Robust, cooperative and transportable natural language front ends to databases

Karen Sparck Jones
Computer Laboratory, University of Cambridge
New Museums Site, Pembroke Street, Cambridge CB2 3QG, UK

This paper was also published in *Information Technology in Indian Languages* (Ed. S.V. Raghavan and S. Venkatasubramanian), Delhi: Macmillan India, 1988, 103-119; reprinted in *Computer Science and Informatics* 18(2), 1989, 21-28.
The present text is authoritative.

Abstract

The paper discusses issues in designing natural language interfaces to database systems that are robust, cooperative, and transportable, and indicates the current state of the art in providing these.

Introduction

Natural language database inquiry has a long history in research on automatic natural language processing (Green et al 1963), and commercial front ends are now available (Harris 1984, Hendrix 1986). Those built in the major projects of the seventies, like LUNAR (Woods et al 1972), LADDER (Hendrix et al 1978) and TQA (Damerau 1980), showed that linguistically motivated and practically efficient front ends could be constructed; and LADDER in particular demonstrated that front ends could be applied to existing DBMSs, i.e. that the direct control of the data retrieval operations characteristic of LUNAR is not needed for an effective front end.

Research and development since the seventies has concentrated on making front ends more robust and more cooperative in any individual application, and more transportable across applications. These two lines of work are in fact related because improving performance requires more information which, if it is application-specific, has to be replaced for a new application. This paper is intended to illustrate the issues involved in robustness and cooperativeness, and in transportability, and indicate the current state of the art in promoting them. The discussion suggests, somewhat surprisingly, that what is required for more robust and cooperative front ends is also required for more transportable ones.

For the purposes of discussion, the general structure and functions of a database system with a natural language front end are assumed to be those shown in Figure 1, where questions are specific, object-level ones of the conventional sort, as illustrated. The general challenge of building a natural language front end is then being able to handle not only input questions like the example shown, "What suppliers ship P3s?", which maps fairly straightforwardly onto the appropriate formal query, but also ones which are more remote from the query language and consequently require more complex transformation, like "Who can I get parts called P3s

from?". In general, as discussed in more detail below, the linguistic interpreter operates in several stages, lexical pre-processing, syntactic parsing, semantic analysis, and conversion into the query language.

Robust and cooperative front ends

Detecting errors

Broadly speaking, making front ends more robust means making them more able to cope with 'ill-formed' or 'irregular', i.e. *defective* inputs; and making them more cooperative means ensuring that they can respond informatively when defective inputs are encountered. From a practical and commercial point of view it is important that front ends should be able to handle misspellings, faulty syntax, and other such 'vulgar' defects. But a front end should also be able to handle other kinds of defective input, like questions which embody some misunderstanding of the database, and these may be much more challenging. It may in any case be difficult in individual cases to decide whether a question is defective for trivial or serious reasons, for example a word in the wrong syntactic category may represent either a misspelling or a misconception.

Formal database queries may of course be defective too, not only superficially in, say, format, but in some more significant way; but defective inquiries are a much more serious problem with natural language questions both because natural language itself is inherently ambiguous, and because the user is (and indeed is intended to be) more detached from the details of the database. At the same time recognising defective inputs is more difficult because it may not be obvious, at any point in the possibly extensive transformation the question has to undergo, whether a question is intrinsically defective or is merely underdetermined for the current stage of processing and may be sorted out later.

Natural language questions addressed to a database may be defective in various ways, as illustrated in Figure 2 for example inputs to the notional Suppliers and Parts database used by Date 1977. The types of defect are rather simplistically characterised, and the distinctions between them are not absolute. Any particular question may, moreover, be defective in a way which is ambiguous between types: for example "Jons" could be a misspelling for "Jones", but could equally represent a user misconception about the set of existing suppliers, i.e. be a type B2 error. (Note that "domain" is used here to refer in an informal sense to refer to the world of the database, and more specifically to the sort of world which is instantiated by the set of individuals constituting a particular state of the database. Thus a description of the domain normally characterises a closed or open set of possible individuals, but may in practice include constant particular individuals.)

