A Robust Portable Natural Language Data Base Interface

Jerrold M. Ginsparg

Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

This paper describes a NL data base interface which consists of two parts: a Natural Language Processor (NLP) and a data base application program (DBAP). The NLP is a general purpose language processor which builds a formal representation of the meaning of the English utterances it is given. The DBAP is an algorithm with builds a query in a augmented relational algebra from the output of the NLP. This approach yields an interface which is both extremely robust and portable.

1. Introduction

This paper describes an extremely robust and portable NL data base interface which consists of two parts: a Natural Language Processor (NLP) and a data base application program (DBAP). The NLP is a general purpose language processor which builds a formal representation of the meaning of the English utterances it is given. The DBAP is an algorithm with builds a query in an augmented relational algebra from the output of the NLP.

The system is portable, or data base independent, because all that is needed to set up a new data base interface are definitions for concepts the NLP doesn't have, plus what I will call the "data base connection", i.e., the connection between the relations in the data base and the NLP's concepts. Demonstrating the portability and the robustness gained from using a general purpose NLP are the main subjects of this paper. Discussion of the NLP will be limited to its interaction with the DBAP and the data base connection, which by design, is minimal. [Ginsparg 5] contains a description of the NLP parsing algorithm.

2. NLP overview

The formal language the NLP uses to represent meaning is a variant of semantic nets [Quillian 8]. For example, the utterances

"The color of the house is green."
"The house's color is green."
"Green is the color that the house is."

would all be transformed to:

g1 Isa: *colored
Tense: present
Colored: g2
Color: g3

g2 Isa: *house
Definite: the

g3 Isa: *color
Value: green

where "*colored", "*color" and "*house" are system primitives called concepts. Each concept is an extended case frame, [Fillmore 2]. The meaning of each concept to the system is implicit in its relation to the other system concepts and the way the system manipulates it.

Each concept has case preferences associated with its cases. For example, the case preference of *color* is *color and the case preference of *colored* is *physical-object.

The case preferences induce a network among the concepts. For example, *color is connected to *physical-object via the path: [*physical-object *colored* *colored *color* *color]. In addition, *color is connected to *writing-implement, a refinement of *physical-object, by a path whose meaning is that the writing implement writes with that color. This network is used by the NLP to determine the meaning of many modifications. For example, "red pencil" is either a pencil which is red or a pencil that writes red, depending on which path is chosen. In the absence of contextual information, the NLP chooses the shortest path.

In normal usage, case preferences are often broken. The meaning of the broken preference involves coercing the offending concept to another one via a path in the network. Examples are:

"Turn on the soup."
"Turn on the burner that has soup on it."

"My car will drink beer."
"The passengers in my car will drink beer."

3. The Data Base Connection

Consider the data base given by the following scheme:

Suppliers(sno,sname,scity)
Projects(jno,jname,jcity)
Parts(pno,pname,color,cost,weight)
Spj(sno,pno,jno,quantity,m,y)

Suppliers and projects have a number, name and city. Parts have a number, name, color, cost and weight. Supplier *sno* supplies a *quantity* of parts *pno* to project *jno* in month *m* of year *y*.

The data base connection has four parts:

1. Connecting each relation to the appropriate concept:

Suppliers -> *supplier
Projects -> *project
Parts -> *part
Spj -> *supply

2. Connecting each attribute to the appropriate concept:

```
sno,pno,jno  -> *indexing-number
sname,pname,jname -> *name
jcity,scity -> *city
m -> *month
y -> *year
cost -> *cost
weight -> *weight
quantity -> *quantity
```

