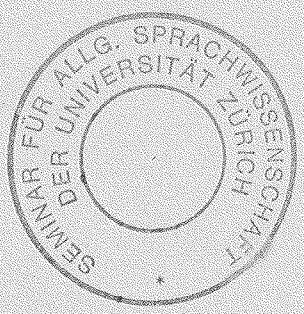
ARBEITEN DES SEMINARS FÜR ALLGEMEINE SPRACHWISSENSCHAFT DER UNIVERSITÄT ZÜRICH

NO. 2

# ABOUT THE ROLE OF CONTROL INFORMATION IN NATURAL LANGUAGE QUESTION ANSWERING SYSTEMS

Michael Hess




1985

# ABOUT THE ROLE OF CONTROL INFORMATION IN NATURAL LANGUAGE QUESTION ANSWERING SYSTEMS

Michael Hess
Seminar of General Linguistics
University of Zurich
Switzerland

## ABSTRACT

Most existing Natural Language Question Answering systems written in a Logic Programming language such as Prolog make use of only one component of the information expressed in the questions, viz. the logic component. However, (many) Natural Language sentences also convey control information. If this type of information is ignored the interpreter must use an input-independent, in most cases fixed, control regime and can not always generate useful replies. We should like to draw attention to a few such cases in the context of Logic Programming: word oppositions like those between "what" and "which" determining whether a generalised solution or a fully evaluated solution is required; topic and comment in queries and DB entries and their use in guiding the search; restrictive and non-restrictive relative clauses and their use in interleaving data acquisition and query evaluation modes. We give a very short outline of a suitable query evaluator. The problem of actually parsing questions is not dealt with here.

## ACKNOWLEDGEMENT

This work was supported under Grant No. 81.703.0.79 of the Swiss National Science Foundation.

To be published in: Dahl,V. and Saint-Dizier,P.,eds., Natural Language Understanding and Logic Programming, North-Holland, 1985

Most Natural Language Question Answering systems written in the context of Logic Programming translate queries into a **logical form**, often cast in terms of straightforward Prolog terms, which are then evaluated direct by the system interpreter over a Data Base consisting of equally straightforward Prolog facts and rules.[1] It is very much in the spirit of Logic Programming to pay attention exclusively to the logical content of a query and to ignore questions of evaluation strategy, i.e. of control, at least in the first stage of designing a system. Moreover, the small and straightforward Data Bases used in those systems are kept very simple as they serve only as example material for the systems to work with. Under these conditions questions of control are fairly immaterial anyway and can be ignored even more safely. The success of many of these systems is ample proof that the concentration on the logical content of NL queries was good research strategy.

But NL as such is not pure logic; in the words of Kowalski:[2] "Natural Language = Logic + Control". Translating NL queries into a logical form very often means ignoring potentially valuable control information.[3] Sometimes this will only mean that the general-purpose evaluation procedure will be **less efficient** than one which makes use of the specific control information provided by the user. However, it often means that we will **not** get, in principle, **the type of answer** from a QA system which we intuitively expect. We will use examples from both categories but we will put more emphasis on the second, more serious, case. This will also be the topic of section 1.

## 1. FULL EVALUATION VS. GENERALISED SOLUTIONS

### 1.1 WHICH VS. WHAT

The two questions

1) **Which managers at IBM earn $ 100000 ?**
2) **What managers at IBM earn $ 100000 ?**

seem to call for two different types of answers,[4] namely

1a) **Hart, Miller, and Jones.**
2a) **Managers in charge of a branch office.**

In the first example the whole Data Base must be searched for entries of explicitly known managers, their salaries must be checked, and the names of all the managers meeting the criterion must be listed. The salary of a manager may be known explicitly, i.e. in the case of a Prolog DB be given as a fact (as in the case of Hart and Miller in Fig.1), or it may have to be

---

[1] This is the approach chosen, for example, in Pereira and Warren's system Chat-80, arguably the most powerful QA system written in Prolog so far (Pereira 1983).

[2] Kowalski 1975:129

[3] as emphasised by Klahr 1980:113 and Nilsson 1980:193-195

[4] as with most other problems of NL understanding we find a first solution of this problem in Winograd's SHRDLU (Winograd1972:164).

computed via a general inference rule (as in the case of Jones). In the second example it is this **general rule** itself which is the more appropriate reply rather than the list of individual instances.