In general, defective questions are signalled by breakdowns in normal question processing: an input word cannot be found in the lexicon so syntactic parsing cannot be initiated; parsing can be initiated but not completed; it may be impossible to carry semantic analysis through; or the semantic representation of the question cannot be fully translated into the data language. But it may also be necessary to take specific steps to check for defective inputs because, at least in some cases, it may be possible to derive formal queries for questions and to search the database with these queries, to obtain outputs which are not recognised as faulty: zero outputs are an obvious case (Kaplan 1982). Defective questions, that is, can lead to misleading responses.

But detecting defective inputs, in the sense of determining the precise nature of the defect

rather than merely noticing that something is amiss, may not be a simple matter, as the discussion in AJCL 1983 shows. Thus if the syntactic parser fails because the next word will not match any of the acceptable syntactic possibilities, this could imply any of a variety of underlying real errors: the word itself is wrong, some other earlier word generating the blocked path is in fact the wrong one, some word has been omitted, and so forth. Again, if there is an intensional failure it may be difficult to locate the error. For example, with "What city do the 300 quantity shipments come from?", it is not clear whether the question is about where parts come from, where suppliers of parts are, or where shipments come from, though only the last is strictly an error.

Detecting defective inputs thus requires not only additional processes like those used by Kaplan for checking question presuppositions or 'presumptions' after a null search, but also that information about the database which may be only implicit should be made explicit. This may involve both a characterisation of semantic constraints which are often simply assumed to be known by the database user, and a metalevel description of the database. Both of these may be offered by a DBMS designed for access through a conventional query language, so questions violating semantic constraints may be trapped, and metalevel questions may be treated as such and not rejected as improper object-level queries. But these DBMS resources may not be fully adequate, given the pressures put on them by variety and indirectness of natural language questions. A conventional data model representation of Suppliers and Parts would be unlikely to provide enough information to pinpoint the nature of the error in "Are Jones's P3s heavy?", for example.

It may of course be the case that a question can be given a perfectly proper interpretation, which retrieves a plausible- looking answer, but the interpretation is not in fact the one the user intended. This can occur because natural language questions can be ambiguous. But this problem is not the issue here, and it is in any case one which has an adequate conventional solution, namely to display either the derived formal query or a natural language paraphrase of this to the user for checking. The point here is ensuring that the system behaves cooperatively when an input defect has been detected. Detection itself presents problems, but generating cooperative responses presents further problems, and in particular may require more powerful processors and richer information resources than those needed to detect the errors themselves.

0.1 Responding to errors

It is useful here to distinguish cooperative responses from friendly or helpful ones. Friendly behaviour includes giving 'not dead' signals; helpful behaviour includes providing a count of the number of items retrieved from the database if this is large so the user can decide whether he wants them all listed, or displaying data in convenient graphic forms like histograms (Harris 1984). Cooperative behaviour is more narrowly focussed on providing linguistically and informationally appropriate responses, in particular to what the system sees as defective inputs. Helpful output may take a linguistic form, e.g. "There are 3000 matching items: do you want to see them all?", and cooperative responses may be supplied for questions which are not defective and for which the DBMS has delivered a correct answer: thus Joshi et al 1984 suggest that it may be necessary to design or amplify an inquiry system's output to ensure that the user does not misunderstand the system's answer (for example because of the beliefs he has previously manifested in the dialogue). However cooperative responses are much more important for defective input because they should show or explain to the user what is wrong with his question. Cooperative response thus depends both on correctly identifying defective

input and on being able to characterise it constructively from the user's point of view, so the user can take whatever steps are appropriate, which can range from reformulating the question to abandoning it as misdirected to the given database.

But cooperative behaviour is not necessarily friendly in the straightforward sense. Post-processing, as in Kaplan's strategy for checking question presumptions when there are grounds for suspicion, may be unfriendly because the user may have the cost and time delay of a large database search before he is told his question is defective, though actual searches are unavoidable for extensional question failures. Apparently friendly behaviour may not be genuinely cooperative either. The system may be able to hypothesise what the user means and, to avoid what the user might see as an irritating interaction about an 'obvious' mistake, correct it automatically. This can be done with syntactic errors, for example. But the system's correction may be wrong, simply because the system's knowledge is inadequate, so it is important to provide feedback. A correction which is obvious to the user may, moreover, be only one of several equally possible ones, so interaction is needed, however tedious for the user. Whether recovery strategies (AJCL 1983) which the system could apply automatically should be so applied is a policy decision relating to the system's context of use, and there are no universally applicable rules for choosing appropriate strategies. So a conservative approach is the safest, given that natural language questions, and especially defective ones, are often ambiguous and the system's capacity to identify and correct errors may be limited by an unavoidable lack of background and contextual information.