3. Capturing the information implicit in each relation:

```
Parts(pno,pname,color,cost,weight)
   *indexnumberp
         indexnumber -> pno
         numbered -> Parts
   *named
         name -> pname
         named -> Parts
   *colored
         color -> color
         colored -> Parts
   *costs
         cost -> cost
         costobj -> Parts
   *weighs
         weight -> weight
         weightobj -> Parts

Projects(jno,jname,jcity)
   *indexnumberp
         indexnumber -> jno
         numbered -> Projects
   *named
         name -> jname
         named -> Projects
   *located
         location -> jcity
         located -> Projects

Suppliers(sno,sname,scity)
   *indexnumberp
         indexnumber -> sno
         numbered -> Suppliers
   *named
         name -> sname
         named -> Suppliers
   *located
         location -> scity
         located -> Suppliers

Spj(sno,pno,jno,quantity,m,y)
   *supply
         supplier ->   sno
         supplied ->   pno
         suppliee ->   jno
         (cardinality-of pno) -> quantity
         time -> m,y
   *spend
         spender -> jno
         spendfor -> pno
         amount (* cost quantity)
```

The *amount* case of *spend maps to a computation rather than a single attribute. If all the attributes in the computation are not present in the relation being defined, the query building program joins in the necessary extra relations. So the definition of *spend works equally well in the example scheme as well as in a scheme (eg., Spj(sno,pno,jno,cost,quantity)) in which the cost of a part depended on the supplier.

4. Creating pseudo relations

```
Pseudo Cities  jcity,scity
```

This creates a pseudo relation, Cities(cname), so that the query building algorithm can treat all attributes as if they belong to a relation. The query produced by the system will refer to the Cities relation. A postprocessor is used to remove references to pseudo relations from the final query. Pseudo relations are important because they ensure uniform handling of attributes. With the pseudo Cities relation, questions like "Who supplies every city?" and "List the cities." can be treated identically to "Who supplies every project?" and "List the suppliers."

The remainder of the data base connection is a set of switches which provide information on how to print out the relations, whether all proper nouns have been defined or are to be inferred, whether relations are multivalued, etc. The switch settings and the four components above constitute the entire data base connection. Nothing else is needed.

The network of concepts in the NLP should only be augmented for a particular data base; never changed. Yet different data base schemes will require different representations for the same word. For example, depending on the data base scheme, it could be correct to represent "box" as either,

```
g1   Isa: *part
     Conditions: *named(g1,box)

g2   Isa: *container
     Conditions: *named(g2,box)

g3   Isa: *box
```

The solution is to define each word to map to the lowest possible concept. When a concept is encountered that has a data base relation associated with it, there is no problem. If there is no relation associated with a concept, the NLP searchs for a concept that does correspond to a relation and is also a generalization of the concept in question. If one is found, it is used with an appropriate condition, usually *titled or *named. So "box" has a definition which maps to *box. In the data base connection given above, "box" would be instantiated as a "*part" since "*box" is a refinement of "*part" and no relation maps to "box."

4. Using the Connection

The information in the data base connection is primarily used in building the query (section 5.). But it is also used to augment the knowledge base of the NLP.

The data base connection is used to overwrite the NLP's case preferences. Since *located* -> Suppliers or Projects, the preference of *located* is specified to *suppliers or *projects. This enables the NLP to interpret the first noun group in "Do any suppliers that supply widgets located in london also supply screws?" as "suppliers in London that supply widgets" rather than "suppliers that supply London widgets". This is in contrast to [Gawron 3] which uses a separate "disambiguator" phase to eliminate parses that do not make sense in the conceptual scheme of the data base.

The additional preference information supplied by the data base connection is used to induce coercions (section 2.) that would not be made in the absence of the connection or under another data base scheme. "Who supplies London" does not break any real world preferences, but does break one of the preferences induced by this data base scheme, namely that *Suppliee* is a *project. London, a *city, is coerced to *project via the path [*project *located* *located *location* *city] and the question is understood to mean "Who supplies projects which are in London."