Obviously we don't want a QA system to reply in full "All managers at IBM in charge of a branch office earn $ 100000"; it should return only those parts of the rule which were not given in the query. But this is the result of filtering out information secondarily, on the basis of dialogue considerations. It doesn't change the fact that the rule had to be accessed as a whole, as a fact, rather than executed, as in example 1. This kind of reply is sometimes called a "**generalised solution**" (Kowalski 1975:184). The control decision whether a rule should be executed, or treated as a mere fact to be returned, is made by the user through the choice of the question word, i.e. "which" vs. "what". It is a decision no interpreter, however intelligent it may be, can make on its own.[5]

```
manager(ibm,hart).
manager(ibm,miller).
manager(ibm,jones).

runs(branch_office,hart).
runs(branch_office,miller).
runs(branch_office,jones).

salary(hart,100000).
salary(miller,100000).

salary(X,100000) :- manager(ibm,X),
                    runs(branch_office,X).
```

**Figure 1: A sample Data Base**

We can paraphrase 1 and 2 as 1b and 2b which intuitively have exactly the same meaning although they may be slightly less acceptable:

**1b) Which are the managers at IBM that earn $ 100000 ?**
**2b) What are the managers at IBM that earn $ 100000 ?**

Now we can see that we ask in both cases (of both versions of the two questions) for the **set of all objects** with certain properties, as expressed by the plural definite article.[6] However, we expect a different type of description for the set: In the first case ("which") we want an **enumerative**,

---

[5] which is basically what the variable-depth NL understanding system outlined in Kayser 1981 tries to do in a similar situation.

[6] We are not saying that the meaning of a plural definite article can be captured completely by a Prolog setof-predicate but it's a good approximation commonly used.

**extensional description**, in the second case ("what") a **constructive, intensional** one.

To further support our claim that "which" and "what" are two control versions of the same logical expression we look at two cases where only one of them can be used without difficulty.

In 3 and 3a we ask for a **unique object**, as expressed by the singular definite article, and reinforced by the superlative. 3 is quite normal as this constraint is compatible with the control information expressed by the interrogative pronoun, viz. with the demand that the result be an exhaustive enumeration of instances: It will simply be a list with exactly one member. Contrary to that 3a sounds odd because the interrogative pronoun asks for a general rule whereas article and superlative ask for a unique instance, and it is just a bit pointless to have a general rule generate a unique object. If we force ourselves to give 3a an interpretation we might come up with a description of the country which is general in character but at least consists of a sufficient number of constraints to make unique identification likely: "It is a constitutional monarchy, an island, and a reluctant member of the European Community".

3) **Which is the European country with the longest coastline** ?
3a) ? **What is the European country with the longest coastline** ?

The different control component in question words such as "which" and "what" also explains why 4 is perfectly normal

4) **What is a country** ?

where the reply is a general rule ("something is a country if it is a political, national, and geographic unit"), while 4a

4a) ? **Which is a country** ?

is quite odd and requires, in order to become acceptable, a reference to an exhaustively listed set from which the answer can be picked:

4b) **Which (one, of the following, ...) is a country:**
**Lichtenberg, Liechtenstein, or Lichtenau** ?

When referring to people we can, in addition to the distiction between **indefinite reference** and **definite reference**, as expressed by "which" and "what", use the further category of "**individual reference**", through the use of "who". Now we ask for an individual, known by name if possible:

5) **What's her husband? He is a film director.**
5a) **Which is her husband? He is the man smoking a pipe in the corner.**
5b) **Who is her husband? He is Paul Jones.**

Note that 5 is precisely the kind of question mentioned as a bit pointless above. But 5 is an (almost idiomatic) special case: One's profession is, in our culture, considered the most important general criterion for social categorization, and "What's her X" asks specifically for X's profession. This becomes clear when we consider 5d

**5d) ? What's her teacher?**

which is definitely odd, and precisely because the (only conceivable type of) answer is given in the question itself.

One problem in this context has not been mentioned yet: What is a QA system supposed to do if we ask an indefinite question ("what") but there is no general rule in the DB, only individual instances? The easiest solution is to make the system treat an indefinite question as a definite question in a situation like that; if we removed the rule about salaries from the DB of Fig. 1 this would indeed be the only sensible behaviour. With a richer DB a more ambitious solution is conceivable: As before, the list of individual instances could be computed, but then the system could try to form equivalence classes from the list of instances, or even synthesize a rule which could generate this list.

Finally it is, of course, conceivable that the system finds several rules when looking for an answer to an indefinite question. Then it is reasonable to list them all as reply.

