gNarLI: A Practical Approach to Natural Language Interfaces to Databases

The Harvard community has made this article openly available. Please share how this access benefits you. Your story matters

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Citable link</td>
<td><a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:23526157">http://nrs.harvard.edu/urn-3:HUL.InstRepos:23526157</a></td>
</tr>
<tr>
<td>Terms of Use</td>
<td>This article was downloaded from Harvard University’s DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at <a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA">http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA</a></td>
</tr>
</tbody>
</table>
gNarLI:
A Practical Approach to Natural Language Interfaces to Databases
AJ Shankar, Wing Yung
TR-03-01
Introduction

Unlike most user-computer interfaces, a natural language interface permits any user (technical or non-technical) to communicate fluently with a complex system with very little preparation. A good NLI allows a user to enter commands and ask questions in English rather than having to negotiate with the underlying system's programming language, which often requires considerable study and intimate knowledge of system specifics. Clearly an NLI makes these systems easier to use and more accessible. This area of research is interdisciplinary; it draws mainly from the areas of linguistics, artificial intelligence, logic, and human-computer interfaces.

While it is not clear whether building a general multi-purpose NLI is even possible, there has been some success in designing natural languages interfaces to databases. Restricting the domain of knowledge to a particular database and limiting the nature of the interaction to queries and responses makes the problem less intimidating. Although the scope of an NLIDB is not as great, trying to design one that works is still a useful exercise. This is one of the few applications where a user-generated query (in English) is not entirely open-ended; the user is probably looking for some specific piece of information. Of course, an NLIDB still has to deal with many of the problems common to general NLIs, such as inherent ambiguities in English, and inferences that appear obvious to human users, but of which a computer has little or no knowledge. Nevertheless, a good NLIDB can, in theory, open up an access channel to huge amounts of data that are stored away.

Within the field of natural language processing there is also the danger of doing too much. Implementing a great number of complex features blurs the boundaries between what the system can and cannot handle. When a system can answer one complex query but not a similarly complex one, a user may become confused, which is one of the things that a natural language system is designed to prevent.

We attempt to solve some of these problems with gNarLI, a sophisticated rule-based pattern matcher that operates in several phases. This was not a typical research project, and this is not a typical research paper. Our approach to the problem is an extremely practical one; instead of trying to employ new algorithms and language processing techniques, we decided to create a system that answers as many real questions generated by real users as possible. The question space, even when the domain is limited to a single database, is extremely large. Rather than picking a very specific area of this domain to examine, we target the parts of the domain in which most simple queries reside.

Related Work

The first NLIDB was called LUNAR - the Lunar Sciences Natural Language Information System - which was used to serve queries regarding moon rocks[7]. It syntactically analyzed natural language queries, and then ran a semantic analysis on the resulting parse tree. Anther effort of the 1970s was RENDEZVOUS, which would prompt a user with a series of queries to figure out what he or she wanted to know[2].
It became apparent that a desirable feature of NLIDBs would be portability, so that the instances of same system could be used to query many different types of data residing in many different databases. Modularizing the components of a NLIDB did not seem like it would be a simple task - after all, constructing a NLI to a database full of lunar mineral data would involve a lexicon of scientific measurements and chemicals, whereas a NLI to a movie database would involve a completely different lexicon.

In 1982, CHAT80 was implemented in order to address this portability issue[6]. Unfortunately, it proved difficult to port. ASK tried to address similar issues by allowing users to add words to the lexicon, although this seems to be a very labor-intensive solution to the problem[5].

Another interesting aspect of ASK was that it attempted to serve as an NLI to other applications in addition to the database, like email. It seems that this was not an uncommon goal as JANUS was an NLI to databases, graphics devices, and other components, all of which could work together to try to respond to a query[4]. Perhaps they envisioned the perfect user-friendly system where all components run underneath an NLI: one that could retrieve some data, relate it to a correspondence (via email), check a schedule, and display everything in an easy-to-interpret fashion. Of course, expanding the NLI to realms outside of the database introduces its own problems because one of the most interesting properties of database queries is that they make up an extremely small subset of natural language. Trying to create an NLI that reaches beyond the data and relationships contained in a database complicates the problem.

The general approach in the 1980s seemed to be one of translating a parse tree into a database query[1]. Nowadays most systems tend to translate a natural language query into some sort of internal representation before translating this representation into a database query[1]. An internal representation independent of data and language can be ported to other databases and re-used.

Demer employs a technique involving unification grammars[3]. These grammars can take a set of compatible structures and return a new structure with all of the information contained in the original structures, and nothing else. Generally unification-based grammars are used to translate between natural languages, but Demer uses them to translate a natural language into SQL.

Commercial NLIDBs have also been released, although they aren’t wildly popular. For example, IBM and Symantec have their own products called LanguageAccess (1992) and Q&A. There are also a number of companies, such as EasyAsk, Iphrase, and Elf Software in the business of creating NLIDBs, as well as several companies that give NLIDBs virtual bodies and personalities.