Thus setting aside being friendly, what matters is that wherever the user is involved, and especially where the user is required to act because the system is either sufficiently uncertain or completely unable to proceed, the user must be shown what is wrong. This is not a trivial matter. First, because the error may be difficult to locate, or ambiguous: second because it may be difficult to indicate what an internal technical failure in, say, parsing means in terms the user can understand; and third because providing a proper characterisation of what is wrong may require access to other background information providing a context for the specific fault. For example, with "Does Rome supply bolts?", a cooperative response should indicate that people not places supply things.

In the face of these difficulties, the natural approach to adopt is the simple one, providing the user with specific and correct information when it is available and can be readily expressed, as in Kaplan's case; with 'obvious' information, as where a word cannot be found in the dictionary, but leaving it to the user to make inferences about what this may imply and take his own decisions as to future actions; or only to offer some simple statement "Processing failed: try presenting the question another way". But this approach, though feasible, is far from satisfactory, since if the user has done something wrong and is not presented with very much information about his error he may be unable to understand how he has erred and hence how to make an appropriate correction. For example, if the user has asked "Does Jones manufacture green parts?", where the database covers supply but not manufacture, telling the user that "manufacture" is not in the lexicon, or to put his question another way, is not being cooperative. It is clearly highly desirable, if not necessary, to try to give the user more substantial information about the nature of the defect, as the system perceives it, showing him what is wrong and why it is wrong.

But the system needs to know a great deal to explain what is wrong and to indicate what the probably correct line to follow is, where this falls within the scope of the application. Some of this information is required for automatic recovery strategies, whenever the nature of the error does itself suggest the appropriate correction, but more may be needed to explain

an error to the user, and also to present the error-related information in a conceptually and linguistically comprehensible way. The need for material system knowledge is particularly true of domain failures. For example, given the question "Does Jones supply bronze screws?", a cooperative response like "The system has information about the colours and weights of parts, but not about the materials of which they are made" depends on a categorisation of bronze as a material and on a grouping of colour, weight and material as a class of related properties as distinct from, say, name or location, even though the database attributes of parts include these. Substantial supporting knowledge is similarly required to suggest to the user what an appropriate question would be, as in responding to "Does Jones supply large bolts" with "The database does not cover part dimensions, only weights: do you in fact want bolts of above average weight?" For questions with defects like these, that is, the system must have access to domain knowledge, and not just to data model or DBMS information; and this domain knowledge, though it may be called a conceptual schema, will have to be richer and more explicit than conceptual schemas as these are conventionally viewed within the framework of DBMSs. As just illustrated, characterising the nature of an error by making generalisations, or indicating relationships, may require reference to concepts which are well outside the scope of the DBMS itself.

Fortunately, the requirement for a rich and explicit characterisation of the application domain does not stem only from the need to provide the user with information about defective questions. It is also a necessary support for ordinary question processing. Giving the user the opportunity to ask questions in his natural language, so he is not constrained to learn a formal query language, let alone be familiar with the detailed organisation of the database, means that he can put his question in terms remote from the database ones. When the domain and database are richer than the simple Suppliers and Parts example, therefore, translating the question into a formal query may not be just a relatively straightforward mapping, but a much more complex transformation, which can only be achieved via a rich domain description and inference operations on this domain model. It is not simply that input lexical items may not have direct equivalents, or that information which is expressed in one syntactic or structural form may have to be turned into something quite different. The conceptual structure representing the question as input has to be linked, through related intermediate concepts, with the conceptual structure of the corresponding output query.

Suppose, for example, we are working with the complex town planning application used for TQA (Damerau 1980), where the database is about pieces of land and their various properties, including location, owner, etc. The input question "What owners are in block B3?" is a naturally expressed legitimate question, but it cannot be mapped at all directly onto a formal query. It has to be transformed by the semantic processor into the formal equivalent of 'What are the owners of properties in block B3?', which requires an enriched characterisation of the domain of the kind illustrated in Figure 3, amplifying the basic picture supplied by the database schema, along with some sort of inference processor capable of operating on this domain model to make the necessary connections (Boguraev et al 1986).