## 1.2 EVERY VS. ALL VS. EACH

Basically the same distinction of full evaluation vs. generalised solutions holds between "every" and "all" although the difference between them seems to be felt less clearly than in the case of "which" and "what".[7] "Each" finally seems to be a third control version of the same underlying universal quantifier.

**6) Does every manager of an IBM branch office earn $ 100000 ?**
**6a) Do all managers of an IBM branch office earn $ 100000 ?**
**6b) Does each manager of an IBM branch office earn $ 100000 ?**

When asking question 6 we would expect a QA system to check every single instance of an entry about IBM managers, and then check the entry for his or her salary, **irrespective** of whether there is a universal rule about managerial salaries in the DB or not. The system should not fail if, for some managers, there is no entry at all about their salaries, provided there is a general rule it can execute; the system should then reply something like "nothing is known about Jones, but in general yes".

When asking question 6a we would be satisfied if the system found a general rule. A general rule says, of course, nothing about the existence of its antecedents (it does **not** say, in Fig. 1, that there are managers at IBM in the first place). However, this is clearly presupposed by the question, and the user therefore does not require the system to check the individual cases.

---

[7] However, the terminology frequently used to describe these words shows that there is a definite difference in their meaning: Leech/Svartvik 1979:50 call "each" and "every" (as opposed to "all") "distributive", Saint Dizier 1884:43 "universal distributive quantifiers" (as opposed to simple "universal quantifiers"). Jespersen 1974:599 describes "every" as "all, taken separately".

Only if no general rule can be found the system must test the individual cases and answer, if appropriate, "as it happens yes, but not necessarily so".

Questions with "each", such as 6b, seem to require an exact "matching" between the domain of the quantifier on the on hand, and the objects or events referred to in the restrictions on the other hand. If such a matching isn't obvious a sentence becomes odd, as the comparison between examples 7 and 7a shows; in our culture you have husbands one at a time, which makes it clear that there are separate events of admiration which can be matched with the husbands one to one. In 7a, however, the uncles aren't pre-arranged in any discernible order, and so it isn't clear how the admiration is to be distributed among them. In order to make 7a unproblematical we must add information which explicitly creates different types of admiration, one for each uncle, as in 7b:[8]

**7) Marge admired each of her husbands.**
**7a) ? Marge admired each of her uncles.**
**7b) Marge admired each of her uncles in a different way.**

We can model these differences rather closely in terms of control information: A QA system, trying to find an answer to an "each"-question, should look only for **base facts** in the DB to be "matched" against each other (in our example: facts about managers and facts about salaries) but it should **not use** any **inference rules** to compute either domain or restriction terms (e.g. to infer that Jones must also earn $ 100000).

## 1.3 ANY VS. SOME

"Any" and "some" can also be seen as the two control versions of one and the same logical expression, viz. of the existential quantifier.[9] The conventional explanation is that the meaning of "some" and "any" is the same but that "some" is used in assertive positions and "any" in non-assertive positions (i.e. in questions, negations, conditionals, comparisons). However, "any" can be used in assertive positions ("Any colour will do"),[10] and "some" can be used in non-assertive positions, although in both cases this atypical use results in a particular connotation of the resulting sentence: In the case of the non-assertive "some" this connotation consists in a clear positive bias,

---

[8] McCawley 1981:98

[9] That "any" isn't a straightforward existential quantifier is shown by the following example where the intonation alone can turn it either into a universal or into an existential quantifier: "I don't lend my books to anybody." With a rise-fall-rise tone on the "any" the sentence means "I don't lend my books to **just** anybody", with a high-falling tone "I don't lend my books to anybody **at all**". (Jespersen 1974:606)

[10] This is a simplification: They are not ordinary assertive positions. They always have some kind of "generalised", even "modal", connotation: "any colour **will** do", "you **can** take any colour", "you **may** come any day, but you must come some day" (Jepsersen 1974:604).

whereas "any", used in the same position, is neutral: 8 clearly is neutral, whereas in 8a the speaker knows for sure that there are female managers at IBM that make $ 100000, and the question is only whether the addressee knows them.

**8) Do you know any female managers at IBM that earn $ 100000?**
**8a) Do you know some female managers at IBM that earn $ 100000?**

In 8a those female managers at IBM are treated as a group, and the addressee is expected to know them as a group: the famous four female top-managers at IBM, mentioned in all the newspapers, etc. He is not expected to know any individual instances. In 8, on the other hand, it is precisely individual instances he is asked about.