Androutsopoulos notes several things that make this area particularly hard to evaluate[1]. He claims that it is difficult to tell how good an NLI is because there isn’t a set of benchmark queries. Also, although there has been a great deal of research in areas like parsing and dictionary usage/creation, there hasn’t been any consensus on the best way to approach a problem. Meanwhile, some of the theoretical problems go unsolved. He finally states that entering natural language queries with a keyboard is not practical, and that enthusiasm for the topic has died down since the 1980s.

We disagree with his statements for several reasons. First of all, the demand for a good NLIDB is higher than it ever has been thanks to the Internet and wireless technologies. Online shoppers looking for specific products may find an NLIDB to be the easiest way to search for items, and they’ll be more likely to spend money on something that they can find in a catalog. Sports fans could use an NLIDB to ask questions about their favorite teams and players without having to look through pages and pages of statistics. In addition, when a good NLI is paired with speech recognition technology and data that everyone needs
(such as stock quotes, weather, bus schedules, etc.), the wireless population will help it to succeed.

First we discuss the basic engine structure, and then we examine the different domains to which gNarLI has been applied.

**Basic Engine Structure**
Our design direction for gNarLI was primarily motivated by an analysis of the sample questions we received; the vast majority of the questions were relatively simple in terms of syntactic structure and data requested. Combining this information with our desire to make gNarLI easily portable to different domains (with a minimum of setup time for each domain), we decided to build gNarLI as a rule-based pattern matcher.

Our core system is written in Perl. Perl is excellent at pattern matching and string processing, and also allows easy manifestation of syntactic and semantic customization rules that aid the NLI. We use Perl’s DBI module to interface with a local MySQL server. All queries and database operations are thus transparent to the user. Execution time is negligible; parsing a question and generating a SQL query takes less than 0.01 seconds.

Each domain has a set of rules, each of which can be matched to a portion of the input question by a regular expression pattern and has some bearing on the SQL query to be generated. The engine attempts to apply the best combination of these rules to the question at hand and thus create a SQL query that will return the correct answer. The algorithm for determining which rules to apply is simple:

1. Find the rule that matches closest to the current point in the question string.
2. In case of a tie, select the rule that matches the largest portion of the string.
3. Apply the rule, advance in the string to the end of the rule, and go to 1.

There are algorithms that may be more effective, such as a dynamic programming approach that starts at the end of the input string and works backwards, minimizing the number of characters not covered by a rule. However, we felt the advancing algorithm we used is the best choice because it works well and is intuitive, and thus is a great aid to rule writers who must anticipate how the engine will parse a string to be able to write successful rules.

A benefit of this rule-based pattern matching approach is that the engine can be abstracted entirely away from the specifics of a domain. In fact, as a result the gNarLI engine is so compact that it is (currently) less than 400 lines of Perl code. It is also natural language independent: it can be as easily applied to Spanish as it is to English.

**Syntax File**
All domain-specific information is kept in a syntax file that is dynamically loaded by the engine at runtime. A gNarLI syntax file contains the following information:

**Metadata**
This section provides global information about the domain: the location and access parameters of the database, its description, the location of the preprocessor file, pronoun definitions, substitution SQL queries, and information on how to join tables. Each of these topics is discussed below.
**Rules**

Each rule is identified by a POSIX-compliant regular expression that the engine attempts to map onto the input string. Parts of the pattern can be tagged with parentheses for use in the rest of the rule; these parts are identified by $a, $b, etc. sequentially. One recommended rule convention is that proper nouns in the question are enclosed by quotes, since it is otherwise very difficult to determine what words constitute the noun. For instance, in the following question,

> tell me about being john malkovich.

it is very difficult for the computer to determine whether the user is asking about the movie "Being John Malkovich" or the actor John Malkovich unless quotes are used.

A rule can have any combination of the following tags:

- **Table**: Indicates that `table` should be SELECTed FROM in the SQL query.
- **Select**: Contains a list of pairs of tables and column names. The engine uses this list to determine what columns to SELECT, given the available tables.
- **Substitute**: Replaces a tagged portion of the pattern with a value from the database.
  - **Where**: Indicates a clause to be added to the WHERE portion of the query.
- **Post**: Indicates an arbitrary clause (usually a sort) to be appended to the query.
- **Pronoun**: Marks tagged portions of the pattern as corresponding to one or more pronouns, for use in subsequent questions.
- **Group-by**: A Boolean flag indicating that the query should be GROUPed
- **Group-attr**: The attribute by which to group the query
- **Onerror**: If there are any errors in executing the rule (for instance, a failed substitution), indicates another rule (that may of course have its own onerror tag) for the engine to execute in its place. gNarLI does not check for circular dependencies, so infinite recursion is possible with poorly designed rules.