Front end processing in itself may therefore require knowledge of the kind also needed for cooperative response. But it is clear that as processing becomes more complex, what the structure of the front end processor should be becomes an important issue.

Transportable front ends

Modular processing is established language processing practice. There are good arguments for a modular front end, exploiting the various bodies of knowledge required to interpret questions in successive stages of processing; for example carrying out syntactic parsing before semantic analysis so the syntactic characterisation of the question can be used to guide the semantic operations. As the variety and complexity of the linguistic inputs to be handled increases, a serial strategy has advantages because it makes it easier to control the various subprocesses involved. This is clearly also potentially useful from the cooperative point of view, since it should make it easier to localise and define types of error, and to provide relevant information about them.

But this 'internal' argument for staged operation is powerfully reinforced by the 'external' need for transportability. It is indeed the case that transportability would force a modular structure even without other arguments for it.

Early systems were essentially purpose built for individual applications, even, as in LUNAR, extending to the data access operations. Providing a front end for an existing DBMS imposed the requirement to fit a conventional query language (Waltz 1978), and work with distributed databases (Hendrix et al 1978) showed it was essential to decouple the basic interpretation of an input question from its expresion as a specific database system query, with the link between the two processes made by an abstract formal query: this incorporated the essential content of the query without reference to the specific administrative organisation of the database by an individual DBMS, or to the local DBMS query format, etc.

Additional gains can be made by taking the division of processing into separate stages further. As front ends are substantial objects, there is everything to be said for maximising their transportability (as well, of course, as their portability at the code level). For relatively restricted applications a tightly-engineered and closely-coupled front end may be very efficient. This combines an application-specific semantic or sublanguage grammar for analysing the input with a direct translation into the database search formulation. But just as this strategy suffers from output inflexibility in adapting to changes in the search system, it suffers from inflexibility in relation to variety in the input. So just as it is convenient even for a single database, because its specific organisation may change over time, to have the question interpreter produce a relatively abstract query which is subsequently specialised, it is equally convenient, even for a single database, to attack the input with a general syntactic grammar which is powerful enough to parse a wide variety of textual forms and to reduce them to a smaller range of syntactic expressions for the next step of semantic processing.

This strategy leads to front end modularisation of the type represented by IRUS (Bates and Bobrow 1983) and TEAM (Grosz et al 1987), schematically shown in Figure 4. There is a syntactic processor supported by a grammar and lexicon, a semantic processor supported by a domain model (and the lexicon), and what I shall call a conversion processor supported by a database schema. Each processor builds a representation of the question in an appropriate representation language, the final representation being the search query. The processors themselves may be quite general, like the parser; the various information resources they exploit range from the most general, the grammar, which is in principle transportable across many applications, through the domain model, which holds for a domain and is therefore transportable across specific DBMSs, to the database schema, which is general only to the limited extent that a specific instance of, say, the relational model may be impervious to changes in the implementation. It may indeed be possible to produce a sufficiently abstract

formal query for it to be compatible with different types of data model, so promoting more transportability in relation to implementations of the database.

It has been suggested (Boguraev and Sparck Jones 1984) that this approach can be taken still further so the general linguistic processing includes application-independent semantic analysis, giving the input to the application-dependent processing is a more fully characterised and normalised representation of the question. The underlying claim is that application domains contain considerable general knowledge which can be factored out.

This kind of modular structure, with detached information sources providing an explicit characterisation of the different kinds of knowledge required, and specifically of domain knowledge, is clearly well suited to cooperative retrieval. The fact that processing is staged makes it easier to allocate question defects, and the background information makes it easier to describe them. Thus "Robinson owning parcels?" would be identified as a syntactic error in the planning application, "Does B2 own a hospital?" as a semantic or domain one depending on the particular architecture adopted, though this is still not to imply that determining what is wrong with an input and why or how it is wrong is easy, or indeed also that it is easy to express this finding, and to express it unambiguously, in the user's natural language.