One more example to highlight the difference:

**9) Did you get any post cards last week?**
**9a) Did you get some post cards last week?**

We could easily imagine a continuation of 9a "... which I sent you from Italy", which doesn't sound right for 9.[11]

The situation with "some" and "any" is an almost exact parallel to the relationship between "all" and "every". In both cases the first word, when used in a query, looks for a **whole group, or set** of objects, known (and assumed by the speaker to be represented in the receiver's mind) as **one single complex entry**, whereas the second word looks for **multiple simple entries** for multiple simple facts. And in both cases we have to use the same strategy if we can't find such a complex entry: we must then look for simple entries scattered in our mental DB, i.e. we have to treat all-questions and some-questions as every-questions and any-questions, respectively. And if, in this situation, the answer is in the affirmative it will usually be preceded by a phrase such as "as it happens" to indicate that the receiver couldn't simply "look up" the answer but had to compute it from scattered bits of information.

## 2. ERROR HANDLING: VIOLATIONS OF EXISTENTIAL PRESUPPOSITIONS

It is well known that missing axioms are a source of trouble in Logic Programming and in common sense reasoning likewise. In some cases the absence of a piece of information may be a perfectly legitimate state of affairs (of which we make good use when we interpret negation as non-provability), but in other cases it is an "error" in our DB, i.e. we don't know something we ought to know.

---

[11] The same distinction is made, incidentally, between "already" and "yet":

**10) Has she gone to bed yet?**
**10a) Has she gone to bed already?**

Particularly counter-intuitive is the situation where we try to prove a universally quantified statement using double negation, i.e. for 11 we have to prove 11a:

**11) Does every manager drive a Cadillac?**
**11a) not(manager(X), not(drives(X,cadillac))).**

If there are no entries about managers in our DB the proof will succeed at once, but really "for the wrong reason".

In some implementations of Prolog we can tell the interpreter how to react when an axiom is found missing during a particular sub-proof: to either simply fail the sub-proof, to abort the whole proof with an error message, to fail but also issue an error message, etc.

Common sense, interpreting NL statements, shows a similar behaviour; we don't treat violations of existential presuppositions the same way we treat negations (given explicitly or as failure). English has special expressions for that: "There are no managers at IBM **to begin with**" or "**in the first place**", as opposed to "no, they don't".

Any useful QA system must make this distinction (as indeed most do).[12] If a system, during the evaluation of an **every- or each-question**, cannot find an entry for any one of the **domain terms** it should fail and **report the violation of an existential presupposition** instead of merely failing the proof. A missing **restriction term** should, of course, make the subproof simply fail. As an **all-question** asks for an explicit all-rule the system doesn't first have to care whether individual instances are also listed in the DB. However, if it cannot find any all-rule, and if it must resort to a search of the DB for individual instances, it must also report violations of presuppositions (i.e. it must treat the "all"-query **exactly** the same way as an "every"-query).

Analogous is the situation with "what"- and "which"-questions, and with "some"- and "any"-questions.

## 3. ORDER OF EVALUATION: TOPIC AND COMMENT

There is yet another way of conveying control information in questions but it is almost too obvious to be noticeable: it is the distinction between topic and comment[13] by means of word order. Questions 12 and 12a

**12) Do any American female managers earn $ 100000 ?**
**12a) Do any female American managers earn $ 100000 ?**

are, of course, logically equivalent, as are all the possible commutations of their translations into Horn-clause 12b

**12b) ?- manager(X), female(X), american(X), salary(X,Y), 100000<Y.**

---

[12] Berry-Rogghe 1980:192-195, Berry-Rogghe 1979:293, Bronnenberg 1980:253. Cf. also Kaplan 1979.

[13] Both Berry-Rogghe 1980:165 and McKeown 1983 deal with the role of topic and comment in QA systems.

The order of words codes the order of evaluation which is considered most efficient by the speaker. However, the two orderings do not necessarily have the same direction: In English the order of adjectives and nouns in a NP is exactly the inverse of the most efficient order of evaluation, whereas post-modifiers already have the right ordering, as shown in 13 and 13a

13) **Do any female American managers in their thirties with children above the age of four ...**
13a) **manager(X), american(X), female(X), age(X,30+), child(X,Y), age(Y,4+), ...**

If we represent the topic-comment distinction of **assertions** in an equally straightforward manner, viz. as the ordering of entries in the DB, the standard interpreter of Prolog will automatically use the most efficient strategy to evaluate the terms of a query, provided they are themselves arranged according to topic and comment.