Since none of these tags is required for any given rule, it is easy to develop a syntax file and get results immediately. For instance, gNarLI will work perfectly well without the grouping and pronoun functionalities defined in the syntax file; they can be implemented later as desired. Furthermore, because of the flat nature of the rules (other than onerror, there is no hierarchy to the rules; all are on equal footing) specific rules can be added or removed as desired without adverse affects on the other rules.

**Engine Operation**

The engine communicates with the user via a simple interface, and upon reception of a question, performs pronoun analysis, preprocessing, rules/substitution, join analysis, and SQL query construction before finally querying the SQL server and printing out the answer, or any error messages.

**User interface**

The gNarLI interface is currently terminal-based; a web interface can be implemented with little difficulty. Optional command line arguments are the syntax file for the desired domain and a list of questions to be automatically answered. The question interface is much like a UNIX shell, with question history and standard keystroke support. Answers are provided in tab-separated format, with one row per line. Warnings and errors are reported in the same fashion. This developmental version of gNarLI also supports several runtime flags that can be triggered by the user:

- **Echo**: Prints out each question as it is asked.
- **Debug**: Displays the preprocessed version of the question and the generated SQL query.
- **Trace**: Prints out every rule that successfully matches the question.
Pronoun: Prints out tagged identities that can later be referenced by pronouns.

Flags can be manipulated via the on, off, and toggle keywords:
> debug on
DEBUG set to ON

Of course, the benefit of a pattern matcher is that gNarLI accepts your demands liberally:
> you better turn off the durned trace or you’re dead
TRACE set to OFF

Appendix A (gNarLI Walkthrough) contains some sample flag output.

Pronoun analysis
The first step in answering a question is to identify and account for any pronouns. gNarLI has no hard-coded pronouns, since it is language independent. Synonymous pronouns (such as "he" and "she" in a gender-independent database) and plurality are defined in the metadata portion of the syntax file. gNarLI attempts to match a pronoun with a value tagged from the previous question or answer. For instance, if a user asks,
> when was "tom hanks" born?
the rule that parses "tom hanks" may tag it as corresponding to the "he" pronoun, so that if the user then asks,
> what movies did he act in?
gNarLI knows that "he" should be replaced with "tom hanks". Answers can be tagged in the same fashion, and tagged values are checked for the same plurality as the pronoun before they are deemed acceptable matches. If there is an ambiguity -for instance, if there are two people in the previous query to which "he" could possibly refer - gNarLI asks the user to select the item to which he is referring.

Preprocessor
Since most languages have many ways of saying the same thing, syntax files would have to grow arbitrarily large to handle the many variations on each rule that would be necessary to parse most queries. To avoid this problem, we preprocess each question in an effort to canonicalize common expressions, which drastically cuts down on the number of necessary rules. A preprocessor input file (whose location is specified in the syntax metadata) is simply a list of before/after regular expression pairs. Consider the following pair:
latest \d+ => $a most recent
that would convert the following question from
> What are the latest 5 action movies?
to
> What are the 5 most recent action movies?
So as long as there is a rule for \d+ most recent, both questions will be answered in the same way, by the same rule.

The preprocessor is especially handy for dates and times, which can come in a dizzying array of formats: 432pm, 19:20, 12/1/1979, 14 January 00, etc. A few preprocessor rules can be used to massage all of these into a standardized format.

Rules/Substitution
At this point, the question is ready for analysis, and gNarLI employs its rule-matching algorithm. Once the engine has decided that a particular rule provides the best match for the current part of the string, it analyzes the rule’s tags, storing most of them away for the joining and SQL query construction phases.
The substitute tag, however, is a special case. Using a SQL query defined in the syntax file, gNarLI queries the database and attempts to replace the marked value with one from the query answer (usually an id number). Substitute provides a good form of error checking. For instance, if the user asks,
> how many Oscars for best actor did "tom hakns" win?
and "tom hakns" is to be substituted for his id in the database, then gNarLI will fail at the substitution phase, since clearly "tom hakns" has no valid id. Thus, the user will know that his or her query failed because "tom hakns" is not a valid actor, instead of getting back an answer of 0 (as opposed to the correct answer of 2 for Tom Hanks) since obviously the fictional actor "tom hakns" never won any Oscars.

In addition substituting entities for their ids before constructing the query often simplifies table joins. Hanks’s id can be used to reference information about him in any other table, rather than having to join with the table in which his name is associated with his id.

Join Analysis
At this point, the question has been fully analyzed, and the main ingredients for a SQL query - what to select, from what tables to select, and under what constraints - are present. Now gNarLI must figure out what additional constraints to impose to enforce proper joining of the various tables required by the query. The syntax file metadata defines simple two-table joins, and gNarLI uses an intuitive joining system and combinations of these two-table joins to create the proper constraints for two or more tables. For simple relations, or a small number of tables, gNarLI consistently picks the correct joins every time. However, as the complexity of the relations or the number of required tables increases gNarLI becomes more error prone. A further analysis of gNarLI’s joining limitations and alternate joining strategies is presented later in this paper.