It is also necessary to recognise that there are costs associated with the modular strategy, notably those of establishing the connectivity between different sources which allows transformations, so that, for instance, "wealthier" in "Are there owners wealthier than Carter?" is connected via concepts of assets and valuation with the database's rating assessments of parcels, and of acquiring the possibly extensive information needed. An integrated knowledge source like that used with the CONSUL mail system interface (Mark 1981) makes it much easier to transform one input representation into another through concept connections, and to invoke the conceptual context of an error. But with a single base like this it is very difficult to separate application-dependent from application-independent information. With distinct modules and sources this separation is marked, but then extra care has to be taken to ensure that the necessary links between the various forms of question representation and their associated sources, are provided, and because the modules are distinct the points of connection may not be obvious.

Customisation is recognised as a serious effort, and sophisticated techniques for acquiring the application vocabulary, and supporting semantic, domain and database information have been developed (Ballard 1986, Grosz et al 1987), in some cases without requiring professional linguistic knowledge. But this work has in fact not been directed towards acquiring extensive domain models of the kind needed for really complex domains and for handling error detection and cooperative response in demanding applications, though IRACQ (Ayuso et al 1987), the acquisition component of IRUS, is intended to be able to handle rich models. But it will clearly never be easy to supply an extensive domain model, and to hook application- specific information onto the permanent knowledge in the front end.

Conclusion

Research on interfaces (including database front ends) has led to the development of a variety of techniques for detecting and recovering from input errors (Carbonell and Hayes 1983, Weischedel and Sondheimer 1983). Current commercial database front ends attempt to be robust and to provide for automatic error recovery (Harris 1984), but are more limited in cooperative response. A good deal of effort has also been devoted to techniques for making

front ends more transportable, mainly by adopting a modular structure with, for example, a general-purpose syntactic processor and perhaps a logical form generator, and by supplying a convenient and effective acquisition component for new application information. The work on transportability has not, however, been significantly combined with that on error handling, so though effective customisation techniques for database front ends have been demonstrated, this has not been for ones offering significant recovery and response on error; and in general the semantic and domain knowledge provided for database front ends is less extensive than that which is needed to handle ill-formed input properly.

For database front ends with a captive and informed user community, serious problems with defective inputs are not likely to arise. But restricting front ends in this way is just what offering natural language access is not intended to presuppose, and as defective inputs can therefore be expected there is a real need to identify and respond to errors. This requires substantial semantic and domain knowledge, as Carbonell and Hayes and Selfridge 1986 show, which it is a major effort to provide. Carbonell and Hayes', and Selfridge's interfaces are to command and inquiry systems, and so are of a more complex sort than those required for database front ends, imposing additional requirements to balance the additional information that a dialogue context may provide. But it is evident that a good deal of what is required in the general interface case is also needed in the more limited database one, though, assuming a database system of the conventional sort and no escalation to a more comprehensive inquiry system, it may be possible to provide enough of the necessary system knowledge, and especially domain knowledge, by exploiting very general data modelling notions like those associated with the entity-relationship model (Chen 1976), and to supply appropriate, limited inference processors for operating on this. Modularisation presents challenges here, because of the need to connect one type of knowledge effectively with another, for example linking domain concepts with database terms as in the town planning illustration of Figure 3, but larger bodies of more explicit information that are required for transportability are also needed for robust and cooperative front ends.

This is very much an area of work in progress: in general, as Bates and Weischedel 1987 show very clearly, respectable natural language interfaces to databases can be built now, but we cannot yet provide the really robust, cooperative and transportable front ends we would like, even allowing for the necessary constraints on the scope of user inquiries imposed by the typical database itself.

Acknowledgement

I am grateful to GEC for support through a Research Fellowship, and to Ann Copestake for help with this paper.