A general query optimisation program could try to rearrange the terms of 12 on its own in such a way as to minimise the size of the search space, using estimates about the cardinality of the sets of entries in the Data Base.[14] However, even this rather sophisticated approach does not make use of the control information supplied by the user in the question, and this can be crucial if there are very general terms, such as "small" or "yellow", in a question - it will be next to impossible to give a reasonable estimate of the cardinality of the set of entries about yellow things in a DB.

## 4. QUERY MODE VS. ASSERTION MODE: RESTRICTIVE VS. NON-RESTRICTIVE PHRASES

One more way to express control information in NL should be mentioned. QA systems don't normally deal with declarative sentences at all. If they do the system has, as a rule, two completely separate modes of operation: one for query answering, and one for data acquisition. However, in some cases NL mixes these two modes freely, for instance when using relative clauses:

14) **Do managers who earn a lot pay a lot of taxes?**
14b) **Do managers, who earn a lot, pay a lot of taxes?**

While 14 is simply a query where the restrictive relative clause adds one more restriction, in 14b the **non-restrictive relative clause** functions as an embedded declarative sentence in a question. The speaker wants to make sure that the receiver knows certain relevant facts before he/she/it sets out to answer the question. While processing the query the receiver has to go from answering mode into data acquisition mode for a short while.

Quite complex combinations of processing mode information and mixed evaluation depth information are possible (although some of the examples sound awkward):

15) **What managers, some of whom pay taxes, are well-paid?**
16) **Which employees, whose bosses all earn a lot, are underpaid?**

---

[14] Warren 1981

17) **Is a manager every employee of whom is underpaid unpopular?**

or much better:

17a) **Is a manager unpopular if every employee of his is underpaid?**

Not only relative clauses can be restrictive or non-restrictive: appositions (18, 18a) and postmodifiers (18b,18c) can be used to make the same distinction:

18) **My friend Peter was here last night.**
18a) **My friend, Peter, was here last night.**
18b) **The substance discovered by accident which had the greatest impact on medicine is penicillin.**
18c) **The substance, discovered by accident, had an enormous impact on medicine.**

## 5. REPRESENTATION OF THE DATA AND OUTLINE OF AN INTERPRETER

If we want to have an input-dependent, flexible control regime which allows a system to access base facts one time, general rules another time, to issue different error messages in case axioms cannot be found during a proof, and to go from query mode to data acquisition mode and back again, we have to change the standard way of representing data, and we also have to modify the interpreter. In order to keep changes to the absolute minimum we simply sketch a small interpreter on top of the system interpreter.

The first requirement is that the interpreter should be able to access inference rules as facts. Thus rule 19 becomes 19a[15]

19) **salary(X,100000) :- manager(ibm,X), in_charge_of(X,branch_office).**

19a) **all(X, [manager(ibm,X),in_charge_of(X,branch_office)] , salary(X,100000)).**

As long as these all-expressions are nothing but Horn-clauses turned into Prolog facts, they should not have more than one term in the third argument position, and a conjunction of goals has to be written as follows
all(X, [manager(X,ibm)] , salary(X,100000)).
all(X, [manager(X,ibm)] , drives(X,cadillac)).

We can, of course, still use explicit inference rules as long as we do not require them to be available as generalised solutions. This may be the rigth thing to do for information about the type hierarchy, such as 20

20) **animal(X) :- dog(X).**

---

[15] This representation is now widely used for quantified formulas; it makes the restriction on the variable, its range, explicit in the second argument, as opposed to the classical quantifiers. Cf. Moore 1981:9, and Pereira 1983:21.

20 is, to us humans, "obvious" in the same way as the grammar rules of our native tongue: In order to retrieve such a rule we must generate an example, and then abstract, from the proof tree of the specific example, what the underlying general rule must look like.

We already stressed the close parallel between all-statements and some-statements as far as their content of control information is concerned: Both have a positive bias, i.e. they presuppose the existence of whole contiguous "chunks" of data, of structured objects,[16] and both will look for individual DB entries corresponding to individual terms in a structured object only in case such a structured object cannot be found. Accordingly we turn the Horn-clauses

```
kangaroo(sk1).
in(sk1,africa).
striped(sk1).
```

for "some kangaroos in Africa are striped" into

```
21) some(sk1, [kangaroo(sk1), in(sk1,africa)], striped(sk1)).
```

The second argument carries the topical information, the third argument is used for the comment.