SQL Query Construction
gNarLI then assembles the SQL query by concatenating each of the subcomponents: SELECT, FROM, WHERE, POST. We regretfully point out that gNarLI currently does not understand "and", "or", and "not" - three words which are commonly used in queries and searches - because of the inherent ambiguities in parsing them. For instance, if a user asks,
> What employees work in Boston and Hartford?
he may mean what he says, but more likely he means, "What employees work in Boston or Hartford?"
Furthermore, users do not write with parentheses, so determining precedence of multiple Boolean operations provides ambiguous results at best. Since AND seems to be the most prevalent of the three Boolean operators, all WHERE clauses are ANDed together.

Additionally, if the group-by flag is set, gNarLI appends a GROUP BY clause to the query. After checking to see that the query is valid, it executes the query and prints out the answer. One special case is when a user asks a yes/no question. In this case it is the responsibility of the domain preprocessor file to prepend "Boolean" to the question (quite easily done). This triggers gNarLI to examine the result; if the first cell is nonzero, it responds "yes", otherwise "no". This approach allows each domain to specify what it considers yes/no questions.

Error Messages
Far more dangerous than not being able to answer a user’s question is giving an incorrect answer that the user thinks is correct. It is very difficult for NLIs to know when they are incorrect, and this problem has been a major deterrent to the success of past NLIs. gNarLI’s error-checking system, while not very robust, nevertheless attempts to catch errors early and specifically. For instance, it reports a specific error
message to the user if it failed at pronoun resolution, if it does not know what to select in the SQL query, or if it does not know from where to select. Furthermore, it warns the user that the answer may be incorrect if it has to skip a large amount of text in the question (whether as an absolute number of characters, or as a proportion of the length of the entire question), which is usually an indication that part of what the user asked is not handled by any rules in the syntax file. It also warns the user if it has to make an ambiguous substitution. For instance, the user may say, > what movie was "tom" in?
Clearly there are likely many such "tom"s in the database, and chances are the one the user is asking about will not be the one for whom the question is answered. Thus, gNarLI will report in the substitution phase if there is more than one replacement candidate.

**Data Domains**
We configured gNarLI to work with two different databases. Because it was difficult for us to find software against which to test (for cost reasons), we do not compare our systems to other natural language software systems. We evaluated gNarLI’s effectiveness based on its simplicity, portability, and ability to answer real user queries.

**Domain 1: Movies**

*Description*
This database contains movie data gathered from the Internet Movie Database [http://www.imdb.com/](http://www.imdb.com/). There are five tables in the database:

- **Person**: biographical information, such as date of birth, place of birth, and a unique ID for each person.
- **Movie**: movie information, such as release date, genre, length of movie, and a unique ID for each movie.
- **Oscar**: Oscar information, such as Oscar type, Oscar year, the movie associated with the Oscar, and the person who won the Oscar (if appropriate). Movies are identified by Movie.id and people by Person.id.
- **Actor**: information on which actors starred in movies. Only the "top" 5 actors for each movie are included. Each row contains a person ID and a movie ID, meaning that a particular person starred in a particular movie.
- **Director**: information on which directors directed movies. Each row contains a person ID and a movie ID, meaning that a particular person directed a particular movie.

As mentioned before, our queries are real questions from real users. We started from a list of 77 actual user questions and expanded this list with several interactive sessions with real-life users.

*Analysis & Results*
Movie trivia was the first domain that we considered. Starting from our question pool, we immediately eliminated 31 questions because they requested data that was not in the database. For instance, our database does not contain information on certain Oscar categories and roles played by actors and actresses.

Since we wrote rules as we were writing gNarLI’s core engine, the majority of the time developing this domain was spent finalizing the rule structure and the rules themselves. We found that the engine was flexible enough to accommodate new features such as pronouns and superlatives like "most" and "earliest" without any additional modifications. We estimate that it would take 10-15 hours to write gNarLI database rules from scratch.
Of the remaining 46 queries, gNarLI correctly answered all but 9 of them. Five of these were various flavors of "Which movies did X and Y star in?" which is a difficult query because it involves a Boolean operator. We hard coded one rule to deal with "Which movies star X and Y?" and could have handled the rest with some preprocessor rules, but a much better solution is to find some clever way to naturally join the Actor table to itself (as opposed to hardcoding the join in a specific rule) on the movie_id attribute, then select entries that contain both actors in them. Another solution would be to build Boolean operator support into gNarLI so that it process set-union and intersection via "and" and "or" itself. As described in the engine section, gNarLI does not handle Boolean operators in the question gracefully.