System structure

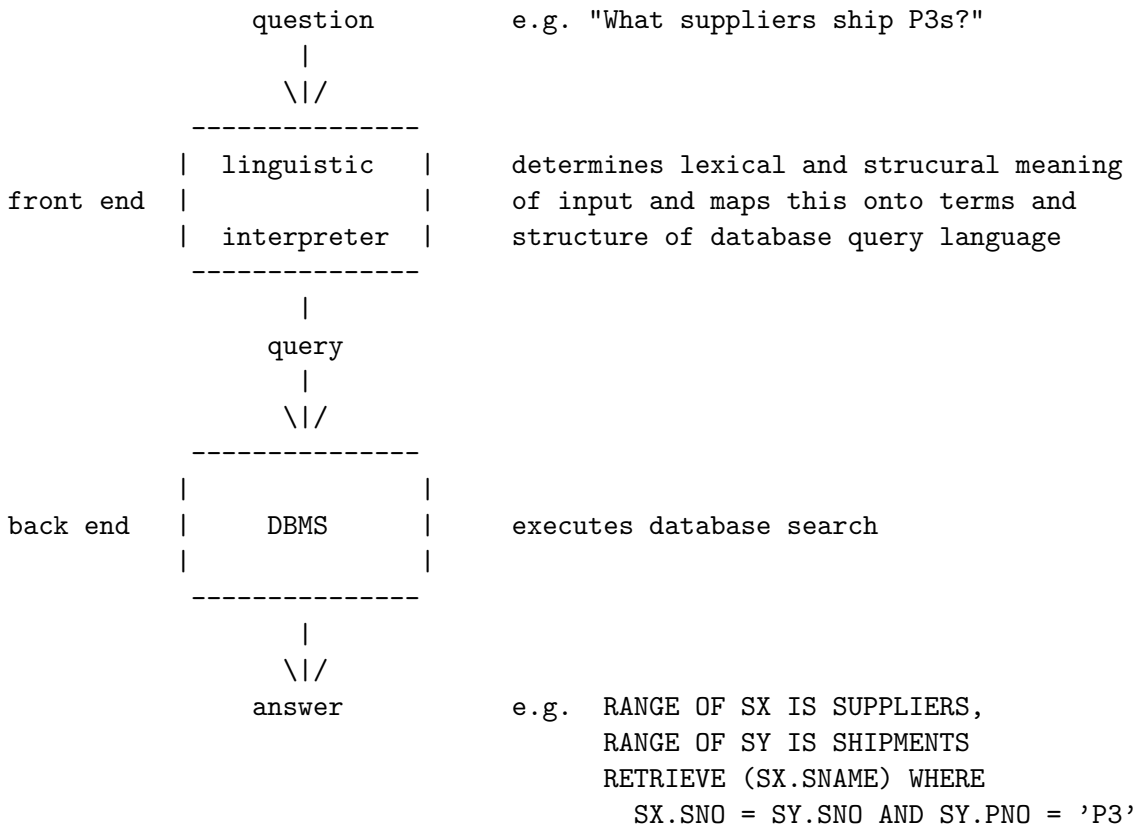


Figure 1

Types of input question defect

Assume the simple Suppliers & Parts database of Date 1977 p 79:
suppliers with number, name, status, city
parts with number, name, colour, weight and city
supplying with supplier number, part number, quantity
(note - Jones does not supply P3s)