Information about individuals is represented the usual way:

```
manager(jim).
american(jim).
drives_a_cadillac(jim).
```

An interpreter would accordingly have to prove an all- or some-query by first looking for a **matching** all- or some-entry in the DB and second, if that fails, **evaluate** the expression:

```
demo(Goal,Answer)  :-  candidate(Goal,Candidate), match(Goal,Candidate),
                       difference(Goal,Candidate,Answer).
demo(Goal,Answer)  :-  evaluate(Goal,Answer).

candidate(all(X,Y,Z),all(U,V,Z))       :-  all(U,V,Z).
candidate(some(X,Y,Z),some(W,V,Z))     :-  some(U,V,Z).

evaluate(all(X,Y,Z),as_it_happens_yes) :-  not(presup(Y),not(demo(Z,_))).
evaluate(some(X,Y,Z),as_it_happens_yes) :-  presup(Y), demo(Z,_).
```

presup(G) is the same as demo(G,_) only it will report missing axioms as presupposition violations.

Matching all-expressions with each other is interesting. If we want to prove 22, given in Predicate Calculus notation,

```
22) ALL X(kangaroo(X) AND in(X,australia) AND female(X)) → brown(X).
```

---

[16] Nilsson 1980:361-415 uses the term for the representation of data in the form of semantic networks, but then semantic networks are only a convenient way to visualise logic anyway.

it becomes, in the notation used here,

22a) ?- **all(sk1,[kangaroo(sk1),in(sk1,australia),female(sk1)],brown(sk1))**.

(note that the matcher expects universally quantified variables in a query to be skolemized). This term will have to match (e.g.) with the DB entry

23) **all(X, [kangaroo(X), in(X,australia)],brown(X))**.

i.e the comments of the two terms ("brown(X)" and "brown(sk1)") have to be a direct match but it is sufficient that the **set of range constraints** of the DB entry be a unifiable **subset** of the constraints of the query. (By "unifiable subset" we understand a set which contains only members which are unifiable (not necessarily identical) with some members of a second set, i.e. the kind of subset we get automatically if we use the regular Prolog definition of "member".) We would, obviously, have to make sure that properties are inherited the usual way through the hierarchy of types, but this problem is of no interest in the present context.

Matching all-expressions through the computation of the subset relation between their topics can be seen as the **simulation on the meta-level of a direct object-level proof**. If we want to prove 22, we have to negate it and to transform it into clausal form, which will give 22b

22b) **kangaroo(sk2), in(sk2,australia), female(sk2), :- brown(sk2)**.

22b can be used as a **direct proof for the existence of an inference rule** 22c

22c) **brown(X) :- kangaroo(X), in(X,australia), female(X)**.

provided we interpret the unnegated terms in 22b as temporary additions to the DB, to be removed after the execution of the proof, and the negated expression (":-brown(sk2)"), of course, as goal to be proved. Given the uniqueness of the Skolem constant this proof can only succeed if rule 22c is in the DB. Additional unnegated terms in the query are harmless: They are additions to the DB which are never accessed. This is what the subset-operation of the matcher modelled on the meta-level.

If we want to match a some-query against a some-entry, everything is inverted. Obviously we will have the unbound variables in the query and the Skolem constants in the DB entry, and the set of range constraints of the DB entry has now to be a unifiable **superset** of those of the query. The query 24 must match the DB entry 25

24) ?- **some(X,[manager(X),american(X)],drives(X,cadiallac))**.
25) **some(sk3,[manager(sk3),american(sk3),female(sk3)],**
**drives(sk3,cadillac))**.

In a direct proof, reordering DB entries and query terms is a simple way to improve efficiency, as mentioned above in the remarks on topic and comment. The ordering of the terms in the range constraints of both all- and some-expressions was made to preserve the ordering of the original query terms and DB entries. Thus we retain this gain in efficiency for direct proofs in their meta-level simulation: The computation of sub- and supersets proceeds from left to right, simulating the most efficient sequence for the direct proof sequence.

The distinction between every- and each-questions can be made as follows:

```
demo(every(X,Y,Z),yes)  :-  not(presup(Y), not(demo(Z,_))), !.
demo(each(X,Y,Z),yes)   :-  not(candidate(Y,U), match(Y,U),
                                not(candidate(Z,V), match(Z,V))), !.
```

What-queries ask for (all) the rules (all-statements) matched by the query; the reply is the set of matched rules. If no rules can be found the query is treated as a which-query. A which-query evaluates to the set of instances found.

```
demo(what(X,Y,Z),Answer)  :-  ( setof(B, (candidate(all(X,Y,Z),C),
                                          match(all(X,Y,Z),C),
                                          difference(C,Y,B)),Answer) ;
                                demo(which(X,Y,Z),Answer) ).
demo(which(X,Y,Z),Answer) :-  setof(Y, (presup(Y), demo(Z,_)), Answer).
```