One of the other queries that we couldn’t answer contains an implied nested query. Trying to answer
> Which movies has Michael Douglas been in since "The Game"?
gNarLI is supposed to figure out that "since The Game" means the same thing as "since the year that The Game" was released." Then it can either do several joins or process "the year that The Game" was released" as a nested query. Unfortunately, MySQL does not support nested queries, so gNarLI would have to iteratively process the nested queries itself. We are looking into adding this functionality. However, considering that only one sample question of the hundreds we encountered requires nested query support, this does not appear to be a notable problem.

The final group of three questions sought movies that won multiple specific Oscars, like "Which movie won Oscars for best picture, best actor, and best actress?" The example query requires joining Oscar to itself twice on its movie_id attribute. gNarLI returns no answer in this case because it is not capable of joining a table to itself (which would require renaming several instances of the same table and propagating these new names through the SELECT and WHERE clauses); instead it tries to find a single Oscar awarded for best picture and best actor and best actress, which cannot exist because each Oscar can only have one type.

One interesting question that gNarLI can almost answer is "Who has been in the most Oscar winning movies?" gNarLI instead answers the question, "Who has been in movies which collectively have won the most Oscars?" gNarLI is not capable of making this distinction, so it returns a slightly wrong answer. It is still promising that it can do quite well with a query of this complexity because we did not encode any rules to deal with this sort of question directly.

Our test results therefore indicate that gNarLI correctly answered 37 of 46 of the initial sample questions. Five more could have been handled with the addition of some preprocessor rules. In addition, in later interactive tests with real users, gNarLI consistently answered more than 85% of questions correctly.

A sample session with the movies database is detailed in Appendix A.

Until very recently, our movie database was the only data domain that we had considered, raising questions about gNarLI’s portability to other domains. After a brief stint with basketball player and team statistics (discussed later), we decided to adapt gNarLI to a more relevant database.

**Domain 2: Classes at Harvard**

*Description*
An NLI to a database containing course, department, and professor information at Harvard struck us as something that is particularly useful for students looking for classes.
This database contains data on professors and classes in 10 departments at Harvard University. The data comes from the registrar’s web site [http://www.registrar.fas.harvard.edu/] and is contained in the following tables.

Classes: class information, such as meeting times, exam group, professor, description.

Professors: unique id, professor name, department id, tenure status, visiting, on leave

Exams: exam group id, fall date, and spring date.

Departments: department id and department names.

Because of time constraints, we generated sample questions in this area ourselves, focusing on questions most students ask when picking their classes for a term. For example, we have questions about course meeting times, exam dates, and professors. Due to the way the database is set up, it is also easy to answer questions about the departments and professors.

Analysis & Results
Porting gNarLI to this domain went smoothly. Generating rules to handle various phrases like "computer science department" and "meet on Monday and Wednesday" was straightforward. In addition, we didn't run into any problems getting Boolean queries and pronoun functionality to work. Implementing counting, grouping, and pronoun support also went well. No modifications to the gNarLI engine were necessary.

Via some clever preprocessing substitutions we were able to convert a time in nearly any form to hh:mm:ss form (in military time). When a.m. or p.m. is not specified, it correctly guesses that all numerical times between 1 and 8 refer to p.m. Thus "100", "1:00", "1pm", and "1:00 p.m." become "13:00:00." This demonstrates the flexibility of the preprocessor and highlights its ability to transform equivalent forms into a standard form before rules are applied.

Writing the appropriate rules and preprocessor substitutions, including full pronoun and grouping support, for gNarLI to answer most of our questions took approximately 8 hours between the two of us. We answered all of our basic orthogonal questions (approximately 30-40 of them), covering all of the columns in the database. We expect most real questions to be combinations of these queries. In probing the system, we have tested several hundred questions.

A sample session with the classes database is detailed in Appendix A.

Domain 3: National Basketball Association (NBA) Statistics: A Failure

Description
When we were first brainstorming ideas for a non-trivial data domain to port gNarLI to, we attempted to use a database of NBA statistics. We extracted data from Yahoo sports [http://sports.yahoo.com/] into the following tables:

Player_stats: player names, teams, and approximately 40 statistics, such as points, rebounds, assists, steals, and turnovers.

Team_stats: team locations, team names, and approximately 35 offensive and defensive statistics.

Highs: high and low, individual and team performances, such as fewest points scored by a team in a game and most steals by a player in a game.

Analysis & Results
After compiling a list of 40 colleague-generated queries and trying to write rules for the patterns, we found that gNarLI is not suited to handle huge statistical datasets. While the table structure is simpler than the other domains, the sheer number of columns caused problems. In addition, we encountered some
trouble because many of the sample queries required calculating averages and sums, comparing stats from different players and teams, and sorting in peculiar ways, all of which we were not prepared for after working with the movie database.

