chosen to illustrate the sort of ambiguity that PRECISE is able to resolve automatically. For brevity and clarity, this example refers to a single relation (**Job**) with attributes **Description**, **Platform** and **Company**. We will then look at a slightly modified version of the sentence that shows how we handle multiple relations.

The tokenizer produces a single complete tokenization of this question: (`what, HP, job, Unix, system`). Note that the tokenizer strips syntactic markers such as "the" and "a". By looking up the tokens in the lexicon, PRECISE efficiently retrieves the set of matching database elements for every token.[2] In this case, `what`, `HP` and `Unix` are value tokens, `system` is an attribute token and `job` is a relation token (see Figure 1).

Next, the matcher constructs the attribute-value graph shown in figure 2. To understand the meaning of nodes in the graph, it is helpful to read it column by column from left to right. The leftmost node is a source node. The Value Tokens column consists of the tokens matching database values (which in turn can be found in the DB Values column). For instance, the token `HP` is ambiguous as it could either match a value of the **Company** attribute or a value of the **Platform** attribute. Edges are added from each value token to each matching database value. Solid edges represent the final flow path while dashed edges suggest alternative flow routes. Let F denote the flow in the network.

The matcher connects each database value to its corresponding database attribute. Each attribute is then connected to its matching attribute tokens and also to the node **I**, which stands for implicit attributes. All attribute tokens link to the node **E**, which stands for explicit attributes. Finally, both **E** and **I** link to the sink node **T**.

Notice the two instances of the column containing DB attribute nodes. The unit edge from each DB attribute node to itself ensures that only one unit of flow in fact traverses each such node. These edges are needed because more than one DB value is compatible with a given DB attribute and a DB attribute may match more than one attribute token - however, our definition of a valid mapping requires each DB attribute be used only once (see the Section 5).

[2]The lexicon contains synonym information (*e.g.*,''system" and "platform" are synonyms).

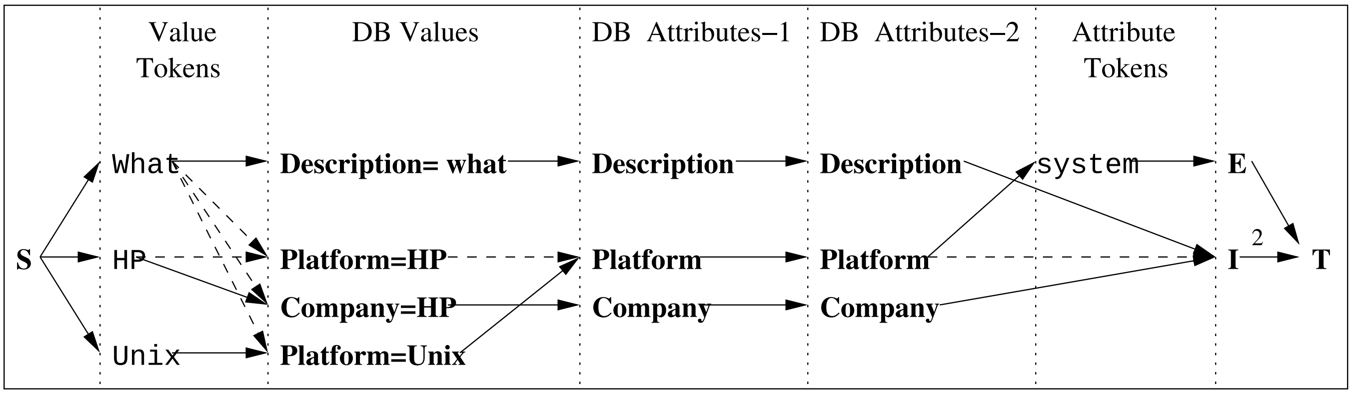

Figure 2: The attribute/value graph created by PRECISE **for the question** *"What are the HP jobs on a Unix system?"*

.

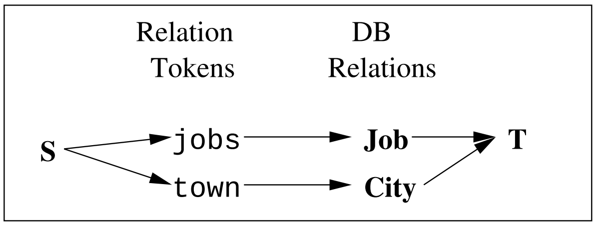

Figure 3: The relation graph built by PRECISE **for the question** *"What are the HP jobs on a Unix platform in a small town?"*

.