For data-acquisition mode we can, of course, use the system predicate "assert". We will have to make sure that existentially quantified variables are first skolemized. 14b ("Do managers, who earn a lot, pay a lot of taxes?") should accordingly translate into

```
11b) assert(all(X, [manager(X)] ,earns_a_lot(X))),
     demo(all(sk4,[manager(sk4)], pays_lotsa_taxes(sk4)),Answer).
```

whereas 15 and 16, repeated here for convenience, should translate into 15b and 16a, respectively:

```
15) What managers, some of whom pay taxes, are well-paid?
16) Which employees, whose bosses all earn a lot, are underpaid?
15b) assert(some(sk5,[manager(sk5)],pay_taxes(sk5)),
     demo(what(X, [manager(X)], well_paid(X)),Answer).
16a) assert(all(X, [manager(X),employee(X,Y)], earns_a_lot(X))),
     demo(which(X, [employee(W,U)], underpaid(U)), Answer).
```

Examples of declaratives embedded in other declaratives are simpler:

```
26) Some car dealers, who are greedy, are crooks.
```

as opposed to

```
27) Some car dealers who are greedy are crooks.
```

and

```
28) Car dealers, some of whom are crooks, are greedy.
```

as opposed to

```
29) Every car dealer who is greedy is a crook.
```

ought to translate into one or several "assert"-commands:

```
26a) assert(some(sk6, [car_dealer(sk6)] ,crook(sk6))),
     assert(all(Y, [car_dealer(Y)] ,greedy(Y))).
```

Note that the default interpretation of a plural as a universal quantifier is used for the relative clause: **all** car dealers are said to be greedy or else it would be said otherwise with an explicit quantifier, as in 28

```
27a) assert(some(sk7, [car_dealer(sk7), greedy(sk7)] , crook(sk7))).
```

28a) **assert(all(X), [car_dealer(X)] , greedy(X))),**
**assert(some(sk8), [car_dealer(sk8)] , crook(sk8))).**
29a) **not(demo(car_dealer(X),_), demo(greedy(X),_), not(assert(crook(X)))).**

29a is interesting in that it will add, for each car-dealer found to be greedy, a separate elementary fact "crook(X)":

**crook(jim).**
**crook(peter).**
**crook(bill)**
**etc.**

There will, however, be no explicit "every"-entry, true to the definition of "every".

Almost as a side-effect the data representation and the interpreter outlined here can cope with certain sentences which otherwise cause problems: Example 30 would be translated into Predicate Calculus as 30a

**30) Managers whose employees are underpaid are unpopular.**
**30a) ALL X(manager(X) AND ALL Y(employee(X,Y) → underpaid(Y))**
**→ unpopular(X))**

It translates, because of the implication in the antecedent, into the **non**-Horn-clauses

**30b) unpopular(X) :- manager(X), underpaid(sk1(X)).**
**unpopular(X), employee(X,sk1(X)) :- manager(X).**

The second clause, with a disjunction in its head, cannot be processed by the Prolog interpreter. Also, the clausal form is highly un-intuitive. If we represent it by means of uninterpreted embedded all-statements we can use it direct as a DB entry

**30c) all(X, [manager(X), all(Y, [employee(X,Y)] , underpaid(Y))] ,**
**unpopular(X)).**

which is also very close to the form of the original NL statement. If our query is

"**demo(unpopular(jim),A)**"

the interpreter will access 30c) by means of the top "demo"-rule of the interpreter, and then work its way down through the embedded terms trying to find generalised solutions wherever possible, otherwise evaluating the terms. If it has to fully evaluate every subterm of 30c) this DB entry is interpreted the same way as the usual translation of 30) into Horn-clauses-cum-negation-as-failure

**30d) unpopular(X) :- manager(X), not(employee(X,Y), not(underpaid(Y))).**

would have done it.

If we want to prove the **general rule** that managers whose employees are underpaid are unpopular, we must turn 30) into the query 30e

```
30e) demo(all(sk6,[manager(sk6),
              all(sk7,[employee(sk6,sk7)],underpaid(sk7))],
                                          unpopular(sk6)),A).
```

If the DB is as follows

```
manager(jim).
unpopular(jim).
all(X, [employee(jim,X)] , underpaid(X)).
```

but 30c itself is not in the DB, we will get the reply "yes, as it happens", by partial evaluation of the subterms of the query. Finally, if the DB contains only base facts such as

```
manager(jim).
employee(jim,joan).
underpaid(joan).
```

we will have to compute the answer as if the query had been given in the standard, fully reduced, form, i.e.

```
30f) ?- not(manager(X), not(employee(X,Y),
                not(underpaid(Y))),not(unpopular(X))).
```

Different is the case of question 31

**31) Is a manager unpopular if every employee of his is underpaid?**

which must be translated into 30f right from the beginning.