At first we attempted to add additional functionality to handle comparisons and superlatives in the most flexible way possible. After several fruitless efforts, it became clear that handling numbers introduced a different sort of complexity. Instead of encoding a rule for "more than X points per game", we had to add several new fields to the rule structure to handle "more than X" and "points per game" separately, otherwise gNarLI would require a separate "more than X" phrase for all 40 of the various statistics in the database. This is clearly an unacceptable solution. A future version of gNarLI will contain a clean way to handle numerical data.

**Future work**

There are several additional features that would make gNarLI even more useful than it is now. As mentioned before, adding support for Boolean operators, numerical datasets, and nested queries are high priorities.

Nested queries would be relatively easy to implement because the majority of them follow simple patterns. Words like "that" and "who" serve as good examples of nested query markers and can be used to break queries into multiple pieces.

We’d also like to implement a more robust join processor because more complex data may require some intricate table joining. One idea, suggested by Zhang, is the semantic graph[8]. The database tables are the nodes of the graph, and an edge between nodes represents a compatible attribute on which the tables can be joined, along with words and phrases that might hint at this join. This reduces the joining problem to a graph path-finding problem. Of course, even the semantic graph probably cannot handle all difficult joins, but it may be able to handle joining on the same table and joining tables with multiple compatible attributes.

**Conclusions**

gNarLI has shown itself to be very well suited to relatively small domains that do not require much numerical comparison or computation. Pattern matching leads to surprisingly good results for these domains: our engine consistently answers a significant majority of user questions correctly. It identifies situations in which it may be generating an incorrect answer, and warns the user. It provides an intuitive user interface, with pronoun support to simulate a real conversation. Furthermore, the engine itself is abstract enough that natural language support for new domains can be developed very quickly. New features for a given domain can be added incrementally without compromising existing functionality.

However, as the size and complexity of the domain grows, gNarLI suffers. Its intuitive table-joining method may fail, and the number of rules may grow beyond control. We are experimenting with better support for numerical data and more sophisticated join processing, but for the time being it is important to note that taking a practical approach to a specific subset of the natural language problem results in a solution that is practical only for that subset.

We still feel that within small domains, an NLI is incredibly useful. Even for experienced SQL programmers, it is a time-saver. Many times when we were investigating a potential aberration in the data, one of us would load up gNarLI, ask it a question, and get the answer before the other person was halfway done.
writing the SQL query to retrieve the offending data! For end users, its potential is much greater. Imagine a customer service representative inquiring about the status of an order ("When does Mrs. Smith’s current order ship, and has she bought any widgets before?"), a shopper searching product catalogs ("Which Nintendo video games released in 2000 do you have in stock?"), or a manager inquiring about his employees ("Which employees work in Boston and have good performance evaluations?"). Each of these domains is well within gNarLI’s capabilities, and an NLI is the perfect vehicle for retrieving such data. The user does not need to know any SQL or even anything about the database. All he needs to know is his native written language.

While some data does not require the sophistication of an NLI, others demand it, especially when ease of use, customer satisfaction and willingness-to-purchase are at stake.

References


Appendix A: gNarLI Walkthrough

User input is highlighted in bold and comments are in italics.

Movie Database

Welcome to gNarLI!
Current syntax file is movie.syn: Test Movie Database.

Please enter your question below, or "quit" to quit.
ECHO set to ON

> tell me about "tom hanks"
29  Tom Hanks  1956-07-09  Concord, California, USA

> when was he born?
1956-07-09

> what war movies came out after then and before 1965?
Lawrence of Arabia
Ciociara, La
Bridge on the River Kwai, The

> how many oscars did "casablanca" win?
2

> which oscars did it win?
BEST-PICTURE
BEST-DIRECTOR

> who won best director the most?
John Ford  4
William Wyler  3
Frank Capra  3
Steven Spielberg  2
Oliver Stone  2

> where was he born?
ERROR: Cannot figure out what you mean by 'he'!

> where were THEY born?
Cape Elizabeth, Maine, USA  John Ford
Mulhouse, France  William Wyler
Palermo, Sicily, Italy  Frank Capra
Cincinnati, Ohio, USA  Steven Spielberg
New York, New York, USA  Oliver Stone

> how many actors from "italy" won oscars?
7

> what action movies longer than 150 minutes won an oscar since 1990
Saving Private Ryan
Titanic
Braveheart

> how long are they?
170  Saving Private Ryan
194  Titanic
177  Braveheart
> what are the 4 longest dramas to win an oscar?
Dances with Wolves  224
Gone with the Wind  222
Ben-Hur  212
Giant  201

> who directed them?
Kevin Costner  Dances with Wolves
Victor Fleming  Gone with the Wind
William Wyler  Ben-Hur
George Stevens  Giant