The graph is interpreted as a flow network where the capacity on each edge is 1, unless otherwise indicated. The capacity on the edge from **E** to **T** is the number of attribute tokens (in our example, 1). The capacity on the edge from **I** to **T** is the number of Value Tokens minus the number of Attribute Tokens. That difference is 2 in our example. Setting the capacity to be this difference forces the maxflow algorithm to send one unit of flow from some value token to each explicit DB attribute. The matcher runs the maxflow algorithm on the graph subject to these capacity constraints and searching for an integer solution. The maximum flow through the network in this example is 3. In fact, the maximum flow in any graph constructed by the PRECISE matcher is equal to the number of value tokens because each value token has to participate in the match produced by the algorithm.

The solid arrows indicate the path chosen by the maxflow algorithm. Note how the ambiguity regarding whether HP is the name of a company or of a platform is automatically resolved by maximizing the flow. The algorithm "decides" that HP is the company because this choice allows flow along two edges with capacity 1 into node **I**. Because the edge **(I,T)** has capacity 2, this choice maximizes the flow through the graph ($F = 3$). If the algorithm "decided" that HP was the platform, there would be no possible interpretation for "Unix" and the final flow would be 2.

After all attribute and value tokens have been matched to database elements, PRECISE ensures that all relation tokens correspond to either a value token or an attribute token. In the case of a unique relation token (`job`), this amounts to checking whether any of the matching database relations contains some attribute matching an attribute token. Since in our example `job` matches only **Job**, the algorithm has found a one-to-one match between the sentence tokens and the database elements that satisfies the semantic constraints in the set of conditions for semantically tractable sentences. At this point, PRECISE checks whether the match satisfies the syntactic constraints represented by pairs of attached tokens:(`what, job`),(`HP, job`), (`Unix, system`).[3] If all the attachment constraints are satisfied it means that a valid mapping has been found. Each valid mapping is converted into a SQL query; in the end PRECISE will return the set of non-equivalent such queries. In our example, a single valid mapping is found and so the PRECISE returns the SQL query at the bottom of Figure 1.

Consider a slightly different version of the above sentence : "What are the HP jobs on a Unix system in a small town?" in the context of a database containing a **Job** as well as **City** table. In addition to the **Description**, **Company** and **Platform** attributes, **Job** also contains a **JobID** field. **City** has attributes **Name**, **Size** and **JobID**, which is a foreign key corresponding to **Job.JobID**.

Using the lexicon, PRECISE determines that "town" is a synonym for "city". PRECISE first builds an attribute-value graph similar to the one in Figure 1. This time, the first node column contains a node for the value token `small`, which matches the value **small** of the database attribute **City.Size**. Since we now have 4 value tokens and 1 attribute token, the capacity of edge **(E, T)** remains 1 while the capacity of **(I, T)** is 3. A unique maximum-flow solution with $F = 4$ exists and suggests the following token-database element mapping: `HP` matches **Job.Company=HP**, `Unix` matches **Job.Platform=Unix**, `small` matches **City.Size=small** and `system` matches **Job.Platform**.

[3] Some types of flow-networks allow syntactic information to be used earlier in the generation of the valid mapping: the initial set of edges is pruned according to attachment information and the max-flow procedure runs on a modified version of the original flow-graph [10].

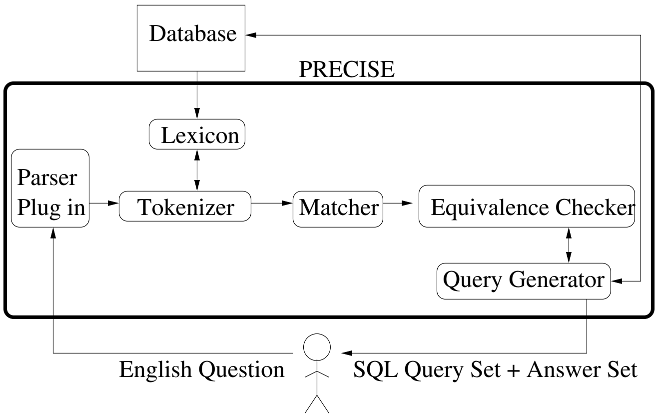

Figure 4: PRECISE **System Architecture**

Since all attribute and value tokens have been matched to database elements, PRECISE must now ensure that all relation tokens correspond to either a value token or an attribute token. We solve the problem by building an additional *relation* flow network as detailed below. PRECISE records the database relations employed by the attribute/value max-flow solution: **Job** and **City**. A relation flow network with unit-capacity edges is constructed as shown in Figure 3. PRECISE finds a max-flow solution in this network such that with $F = |\text{relation tokens}| = 2$. Such a solution yields a one-to-one match of relation tokens to database relations that contain attributes and values that match sentence attribute tokens and value tokens. PRECISE also ensures that the relation tokens are attached to these attribute and value tokens, which means that a valid mapping has been found.