As mentioned in Section 2., the NLP determines the meanings of many modifications by searching for connections in a semantic net. The data base connection is used to augment and highlight the existing network of the NLP. If the user says, "What colors do parts come in?", the NLP can infer that the meaning of "come-in" intended by the user is *colored since the only path through the net between *color and *part derived from the case preferences induced by the data base connection is

[*part *colored* *colored *color* *color]

Similarly, when given the noun group "London suppliers" the meaning is determined by tracing the shortest path through the highlighted net,

[*supplier *located* *located *location* *city]

The longer path connecting *supplier and *city,

[*supplier *supplier* *supply *suppliee* *project *located* *location *location* *city]

which means "the suppliers that supply to london projects" is found when the NLP rejects the first meaning because of context. If the user says "What are the locations of the London suppliers" the system assumes the second meaning since the first (in the domain of this data base scheme) leads to a tautological reading. The NLP is able to infer that "The locations of the suppliers located in London" is tautological while "The locations of the suppliers located in England" is not, because the data base connection has specified *located to be a single valued concept with its *location* case typed to *city. If the system were asked for the locations of suppliers in England, and it knew England was a country, the question would be interpreted as "the cities of the suppliers that are located in cities located in England."

5. A trace of the query building algorithm.

The query building algorithm is illustrated by tracing its operation on the question, "Does blake supply any projects in london?"

The NLP's meaning representation for this question is shown below.

```
g0   Isa: *show          g5   Isa: *name           g9   Isa: *named
     Tense: present           Value: blake              Tense: present
     Toshow: g1          g6   Isa: *project             Named: g3
                              Element-of: g4            Name: g5
g1   Isa: *set                Conditions: g10     g10  Isa: *located
     Element: g2                                        Tense: present
     Subset-of: g4       g7   Isa: city                 Located: g6
                              Conditions: g11           Location: g7
g2   Isa: *project
     Conditions: g8      g8   Isa: *supply        g11  Isa: *named
                              Tense: Present            Tense: present
g3   Isa: *supplier           Suppler: g3               Named: g7
     Conditions: g9           Suppliee: g2              Name: g12

g4   Isa: *set                                    g12  Isa: *name
     Subsets: g1                                        Value: london
     Element: g6
```

The NLP treats most true-false questions with indefinites as requests for the data which would make the statement true. The question's meaning is "to show the subset of london projects that are supplied by Blake."

The query building algorithm builds up the query recursively. Given an instantiated concept with cases, it expands the contents of each case and links the results together with the relation corresponding to the concept. Given an instantiated concept with conditions, it expands each condition. For the example, we have.

1> Expand g1

2> Expand g2, the Element of g1

3> Expand g8, the Condition of g2.

4> Expand g3, the Supplier case of g8.

5 Expand g9, the Condition of g3. From the data base connection, a *named whose *named* case is a *supplier is realized by the Suppliers relation using the sname attribute. So we have,

4< g9 = *select from* Suppliers *where* sname = blake

3< From the data base connection, a *supply is realized by the Spj relation. This results in,
g8a = *project* jno *from join* Spj *to* g9

2< g8 = *join* g8a *to* Projects
g8 is the projects supplied by Blake.

2> Expand g4, the set g1 is a subset of, by expanding its element, g6

3> Expand g10, the Condition of g6

4 Expand g7, the location case of g10 yielding
g11 = *select from* Cities *where* cname = london

3< A *located with a *project in the *located* case is realized by the Projects relation using the jcity attribute. So we have,
g10a = *join* Projects *to* g11 *where* jcity = cname
g10b = *project* jno *from* g10a

2< g10 = *join* g10b *to* Projects
g10 is the projects in London.

1< Intersect the expansions of g2 and g4 and project the project names.
g13 = *project* jname *from intersection* g8 g10

The entire query is,

g9 = *select from* Suppliers *where* sname = blake
g8a = *project* jno *from join* Spj *to* g9
g8 = *join* g8a *to* Projects
g10 = *select from* Projects *where* jcity = london
g13 = *project* jname *from intersection* g8 g10

where the extra join resulting from the pseudo Cities relation has been removed by the post processor (section 3.)