> debug on
DEBUG set to ON

> what director from "california" directed the most sci-fi flicks?
PREPROCESS: who from "california" directed the most science fiction movies?
QUERY: SELECT DISTINCT Person.name, count(*) as cnt FROM Movie, Director, Person WHERE Person.pob like "%california%" AND Movie.genre like "%S%" AND Director.movie_id = Movie.id AND Director.director_id = Person.id GROUP BY Person.id order by cnt desc limit 5;
ANSWER:
George Lucas  2
Michael Bay  1

> what rated r movies with "love" in the title won best picture?
PREPROCESS: which rated r movies with "love" in the title won best picture?
QUERY: SELECT DISTINCT Movie.name FROM Oscar, Movie WHERE Movie.rating like "r" AND Movie.name like "%love%" AND Oscar.type = "BEST-PICTURE" AND Oscar.movie_id = Movie.id;
ANSWER:
Shakespeare in Love

> debug off
DEBUG set to OFF

> show me what dramas that have some ridiculously obscure piece of information and came out between 1970 and 1973
WARNING: Ignored a large amount of text: 's that have some ridiculously obscure piece of information and '
French Connection, The
Godfather, The
Klute

> what PG-rated musicals starring "liza minnelli" came out then
Cabaret

> pronoun on
PRONOUN set to ON

> tell me what actors from "new york, new york" were born after 1979, and the exact birthdate?
pronoun: "select" => 'they'
pronoun: "select" => 'he'
pronoun: "new york, new york" => 'there'
pronoun: in 1979 => 'then'
pronoun: in select => 'then'
WARNING: Ignored a large amount of text: ', and the exact '
Macaulay Culkin 1980-08-26

> pronoun off
PRONOUN set to OFF

> what movies came out then
By 'then' do you mean
1. in 1979
2. in 1980-08-26
? 1
Kramer vs. Kramer
Norma Rae
Thank you for using gNarLI!

Course Catalog Database

Welcome to gNarLI!

Current syntax file is classes.syn: Test Courses Catalog Database.

Please enter your question below, or "quit" to quit.
ECHO set to ON

> what "chemistry" classes involving "research" start after 11?
Introduction to Research-Junior Year
Experimental Physical Chemistry

> who teaches them?
James E. Davis Introduction to Research-Junior Year
Bretislav Friedrich  Experimental Physical Chemistry

> what "econ" profs teach the most classes
Caroline M. Hoxby      7
Alberto F. Alesina      6
Richard E. Caves        4
Christopher L. Foote    4
Jerry R. Green          4

> what "af-am" classes about "religion" meet in the fall?
Afro-Atlantic Religions

> what department has the most lecturers?
6  Anthropology
4  Afro-American Studies
4  Economics
3  Chemistry and Chemical Biology
3  Computer Science

> what graduate "physics" classes meet on friday?
General Theory of Relativity
Electrodynamics I
Advanced Quantum Mechanics I
Advanced Quantum Mechanics II
Statistical Physics
Group Theory with Application to Particle Physics
The Standard Model
Non-Relativistic Quantum Electrodynamics

> is "seltzer" teaching "operating systems"?
Yes
> which "phil" classes have the earliest exams
The Pre-Socratics: Proseminar  
Socrates: Proseminar  
Existentialism: Proseminar  
Philosophy of Science  
Philosophy and Literature.

> trace on
TRACE set to ON
> what graduate "physics" classes meet on friday and are taught by full profs?
which graduate "(["\"]*")" class
meet
DAY:(.)
full
professor
WARNING: Ignored a large amount of text: ` and are taught by`
General Theory of Relativity  
Electrodynamics I  
Advanced Quantum Mechanics I  
Advanced Quantum Mechanics II  
Statistical Physics  
Non-Relativistic Quantum Electrodynamics

> what departments have the least professors on leave
which department least
professor
on leave
1 Computer Science  
2 Applied Mathematics  
3 Afro-American Studies  
3 Engineering Sciences  
4 Chemistry and Chemical Biology

> trace off
TRACE set to OFF
> how many profs do those departments have
21 Computer Science  
10 Applied Mathematics  
15 Afro-American Studies  
31 Engineering Sciences  
28 Chemistry and Chemical Biology

> debug on
DEBUG set to ON
> what undergraduate "cs" classes meet at 10-11:30 in the fall with the final exam before January 23?
PREPROCESS: which undergraduate "computer science" class meet at 10:00:00 and end at 11:30:00 in the fall with the exam before DATE:20010123?
QUERY: SELECT class_name FROM exams, classes WHERE class_number between 100 and 200 AND classes.department_id = 106 AND start = "10:00:00" AND end = "11:30:00" AND (semester = 'B' OR semester = 'F' OR semester = 'Y') AND (fall_date < 20010123) AND (semester = "B" OR semester = "F" OR semester = "Y") AND classes.examgrp1 = exams.grp;
ANSWER:
Introduction to Formal Systems and Computation
> what profs have the most classes on thursdays?
PREPROCESS: who have the most class on DAY:Hs?
QUERY: SELECT DISTINCT prof_name, count(*) as cnt FROM classes, professors WHERE days like "%H%" AND (classes.prof1_id = professors.prof1_id OR classes.prof2_id = professors.prof1_id) GROUP BY professors.prof1_id order by cnt desc limit 5;
ANSWER:
Donald G. M. Anderson 6
Garrett B. Stanley 4
Gary Chamberlain 3
Dale W. Jorgenson 3
Michael S. Brandstein 3