No other valid mappings are found and the Query Generator constructs the SQL interpretation(s) corresponding to the given sentence. When the valid mapping contains multiple database relations, a different SQL interpretation is built for each join path corresponding to the set of relations. Our example assumes that the database contains only the tables **Job** and **City** and the only possible join path is **Job.JobID= City.JobID**.

The following SQL query is then built:

```
SELECT DISTINCT Job.Description
FROM Job, City
WHERE (Job.Company ='HP')
      AND (Job.Platform = 'Unix')
      AND (City.Size = 'small')
      AND (Job.JobID = City.JobID);
```

4. SYSTEM ARCHITECTURE

We now look at the components of the system in greater detail (see Figure 4 for an overview).

4.1 Lexicon

The lexicon supports the following two operations:

1) given a word stem ws, retrieve the set of tokens which contain ws.

2) given a token t, retrieve the set of database elements matching t.

In the following, we describe the manner in which the lexicon is derived from the database. The names of all database elements are extracted and split into individual words. Each word is then stemmed and a corresponding set of synonyms is identified using a general-purpose word ontology (Word Net). Each database element is thus associated with a set of word stems and each word stem is in turn associated with a set of synonyms. For every set s_d of stemmed words corresponding to a database element d, a *synonym-augmented* set of word stems syn_d is computed by successively replacing each word stem in s_d with a possible synonym. For instance, the set {`require, experience`} will yield the synonym-augmented set `need experience`. Any synonym-augmented word set syn_d represents a token matching the database element d. In the above example, {`require, experience`} and {`need, experience`} are tokens matching the database attribute **Required Experience**. All tokens derived in this manner point to hash tables containing the database elements they match. Furthermore, the tokens are placed in a hash table indexed by word stems - this two-level storage structure ensures that the lexicon can easily perform its two defining operations.

4.2 Tokenizer

The tokenizer's input is a natural language question and its output is the set of all possible complete tokenizations of the question. The tokenizer proceeds by stemming each word in the question, and the looking up in the lexicon the set of tokens containing the word stem. For each potential token, the tokenizer checks whether the other words in the token are also present in the question. For example, the word "price" is contained in tokens matching several database attributes (**Breakfast.Price, Lunch.Price, Dinner.Price**). However, when the question contains the phrase "price of breakfast" the only relevant attribute is **Breakfast.Price** and the only relevant token is `price breakfast`. Finally, the tokenizer also assigns to each token the *types* of database elements it could potentially match to (*e.g.*, value, attribute, etc.).

Once potential tokens are identified, computing the set of complete tokenizations is equivalent to the NP-hard problem of exact set covering. In practice, however, the average number of complete tokenizations is close to 1 and tokenization takes less than 2 seconds (wall-clock time).

4.3 Matcher

The matcher embodies the key innovation in PRECISE. We reduce the problem of finding a semantic interpretation of ambiguous natural language tokens as database elements to a graph matching problem. More precisely, according to [8], our reduction is to a maximum-bipartite-matching problem with the side constraints that all Value Token and Attribute Token nodes and a specified subset of the DB Value and DB Attribute nodes be involved in the match [10]. The matcher runs in polynomial time in the length of the natural language question and in the maximum ambiguity of question tokens.

4.4 Parser Plug in

PRECISE relies on the Charniak parser [4] to parse the question. PRECISE then extracts attachment relationships between tokens from the parse tree. For example, the parser enables PRECISE to realize that, in the question "What are the capitals of the US states?", the token `capital` is attached to the token `state`. The attachment relationships are used by the matcher in the generation of valid mappings (only semantic interpretations which satisfy the syntactic attachment constraints represent valid mappings).