Entirely as a side effect of the way the query is generated, the system can easily correct any false assumptions made by the user [Kaplan 7]. For example, if there were no projects in London, g10 would be empty and system would respond, generating from the instantiated concept g10 (i.e., the names used in query correspond to the names used in the knowledge representation), "There are no suppliers located in London." No additional "violated presupposition" mechanism is required.

The remainder of this section discusses several aspects of the query building process that the trace does not show.

Negations are handled by introducing a set difference when necessary. If the example query were "Does Blake supply any projects that aren't in London?", the expansion of g7 would have been,

Expand g7, the location case of g10 yielding
g11a = *select from* Cities *where* cname = london
g11 = *difference of* Cities *and* g11a

Conjunctions are handled by introducing an an intersection or union. If the example query were "Does Blake supply any projects in London or Paris?", the *location* case of g10 would have been the conjunction g13.

```
g13  isa *conjunction
     Type: or
     Conjoins: g7 g14

g14  Isa: *city
     Conditions: g15

g15  Isa: *named
     Named: g15
     Name: g16

g16  Isa: *name
     Value: paris
```

The result of expanding g13 would be,

g11 = *select from* Cities *where* cname = london
g15 = *select from* Cities *where* cname = paris
g13 = *Union of* g11 and g15

In general, "or" becomes a union and "and" becomes an intersection. However, if an "and" conjunction is in a single valued case (information obtained from the data base connection), a union is used instead. Thus "Who supplies london and paris?" is interpreted as "Who supplies both London and Paris?" and "Who is in London and Paris?" is interpreted as "Who is in London and who is in Paris?" in the example data base scheme.

Quantifiers are handled by a post processing phase. "Does blake supply every project in London?" is handled identically to "Does Blake supply a project in London?" except that the expansion of "projects in London" is marked so that the post processor will be called. The post processor adds on a set of commands which check that the set difference of London projects and London projects that Blake supplies is empty. The resulting query is,

g1 = *select from* Suppliers *where* sname = blake
g2 = *select from* Projects *where* jcity = london
g3 = *join* Spj *to* g1
g4 = *join* g3 *to* g2
g5 = *project* jno *from* g2
g6 = *project* jno *from* g4
g7 = *difference of* g5 *and* g6
g8 = *empty* g7

The first four commands are the query for "Does Blake supply a project in London?". The last four check that no project in London is not supplied by Blake.

A minor modification is needed to cover cases in which the query building algorithm is expanding an instantiated concept that references an instantiated concept that is being expanded in a higher recursive call. The following examples illustrate this. Consider the data base scheme below, taken from [Ullman 9].

Frequents(drinker,bar)
Serves(bar,beer)
Likes(drinker,beer)

If we ask, "Who frequents a bar that serves a beer John likes?", we get the following query.

g1 = *select from* Likes *where* drinker = john
g2 = *project* beer *from* g1
g3 = *join* Serves *to* g2
g4 = *project* bar *from* g3
g5 = *join* Frequents *to* g4

If we ask "Who frequents a bar that serves a beer that he likes?" the correct query is,

g1 = *select from* Likes
g2 = *project* beer,drinker *from* g1
g3 = *join* Serves *to* g2
g4 = *project* drinker,bar *from* g3
g5 = *join* Frequents *to* g4

In the first query "beer" was the only attribute projected from g1. In the second, the system projected both "beer" and "drinker", because in expanding "a beer he likes" it needed to expand an instantiated concept (the one representing "who") that was already being expanded.

All of these cases interact gracefully with one another. For example, there is no problem in handling "Who supplies every project that is not supplied by blake and bowles".