## 6. CONCLUSIONS

Many questions provide a QA system with valuable control information which either **must** be used to generate a pragmatically useful reply, as opposed to a merely logically correct reply, or it **can** be used to prevent the system from performing a search which is very unlikely to succeed. There must be many more types of control information conveyed by NL which might be put to good use in a similar way: Adverbs and adverbial constructions ("what are, generally speaking, the..."), "meta-nouns" ("the number of", "average value of"), adjectives (attributive adjectives for existential presuppositions).[17]

In this approach, some work is shifted from the parser to the query evaluator; all the quantifier and question terms mentioned, such as "every", "all" , "which", etc. are transferred unchanged from the text to either the DB entries or the query, and their embedding structure is also maintained (and taken care of by the query evaluator). Nevertheless the design of a parser dealing with the phenomena outlined will still not be a simple affair.[18]

---

[17] v.Hahn 1980:183 investigates this question in some detail.

[18] Porto 1984:228-232, made a beginning for a similar approach.

## 7. BIBLIOGRAPHY


Berry-Rogghe, G.L./Dilger,W., 1979. *Konzeption eines Terminterpreters.* in: Kolvenbach 1979, 289-304.

Berry-Rogghe, G.L, et al.,1980. *Interacting with PLIDIS a deductive question answering system for German* in: Bolc 1980, 138-220.

Bolc,L.,ed.,1980. *Natural Language Question Answering Systems.* Hanser: 1980.

Bolc, L., ed., 1980a. *Natural Language Based Computer Systems.* Hanser: 1980.

Bronnenberg, W.J.H., et al., 1980. *The Question Answering System PHILIQA1.* in: Bolc 1980, 217-305.

v.Hahn,W., et al., 1980a. *The Anatomy of the Natural Language Dialogue System HAM-RPM.* in: Bolc 1980a, 119-253.

*ISLP 1984.* International Symposium on Logic Programming, Feb 6-9 1984, Atlantic City. IEEE Computer Society Press: 1984.

Jespersen, O., 1974. *A Modern English Grammar* Part VII, Syntax. Allen and Unwin: 1974.

Kaplan, J. 1979. *Cooperative Responses from a Portable Natural Language Data Base Query System.* PhD Thesis, CS Dept., U. of Pennsylvania: 1979.

Kayser, D./ Coulon, D. 1981. *Variable-Depth Natural Language Understanding* in: Proceedings IJCAI-81: 64-66

McKeown, K.R., 1984. *Paraphrasing Questions using Given and New Information.* in: AJCL, vol.9, Nr.1, 1984, 1-10.

Klahr, Ph./ Travis, L./ Kellogg, Ch., 1980. *A Deductive System for Natural Language Question Answering.* in: Bolc 1980: 74-136.

Kolvenbach, M./ Lötscher, A./ Lutz, H.D., eds. *Künstliche Intelligenz und natürliche Sprache.* Narr: 1979

Leech, G./ Svartvik, J., 1975. *A Communicative Grammar of English.* Longman: 1975.

Kowalski, R., 1975. *Logic for Problem Solving.* North Holland: 1975.

McCawley, J.D., 1981. *Everything Linguists have Always wanted to Know About Logic* Blackwell: 1981.

Moore, R.C., 1981. *Problems in Logical Form.* SRI International, Technical Note 241.

Nilsson,N.J., 1980. *Principles of Artificial Intelligence.* Tioga: 1980.

Pereira, F.C.N., 1983. *Logic for Natural Language Analysis.* SRI International, Technical Note 275.

Porto, A., et al., 1984. *Natural Language Semantics: A Logic Programming Approach.* in: ISLP 1984: 228-232.

Quirk, R./ Greenbaum, S./ Leech, G./ Svartvik, J. *A Grammar of Contemporary English.* Longman: 1979

Saint Dizier, P., 1984. *Quantifier Hierarchy in a Semantic Representation of Natural Language sentences.* This volume.

Warren, D.H.D., 1981. *Efficient Processing of Interactive Relational Data Base Queries Expressed in Logic.* in: 7th International Conference on Very Large Data Bases, 1981.

Winograd, T. 1972. *Understanding Natural Language.* Edinburgh U.P.: 1972