4.5 Query Generator

The query generator takes the database elements selected by the matcher and weaves them into a well-formed SQL query. In the case of single-relation queries, this process is straightforward. The SELECT portion of the query contains the database elements paired with wh-words; the WHERE portion contains a conjunction of attributes and their values, and the FROM portion contains the relevant relation name for the attributes in WHERE. In the case of multi-relation queries, the generator adds *join conditions* to the WHERE clause, which reflect a join path that contains all the relations implicitly invoked by attributes in the query. The participating relation names are also listed in the FROM clause. If the join path is unique, the generator terminates. Otherwise, the generator generates a query for each possible join path and submits the queries to the equivalence checker (below). It is possible for multiple join paths to yield SQL queries that are not equivalent, leading PRECISE to flag a question as ambiguous. We omit many technical details here for brevity, but see [10] for formalization and detailed examples.

4.6 Equivalence Checker

The equivalence checker tests whether there are multiple distinct solutions to the maxflow problem and whether these solutions translate into distinct SQL queries. We show in [10] that finding distinct max-flow solutions amounts to finding all maximum-matchings in two bipartite subgraphs of the current flow-graph; we employ a well known maximum-matching enumeration algorithm [20] to do so. PRECISE checks for query equivalence using the algorithm in [5], which is polynomial time for acyclic conjunctive queries.

If PRECISE finds two distinct SQL queries, it does not output an answer, since it cannot be certain which query is the right one. Instead, PRECISE asks the user to choose between two or more competing semantic interpretations of particular tokens. For example, consider the question "What are the systems analyst jobs in Austin?". "Systems analyst" could refer to the job title **systems analyst**; on the other hand, "systems" could refer to the area **systems** and "analyst" could refer to the job title **analyst**. Thus, PRECISE asks the user to indicate the appropriate interpretation.

This completes our overview of PRECISE. While modules like the tokenizer and the equivalence checker are intractable in the worst case, in practice PRECISE is quite fast taking an average of 6 seconds per query (wall-clock time).

5. THEORY

In this section we formally define the class of semantically tractable questions. We show that for this class of questions, PRECISE is provably reliable (see the Theorem at the end of this section). We start by introducing the necessary notation and definitions.

Given a set of database elements E, let E_v, E_a and E_r denote the sets of values, attributes and relations in E. Given a set of tokens T and a set of database elements E such that each token in T refers to a unique matching element in E, let T_v, T_a and T_r denote the sets of value tokens, attribute tokens and relation tokens in T.

We can now define the notion of a **Valid Mapping:**

Given a question q with a tokenization T, attachment function AT, a lexicon L, and a set of database elements E, we say that there is a valid mapping from T to E if the following conditions hold:

1. *Sentence Token – Database Element Match*

   There is a one-to-one match (respecting L) between the tokens in T and the database elements in E.

2. *Attribute Token – Value Token Correspondence*

   Each attribute token *corresponds* to a unique attached value token. Formally, all T_a tokens can be assigned t o value/attribute token pairs (t_v , t_a) (no token appearing more than once) such that t_v , t_a are attached and for each (t_v , t_a) pair , a unique value/attribute pair (d_v , d_a) exists in E'_{av} such that t_v and t_a match d_v and respectively d_a as described in 1.

3. *Implicit Attributes*

   Each value token matches to a database value that is compatible with some database attribute. Some of these database attributes may not match any of the attribute tokens in the sentence; we refer to them as *implicit attributes.* Formally, E can be extended to set E' by adding attribute elements such that E' can be grouped into: - a set E_r consisting of distinct database relations. - a set E'_{av} of compatible attribute/value pairs, with no database element appearing more than once such that each relation in E_r contains at least one attribute in E'_{av}.

4. *Relation Token – Attribute/Value Token Correspondence*

   Each relation token *corresponds* to either an attached attribute token or an attached value token. Formally, each T_r token can be assigned to either an attribute token or a value token, creating the pair (t_r, t) with the following properties: a) the elements of the pair are attached, b) t_r matches a relation element d_r in E_r and t matches an element d of some E'_{av} pair such that d and to d_r are compatible.

Given a valid mapping for a question q, it is a straight forward syntactic manipulation to construct the *SQL interpretation* ϕ (see [10] for the laborious details).

Finally, we can introduce our key taxonomic distinction. A question q is said to be *semantically tractable* relative to a given lexicon L, and an attachment function AT if and only if q has at least one complete tokenization T such that:

1) All tokens in T are distinct.

2) T contains at least one wh-token.