A linguistic defects

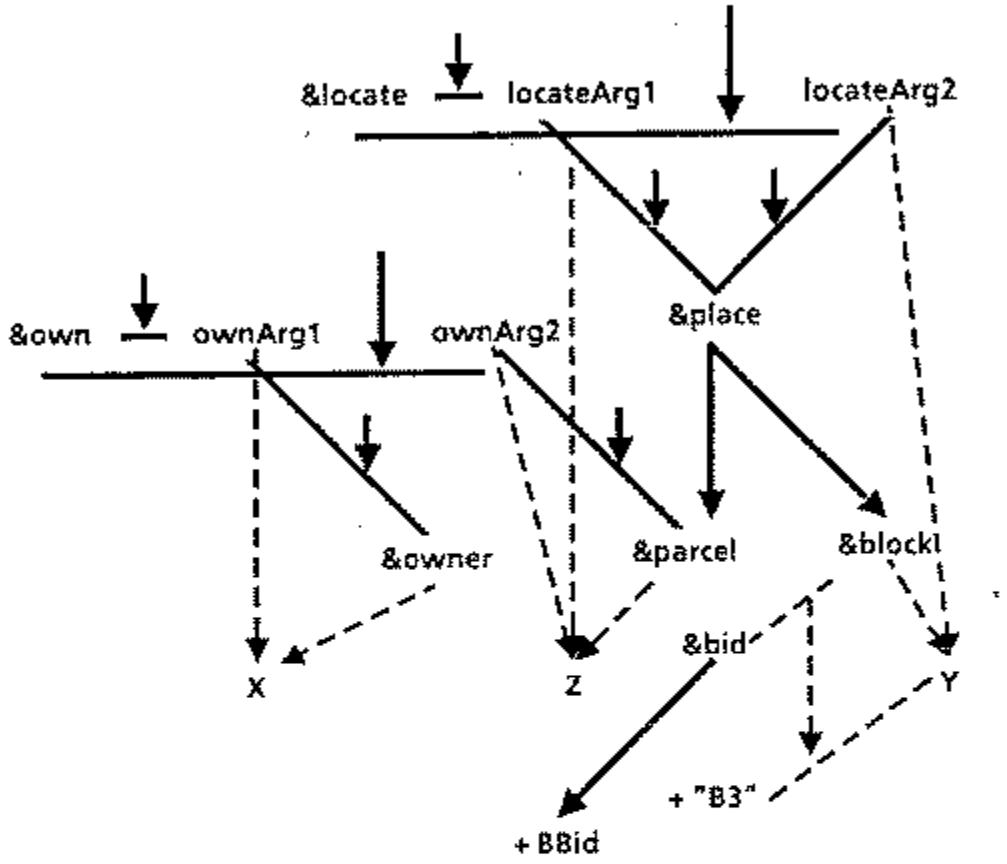
1 clerical : e.g. spelling mistakes
e.g. "Does Jones suply P3s?"
2 syntactic : e.g. number, tense faults
e.g. "Does Jones supplies P3s?"
3 grammatical : elliptical and fragmentary expressions
e.g. "Jones P3s"
4 semantic : selection restriction, case role violations
e.g. "Does Paris city supply P3s?"
5 discourse : speech act improprieties
e.g. "Jones supplies P3s" (i.e. not a question)

B non-linguistic defects

1 intensional : errors about the domain
a) what sorts of objects are known about
e.g. "Does Jones supply nails?"
b) what sorts of properties are possessed
e.g. "Does Jones supply brass P3s?"
c) what sorts of actions are permitted
e.g. "Does Jones make P3s?"
d) what aspects of actions are covered
e.g. "Does Jones supply P3s to Paris?"
2 extensional : errors about the database
a) what objects exist
e.g. "Are Jones P3s in batches of 200?"
b) what properties are possessed
e.g. "Do the green P3s weigh 17?"
c) what actions occur
e.g. "Are the P3s that are shipped by Jones shipped in 300s?"
d) what aspects of actions hold
(not exemplifiable for simple illustrative database)

Figure 2

Domain and database characterisation



key: simple arrows are specialisation links;
arrows pointing to lines are the lower halves of role-owner correspondence relations
dotted lines are temporary additions
&terms are used as intermediate representation terms
+terms are output query language terms
the variable Z referring to parcels represents the link established between X and Y referring respectively to owners and blocks in a full logical form representation of the queation using domain terms to which database terms needed for the search query are also linked.