6. Advantages of this approach

The system can understand anything it has a concept about, regardless of whether the concept is attached to a relation in the data base scheme. In the Suppliers data base from Section 4., parts had costs and weights associated with them, but not sizes. If a user asks "How big are each of the parts?" and the interface has a *size primitive (which it does), the query building process will attempt to find the relation which *size maps to and on failing will report back to the user, "There is no information in the data base about the size of the parts." This gives the user some information about the what the data base contains. An answer like "I don't know what "big" means." would leave the user wondering whether size information was in the data base and obtainable if only the "right" word was used.

The system can interpret user statements that are not queries. If the user says "A big supplier is a supplier that supplies more than 3 projects" the NLP can use the definition in answering later queries. The definition is not made on a "string" basis e.g., substituting the words of one side of the definition for the other. Instead, whenever the query building algorithm encounters an instantiated concept that is a supplier with the condition *size(x, big) it builds a query substituting the condition from the definition that it can expand as a data base query. Thus the system can handle "big london suppliers" and answer "Which suppliers are big" which it couldn't if it were doing strict string substitution.

This facility can be used to bootstrap common definitions. In a commercial flights application, with data base scheme,
Flights(fl#,carrier,from,to,departure,arrival,stops,cost)
the word "nonstop" is defined to the system in English as, "A nonstop flight is a flight that does not make any stops." and then saved along with the rest of the system's definitions.

Coercions (section 2.) can be used solve problems that may require inferences in other systems. [Grosz 6] discusses the query "Is there a doctor within 200 miles of Philadelphia" in the context of a scheme in which doctors are on ships and ships have distances from cities, and asserts that a system which handles this query must be able to infer that if a doctor is on a ship, and the ship is with 200 miles of Philadelphia, then the doctor is within 200 miles of Philadelphia. Using coercions, the query would be understood as "is there a ship with a doctor on it that is within 200 miles of Philadelphia?", which solves the problem immediately.

Since the preference information is only used to choose among competing interpretations, broken preferences can still be understood and responded to. The preference for the *supplier* case is specified to "supplier but if the user says "How many parts does the sorter project supply?" the NLP will find the only interpretation and respond "projects do not supply parts, suppliers do."

Ambiguities inherent in attribute values are handled using the same methods which handles words with multiple definitions. For example, 1980 may be an organization number, a telephone extension, a number, or a year.

The NLP has a rudimentary (so far) expert system inference mechanism which can easily be used by the DBAP. One of the rules it uses is "If x is a precondition of y and z knows y is true then z knows x was and may still be true" One of the facts in the NLP knowledge base is that being married is a precondition of being divorced or widowed. If a user asks "Did Fred Smith used to be married?" in a data base with the relation Employees(name, marital-status) the system can answer correctly by using its inference mechanism. The exact method is as follows. The data base application receives the true-false question:

"Fred Smith was married and Fred Smith is no longer married"

Since the data base includes only current marital status information, the only way to answer the first part of the question is to infer it from some other information in the data base. The data base application sends the query to the NLP inference mechanism which would ordinarily attempt to answer it by matching it against its knowledge base or by finding a theorem which would gives it something else to match for. When called by the data base application, the inference mechanism simply uses its rules base to decide what it should match for, and then returns to the data base program. In this, example, the inference mechanism receives "Fred Smith was married" and using the precondition rule mentioned above, returns to the data base program, "Is Fred Smith divorced" or "Is Fred Smith widowed", which can be answered by the data base. The DBAP can call the inference mechanism recursively if necessary.

7. Implementation Status and Details

The DBAP is fully implemented and debugged. The NLP is implemented and still growing. Both are implemented in Franz Lisp, a dialect of LISP. Language processing and query generation are performed in virtually real time (average 1-3 cpu seconds) on a Vax 11-780.

The system is intended to be used with a Data Base Management system. The interface between the DBAP and the DBMS is a straightforward translator from relational algebra to the query language of the DBMS. I have written a translator for Polaris [Gielan 4].