3) There exists a valid mapping (respecting AT and L) from T to some set of database elements E.[4]

A semantic interpretation algorithm is said to be *sound* if it returns only valid mappings; the algorithm is said to be *complete* if it returns all valid mappings. The properties of soundness and completeness are essential to NLI reliability. Soundness blocks erroneous interpretations of the user's questions, and completeness ensures that when the user's question is ambiguous (i.e, has multiple distinct interpretations) the system can detect the ambiguity and respond appropriately.

[4]This definition of semantically tractable questions is simplified to facilitate the exposition of our theory. In fact, PRECISE handles a broader subset of English that includes sentences containing repeated tokens, database-specific functions, aggregate operators, negation, etc.

5.1 Formal Results

With the above definitions in place we can state our key guarantee regarding PRECISE.

THEOREM 1. *Given a lexicon L and an attachment function AT,* PRECISE *is sound and complete for any semantically tractable question.*

Furthermore, given any question, PRECISE can detect whether is semantically tractable. Thus, the following holds:

COROLLARY 1. *Given a question q,*
a) if q is semantically tractable with respect to L and AT, PRECISE will output the set of non-equivalent SQL interpretations of the question q.
b) Otherwise, PRECISE will reject the question as semantically intractable.

Naturally, the above theoretical results only apply when their assumptions (enumerated earlier) are satisfied. For instance, to prevent misunderstanding, PRECISE declines to answer questions containing words absent from its lexicon. However, that restriction is key to reliability — a single unknown word can easily change the meaning of a question.
Of course, our theory does not guarantee the reliability of a full-blown dialog system. However, we believe that interface designers will see great value in having a provably reliable NLI module, which would enable the interface designer to localize interpretation errors to other modules and recover appropriately.
How prevalent are semantically tractable questions in practice? If they are rare, then our results are of limited interest. In fact, our experimental results below provide evidence that semantically tractable questions are quite common.

6. EXPERIMENTAL RESULTS

We ran our experiments on three benchmark databases in the domains of restaurants, jobs, and geography. Each database was tested on a set of several hundred English questions whose corresponding SQL query was manually generated by an expert.[5]
Our first experimental question is: what is the prevalence of semantically tractable questions? We addressed this question by measuring the frequency of such questions in Mooney's independently collected set of questions. We detected whether a question was "semantically tractable" by running PRECISE on the question, if PRECISE generated one or more answers for the question then we knew that the question is semantically tractable; PRECISE automatically rejects intractable questions. The results are shown in Table 1. We see that semantically tractable questions are quite common ranging from 77.5% in the geography database to 97% in restaurant database. Examples of complex, but semantically tractable questions include "What is the largest city in the state with the smallest population?" and "What river does not traverse the state with the smallest population?"
The set of intractable questions for the Geography database includes questions containing unknown words, requiring

[5]The databases, English questions, and corresponding queries were generously supplied to us by Ray Mooney and his group. They were used in [19].

Restaurants	Geography	Jobs
97%	77.5%	88%

Table 1: The prevalence of semantically tractable questions in Mooney's data.

database functions not yet handled by PRECISE, or inquiring about information not present in the database. For instance, the question "What is the population density of the major cities in the US?" is not semantically tractable in the context of the Geography database because only information concerning the population density of US states is available. The question "What are some of the neighborhoods of Chicago?" cannot be handled by PRECISE because the word "neighborhood" is unknown.
The second experiment augments our theoretical soundness result for PRECISE. While PRECISE is provably guaranteed to be sound, that result holds under the assumptions spelled in Section 5 — what happens in practice? We found that on each of our benchmark databases, PRECISE made *no mistakes.* That is, the SQL queries it generated were the same as the ones generated by hand in Mooney's data. Thus, it is reasonable to believe that they captured the user's intent in asking their natural language question. This result is very important in meeting our goal of building a reliable NLI (see Figure 5).
To assess the difficulty of the problem, we compared PRECISE's accuracy with that of Mooney's learning NLI and with Microsoft's NLI for SQL server ("English Query"). The metric we used was *precision*, adapted from the information retrieval literature by [19]. Formally, the precision of an NLI on a data set is the number of English questions where the NLI correctly maps a question to the corresponding SQL query, divided by the number of questions that the NLI answers (each NLI declines to answer some questions). PRECISE shows better precision than the learning NLI and massively outperforms Microsoft's product on each of the databases tested.