> what and when are the latest "engineering" classes
PREPROCESS: which and when are the latest "engineering" class
QUERY: SELECT days, start, end, semester, class_name FROM classes WHERE start IS NOT NULL AND classes.department_id = 108 order by start desc limit 5;
ANSWER:
MW 16:30:00 18:00:00 S Introduction to Operations Research
H 16:00:00 18:00:00 B Engineering Design Projects
H 16:00:00 18:00:00 B Engineering Design Projects
MW 16:00:00 17:30:00 F Estimation and Control of Dynamic Systems
TH 16:00:00 17:30:00 F Special Topics in Biomedical Engineering: Orthoped

> debug off
DEBUG set to OFF

> how many visiting associate professors teach a "math" class
2

> tell me about "cs 50"
106 50 Introduction to Computer Science I 4949 10614 F MWF 10:00:00 11:00:00 3 Introduction to the intellectual enterprises of computer science. Algorithms: their design, specification, and analysis. Software development: problem decomposition, abstraction, data structures, implementation, debugging, testing. Architecture of computers: low-level data representation and instruction processing. Computer systems: programming languages, compilers, operating systems. Computers in the real world: networks, security and cryptography, artificial intelligence, social issues. Laboratory exercises include extensive programming in the C language and experimenting with and analyzing software systems. Note: No previous computer experience required. Computer Science 50

> what introductory "math" classes meet on monday and wednesday from 9-10?
Introduction to Linear Algebra and Multivariable C

> what department has the most assistant profs
15 Mathematics
14 Economics
6 Biological Sciences
5 Anthropology
5 Philosophy

Thank you for using gNarLI!

---

Appendix B: Sample Error Output and Failures

Welcome to gNarLI!
Current syntax file is movie.syn: Test Movie Database.

Please enter your question below, or "quit" to quit.
ECHO set to ON

> trace on
TRACE set to ON
> debug on
DEBUG set to ON

gNarLI (incorrectly) interprets the following question as "Who has starred in movies that have collectively won the most oscars?"

> who has starred in the most oscar winning movies?
PREPROCESS: who has was in the most oscar winning movies?
who
most
oscar
movies
WARNING: Ignored a large amount of text: ' has was in the '
QUERY: SELECT DISTINCT Person.name, count(*) as cnt FROM Movie, Oscar, Actor, Person WHERE Oscar.movie_id = Movie.id AND Actor.actor_id = Person.id and Actor.movie_id = Movie.id GROUP BY Person.id order by cnt desc limit 5;
ANSWER:
Jack Nicholson 10
Clark Gable 9
Dustin Hoffman 9
Diane Keaton 8
Robert Duvall 8

gNarLI ANDs these criteria together instead of attempting a same-table join

> which movies won oscars for best actor and best actress?
PREPROCESS: which movies won oscars for best actor and best actress?
which
movies
oscar
best actor
best actress
QUERY: SELECT DISTINCT Movie.name FROM Oscar, Movie WHERE Oscar.type = "BEST-ACTOR" AND Oscar.type = "BEST-ACTRESS" AND Oscar.movie_id = Movie.id;
ANSWER:

Forrest Gump
Apollo 13

18
... but because the domain did not include certain preprocessor rules, it cannot answer the following equivalent question:

> have "hanks" and "sinise" been in a movie together?
PREPROCESS: boolean has "hanks" and "sinise" was in a movie together?
boolean
"([^\"]*)" "([^\"]*)" was in
QUERY: SELECT count(*) FROM Actor, Person WHERE Person.id = 29 AND Actor.actor_id = 31 AND Actor.actor_id = Person.id;
ANSWER: No

*gNarLI reports an ambiguous substitution.*

> what movies was "michael" in?
PREPROCESS: which movies was "michael" in?
which
movies
(was|were) "([^\"]*)" in
WARNING: Ambiguous: more than one 'michael' in database; using Michael Anderson (I)
QUERY: SELECT DISTINCT Movie.name FROM Movie, Actor WHERE Actor.actor_id = 1214 AND Actor.movie_id = Movie.id;
ANSWER: 

*gNarLI complains that it does n't know what to do.*

> "tom hanks" is great
PREPROCESS: "tom hanks" "([^\"]*)"
QUERY: SELECT FROM Person WHERE Person.id = 29;
ERROR: Could not determine what information to retrieve!

Thank you for using gNarLI!