The system handles all the examples in this paper as well as a wide range of others (Appendix A.). Several different data bases schemes have been connected to the system for demonstrations, including one "real data base" abstracted from the on-line listing of the Bell Laboratories Company Directory.

8. References

1. Cohen, P., Perrault C, and Allen J., *Beyond Question-Answering*, Report No. 4644, Bolt Beranek and Newman Inc., May 1981.

2. Fillmore C., *The Case for Case*, in **Universals in Linguistic Theory**, Eds., Bach E, and Harms, R., Holt, Rineheart and Winston, New York, 1968.

3. Gawron M. G., et. al., *Processing English with a Generalized Phrase Structure Grammar*, 20th Annual Meeting of the Association for Computational Linguistics, June 1982.

4. Gielan D., **Polaris User Manual**, New York Telephone Company, January 1981.

5. Ginsparg,J., *Natural Language Processing in an Automatic Programming Domain*, Memo 316, Stanford Artificial Intelligence Laboratory, Stanford University, 1978.

6. Grosz, B., *Transportable Natural-Language Interfaces: Problems and Techniques*, 20th Annual Meeting of the Association for Computational Linguistics, June 1982.

7. Kaplan, S. J., *Cooperative Responses from a Portable Natural Language Data Base Query System*, Ph.D. dissertation, Department of Computer and Information Sciences, University of Pennsylvania, Phila., Pa., 1979.

8. Quillian, M. R., *Semantic Memory*, in Minsky, M., Ed., **Semantic Information Processing**, The M.I.T. Press, 1968.

9. Ullman, J., **Principles of Database Systems**, Computer Science Press, Potomac, Maryland, 1980.

Appendix A: Sample Questions and Answers

* does every department head in 1135 have an office in holmdel?

No. all heads that work for center number 1135 are not located in an office in ho

* who isn't

alles,h g is the head not located in an office in ho that works for center number 1135

* who has been with us for longer than fred smith?

There are no employees that have worked for you for more than the length of the time fred smith has worked for you

* list the employees who used to be married and are not married now.

paul rochester, verlon biggs and willie davis are the employees that were married and that are not married now.

* production between february 1979 and april 1981

```
11352 - 768 - widget - march 1981
11352 - 888 - widget - april 1981
11352 - 133 - screw - jan 1980
11354 - 133 - widget - jan 1980
```

* who flies between london and paris

twa and pan am are the airlines that fly from london into paris

* get me a flight to paris from london before 3pm

```
a flight before 1500 into paris from london:
fl# airline departure arrival plane stops price
2   pan am  1400      1800    dc10  0     180
```

* the latest flight out of london

```
the latest flight from london:
fl# airline to    departure arrival plane stops price
1   twa     paris 1500      1700    dc10  3     280 *
```

* who supplies more than two projects

bowles and blake are the suppliers that supply to more than 2 projects

* who supplies more than the boss project

the suppliers that supply to the projects besides the boss project are:
decker, blake, bowles and white

* who supplies more projects than Decker

bowles and blake are the suppliers that supply to more projects than the number of the projects that are supplied to by decker

* list the suppliers who supply projects in cities they are not in

the suppliers that supply to the projects located in the cities in which they are not located in are:
decker, blake, bowles and white

* how many parts does white supply

There are 9 parts supplied by white

* how many different parts does white supply

There are 2 kinds of parts supplied by white

* what are they

box and screw are the kinds of parts supplied by white

* who supplies the most kinds of parts

bowles is the supplier that supplies the most kinds of parts

* how many does it supply

There are 4 kinds of parts supplied by the suppliers that supply the most kinds of parts

* how much does the mixer project spend on widgets from blake

440 is how much was spent by the mixer project on widgets supplied by blake to it

* how much did the mixer project spend on each part

```
widget - 649
box - 3
```

* print the drinkers that frequent only bars that serve some beer that they like

george and john are the drinkers that frequent only bars that serve some beers liked by them