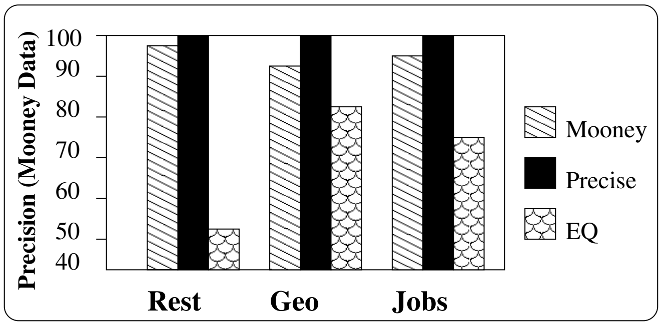

Figure 5: Precision on Mooney's Data: PRECISE makes no errors.' The precision differences between PRECISE and Mooney's NLI are statistically significant for the Geo and Job dbs (two-sample t-test, $p = 0.05$).

We were surprised by the extent to which both research prototypes outperformed Microsoft's English Query (EQ). EQ had trouble identifying the correct database elements referred to by the sentence. For instance, in the Restaurant domain some names of counties were mistaken for names of cities. In addition, both EQ and the learning NLI ignore parts of the question that they fail to parse, yielding incorrect answers. In contrast, PRECISE refuses to return an SQL

query in such cases.
A key advantage of PRECISE compared with a learning NLI such as Mooney's is that PRECISE is not sensitive to changes in the distribution of questions whereas inductive learning systems typically suffer when the distribution of the data over time changes from the distribution of training data. To measure the impact of this "distribution drift" on Mooney's system we collect a new data set of 70 questions for each domain from graduate students in the University of Washington, manually labeled them with the correct SQL queries, and input the questions to both the learning NLI and PRECISE. As the results in Figure 6 show, the learning NLI's precision dropped by 5.6% on average. In contrast, PRECISE remained 100% precise.

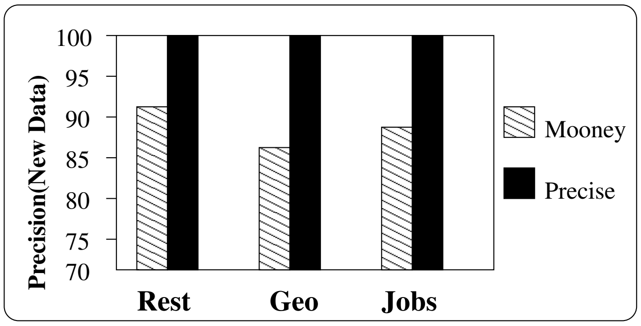

Figure 6: Precision (New Data): The precision of the learning NLI is sensitive to distribution changes whereas PRECISE remains error free. Each of the differences shown is statistically significant at $p = 0.05$.

All the NLIs tested in our experiments decline to answer some questions. We refer to the number of questions answered by an NLI, divided by the total number of questions as that NLI's *recall*. The recall for the different NLIs is shown in Figure 7. Although PRECISE's recall is lower than 80% on the geography databases, it is important to keep in mind that PRECISE is a highly portable system that does not require training examples (each manually labeled with the appropriate SQL query), nor does PRECISE require extensive manual customization as is the case with Microsoft English Query.[6] This is in sharp contrast to early NLIs, based on semantic grammars, which typically took months of programmer-time per database [12].

7. RELATED WORK

Our approach to language understanding differs from that of other natural language systems such as spoken language dialog systems [16], tutorial systems [7], etc. Speech systems have traditionally employed n-grams (sometimes in combination with CFG grammars) for language-modeling purposes. More recent work also addresses the use of large-scale dependency grammars which can include semantic and morphologic information, hierarchical statistical language models [16], etc. While powerful, these systems don't offer theoretical guarantees, and are based on very different algorithms. While there has been extensive work on NLIs [2], most of the earlier work is different from our own. PRECISE is *transportable* to arbitrary databases in the sense of [11], and in contrast with hand crafted semantic grammars,

[6] An undergraduate spent over 15 hours per database to customize the Microsoft product.