Figure 3

Front end structure

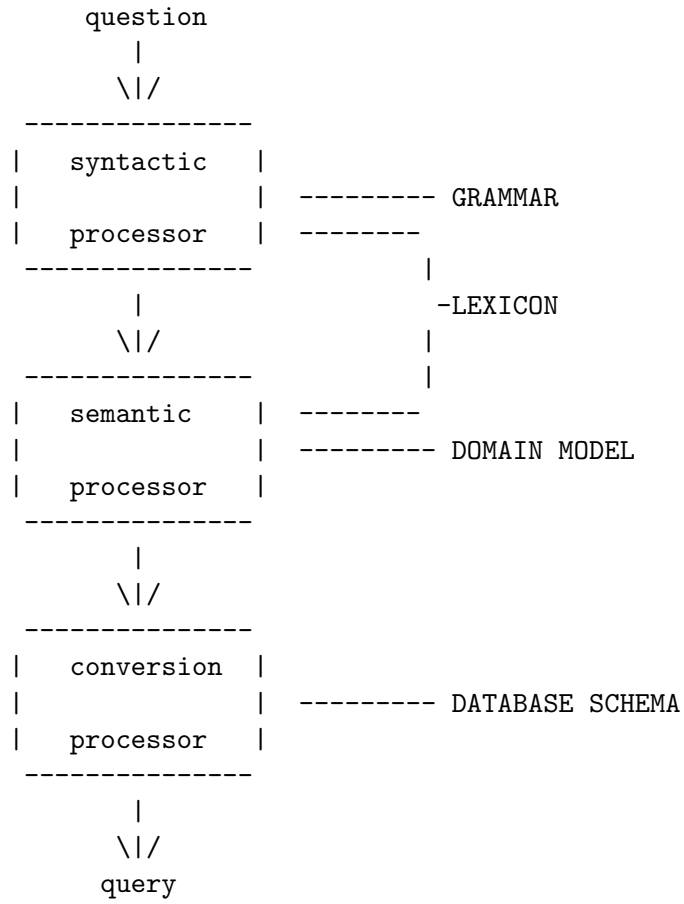


Figure 4

References

AJCL: *American Journal of Computational Linguistics*, special issue on ill-formed input, 9 (3-4), 1983, 123-196.

Ayuso, D.M., Shaked, V. and Weischedel, R.M. "An environment for acquiring semantic information", *25th Annual Meeting of the Association for Computational Linguistics*, 1987, 32-40.

Ballard, B.W. "Semantic acquisition in TELI: a transportable user-customised natural language processor", *24th Annual Meeting of the Association for Computational Linguistics*, 1986, 20-29.

Bates, M. and Weischedel, R. "Evaluating natural language interfaces", Tutorial, *25th Annual Meeting of the Association for Computational Linguistics*, 1987.

Bates, M. and Bobrow, R. "A transportable natural language interface for information retrieval", *Proceedings of the Sixth Annual International ACM SIGIR Conference*, 1983, 81-86.

Boguraev, B.K. and Sparck Jones, K. "A natural language front end to databases with evaluative feedback" in *New applications of databases* (ed Gardarin and Gelenbe), London: Academic Press, 1984.

Boguraev, B.K., Copestake, A.A. and Sparck Jones, K. "Inference in natural language front ends for databases" (1986) in *Knowledge and Data (DS-2)* (ed Meersman and Sernadas), Amsterdam: North-Holland, in press.

Carbonell, J.G. and Hayes, P.J. "Recovery strategies for parsing extragrammatical input", *American Journal of Computational Linguistics*, 9, 1983, 123-146.

Chen, P.P.S. "The entity-relationship model - towards a unified view of data", *ACM Transations on Database Systems*, 1, 1976, 9-36.

Damerau, F.J. *The Transformational Question Answering (TQA) system: description, operating experience and implications*, Report RC8287, IBM Thomas J Watson Research Centre, 1980.

Date, C.J. *An Introduction to Database Systems*, 2nd ed, Reading, MA: Addison-Wesley, 1977.

Green, B.F. et al "BASEBALL: An Automatic Question Answerer" in *Computers and Thought* (ed Feigenbaum and Feldman), New York: McGraw-Hill, 1963.

Grosz, B.J. et al "TEAM: an experiment in the design of transportable natural language interfaces", *Artificial Intelligence* 12, 1987, 173-243.

Harris, L.R. "Experience with INTELLECT", *The AI Magazine*, 5(2), 1984, 43-50.

Hendrix, G.G. et al "Developing a natural language interface to complex data", *ACM Transactions on Database Systems*, 3, 1978, 105-147.

Hendrix, G.G. "Q&A: Already a Success?", *Proceedings of COLING86*, 1986, 164-166.

Joshi, A., Webber, B. and Weischedel, R.M. "Preventing false inferences", *Proceedings of COLING84*, 1984, 134-138.

Kaplan, S.J. "Cooperative responses from a portable natural language query system", *Artificial Intelligence*, 19, 1982, 165- 187.

Mark, W. "Representation and inference in the Consul system", *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, 1981, 375-381.

Selfridge, M. "Integrated processing produces robust understanding", *Computational Linguistics* 12, 1986, 89-106.

Waltz, D.L. "An English language question answering system for a large relational database", *Communications of the ACM*, 21, 1978, 526-539.

Weischedel, R.M. and Sondheimer, N.K. "Meta-rules as a basis for processing ill-formed inpout", *American Journal of Computational Linguistics*, 9, 1983, 161-177.

Woods, W.A., Kaplan, R.M. and Nash-Webber, B. *The Lunar Sciences Natural Language Information System*, Final Report, Bolt, Beranek and Newman Inc., Cambridge MA, 1972.