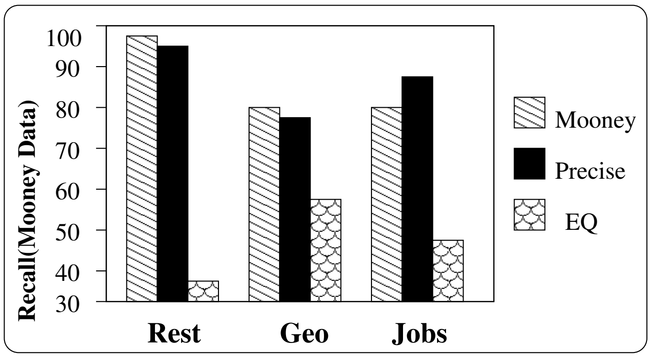

Figure 7: Recall on Mooney's Data: PRECISE handles slightly fewer questions than Mooney's learning NLI, but more questions than Microsoft's English Query. The recall differences between PRECISE and the learning NLI are only statistically significant on the Job dataset at $p = 0.05$.

which are tailored to an individual database (*e.g.*, LADDER [12]).
The early work with the most similarities to PRECISE was done in the field of sublanguages. Previous research on *sublanguages* has focused on subsets of natural language that are restricted to a single domain (e.g., medicine). Kittredge [14] realized that applications such as machine translation or text generation systems have a better chance of achieving high performance when tailored to specific domains.
In order to correctly analyze a given text, Kittredge relies on semantic constraints which cannot be stated for the entire English language but can be written down via a domain-specific grammar [13]. In addition, a sublanguage typically relies on a hand crafted domain-specific lexicon. Some sublanguages, such as the medical domain sublanguage, may use specialized word-formation rules. Traditional sublanguage work has looked at defining sublanguages for various domains. More recent work [15, 17] suggests using AI techniques to learn aspects of sublanguages automatically.
Our work can be viewed as a generalization of traditional sublanguage research because it is domain independent. We do not rely on a domain-specific grammar or lexicon. Instead, our lexicon is automatically extracted from the input database, and we make use of standard parsers. Instead of restricting ourselves to a particular domain, we restrict ourselves to sublanguages of English that can be defined more generally. For example, this paper shows how questions that can be mapped to non-recursive datalog clauses are semantically tractable. Finally, in addition to offering formal guarantees we report on experiments that quantify the prevalence of our "sublanguage" in the sample of questions collected independently by Mooney [19].
A more recent paper [18] uses *description logics* to translate between the logical form of a question and its SQL interpretation. The logical form of a question is obtained in a compositional manner from its syntactic information. Unlike PRECISE, the system uses extensive domain-specific knowledge and is not evaluated experimentally.
Our focus on NLIs was inspired by that of [19]. Mooney argued convincingly for renewed interest in NLIs and showed how strong performance can be achieved by a combination of novel learning methods. We have built directly on Mooney's experimental framework, but have chosen to explore a different point in the NLI design space. Namely, we focus on high

precision NLIs at the expense of recall. Thus, for example, we analyze every word in the sentence whereas Mooney's NLI will ignore unknown words and still try to interpret the sentence. Ours is not a learning approach, which obviates creating training examples and labeling each example with the appropriate SQL queries; finally, we insist on being able to plug in state-of-the-art parsers to leverage their ongoing improvements, which Mooney cannot easily accommodate. It is natural to consider synergies between the two approaches as they have complimentary strengths. For example, could PRECISE be a source of training examples for Mooney's learning systems?

After developing PRECISE independently, our literature search uncovered [6]. Chu and Meng's system stores information about a given database in a graph whose nodes represent database tables and whose edges represent different types of relationships between tables. Given a natural language question, the system uses a statistical approach in order to recognize the database elements (relations/attributes/values) referred to by the question. Next, the system relies on heuristics to supplement the information extracted from the question (*e.g.*, adding join conditions.) so that a valid SQL query can be formed. Like PRECISE, the system is transportable and graph based. However, the graph representation and the associated semantic interpretation algorithm are very different. Because Chu and Meng's system does not reduce semantic interpretation to a matching problem, is not able to offer any theoretical guarantees. Finally, the system was never evaluated empirically so it is difficult to assess its effectiveness in practice.

Our own group's work on the EXACT natural language interface [22] builds on PRECISE and on the theoretical framework laid out in this paper. EXACT composes an extended version of PRECISE with a sound and complete planner to develop a powerful and provably reliable interface to household appliances.

8. CONCLUSIONS AND FUTURE WORK

We have described a novel theoretical and practical approach to the problem of producing a reliable NLI to databases. Such interfaces are increasingly important on web sites, particularly as people access information from PDAs, cell phones, and other devices where small screens make GUIs less appealing. To the best of our knowledge, ours are the first formal guarantees on the soundness and completeness of an NLI. Moreover, we have shown experimentally that the guarantees result in a NLI that is highly reliable in practice. PRECISE provided correct answers to over 80% of the questions in our data sets, and correctly identified the rest as questions it does not understand.

Although PRECISE is only effective on semantically tractable questions, in future work we plan to explore increasingly broad classes of questions both experimentally and analytically. Finally, we plan to include PRECISE as a module in a full-fledged dialog system and investigate what theoretical guarantees can be made in this broader context.

An important direction for future work is helping users understand the types of questions PRECISE cannot handle via dialog, enabling them to build an accurate mental model of the system and its capabilities. PRECISE's responses to questions it decline to answer can be very helpful here. For example, PRECISE could be extended to say "please use simpler sentences" based on an analysis of its parse tree, and it could indicate the limitations of its knowledge with "I don't have geographical information outside the U.S.".

9. REFERENCES

[1] J. Allen, G. Ferguson, and A. Stent. An architecture for more realistic conversational systems. In *Intelligent User Interface*, 2001.

[2] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. Natural Language Interfaces to Databases - An Introduction. In *Natural Language Engineering, vol 1, part 1*, pages 29–81, 1995.

[3] F. Baader and B. Hollunder. A Terminological Knowledge Representation System with Complete Inference Algorithms. In *Proceedings of the First International Workshop on Processing Declarative Knowledge*, 1991.

[4] E. Charniak. A Maximum-Entropy-Inspired Parser. In *Proceedings of NAACL-2000*, 2000.

[5] C. Chekuri and A. Rajamaran. Conjunctive Query Containment Revisited. In *Proceedings of the Sixth International Conference on Database Theory*, 1998.

[6] W. Chu and F. Meng. Database Query Formation from Natural Language using Semantic Modeling and Statistical Keyword Meaning Disambiguation. Technical Report 990003, UCLA CS Dept., 16, 1999.

[7] M. Core, J. Moore, and C. Zinn. Initiative in Tutorial Dialogue. In *Proceedings of ITS 2002 Workshop on Empirical Methods for Tutorial Dialogue Systems (ITS-02)*, 2002.

[8] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts, 1984.

[9] F. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The Complexity of Concept Languages. In *Information and Computation, 134*, 1997.

[10] O. Etzioni, H. Kautz, and A. Popescu. Towards a theory of natural language interfaces to databases. In *Submitted for publication*, 2002.

[11] B. Grosz, D. Appelt, P. Martin, and F. Pereira. TEAM: An Experiment in the Design of Transportable Natural Language Interfaces. In *Artificial Intelligence 32*, pages 173–243, 1987.

[12] G. Hendrix, E. Sacerdoti, D. Sagalowicz, and J. Slocum. Developing a natural language interface to complex data. In *ACM transactions on Database Systems 3(2)*, pages 105–147, 1978.

[13] R. Kittredge. Sublanguages. In *American Journal of Computational Linguistics 8(2)*, pages 79–84, 1982.

[14] Kittredge.R. Variation and homogeneity of sublanguages. In R. Kittredge and J. Lehrberger, editors, *Sublanguage: Studies of Language in Restricted Semantic Domains*, pages 107–137. de Gruyter, Berlin, 1982.

[15] G. R. Adaptive information extraction and sublanguage analysis. In *Proceedings of IJCAI 2001*, 2001.

[16] S. Satingh, D. Litman, M. Kearns, and M. Walker. Optimizing Dialogue Management With Reinforcement Learning: Experiments with the NJFun System. In *Journal of Artificial Intelligence Research (JAIR)*, 2002.

[17] S. Satoshi. A New Direction For Sublanguage Nlp. In *New Methods in Language Processing*, pages 165–177, 1997.

[18] D. Stallard. A terminological transformation for natural language question-answering systems. In *Proceedings of ACL-86*, 1986.

[19] L. Tang and R. Mooney. Using Multiple Clause Constructors in Inductive Logic Programming for Semantic Parsing. In *Proceedings of the 12th European Conference on Machine Learning (ECML-2001), Freiburg, Germany*, pages 466–477, 2001.

[20] T. Uno. Algorithms for Enumerating All Perfect, Maximum and Maximal Matchings in Bipartite Graphs. In *International Symposium on Algorithms and Computation (ISAAC 1997)*, pages 92–101, 1997.

[21] L. Valiant. A Theory of the Learnable. In *Communications of the ACM, v27 n.11*, pages 1134–1142, 1984.

[22] A. Yates, O. Etzioni, and D. Weld. A Reliable Natural Language Interface to Household Appliances. In *Submitted to IUI-2003*, 2002.