Submitted to the Logic Programming Workshop, Portugal, June 83

KBO1: A Knowledge Based Garden Store Assistant

Adrian Walker

IBM Research Laboratory
San Jose, California, USA

Antonio Porto

Universidade Nova de Lisboa
Lisboa, Portugal

ABSTRACT

This paper describes a prototype knowledge based system which answers English questions about some of the products supplied by a garden store. The system answers questions such as

what can I use to kill snails ?

is there anything I can use that will fertilize my lawn ?

what can I use to kill weeds in my lawn in spring ?

does product A kill dandelions in less than 20 days ?

in less than one second each, and it produces a helpful phrase when it cannot answer a question.

The syntax, semantics and knowledge needed by the system are written in the language Prolog. The behavior of the system indicates that Prolog appears to be a good language for the construction of practical knowledge based systems which can answer questions in ordinary English.

1. INTRODUCTION

This paper describes KBO1, a prototype knowledge based system which answers English questions about some of the products supplied by a garden store. The system matches, to a certain extent, the behavior of a helpful, knowledgeable assistant in a store which sells products such as domestic pesticides and weed-killers.

Some questions which produce brief, but useful and accurate answers, are:

what products do you sell ?

what is each product that you sell for ?

what can I use to kill snails ?

is there anything I can use that will fertilize my lawn ?

what can I use to kill weeds in my lawn in spring ?

what can I use to kill weeds around my fence ?

do I need a sprayer to use product A ?

what is the response time of weeds to product A ?

does product A kill dandelions in less than 20 days ?

The present system only answers questions about certain house and garden products. Questions which lie outside this scope, such as

is there a bus stop near here ?

are answered with a sentence such as

I'm sorry, I don't know the word: bus

The system has been written to explore the feasibility of building useful knowledge bases in the programming language Prolog (1, 2). Prolog appears promising as a notation for implementing knowledge based natural language systems, since knowledge rules and grammar-like rules can be written down and executed more or less directly (3, 4, 5).

In the case of our KBO1 system, it was not necessary to write an inference engine or a parser. Both of these items were covered by the use of the built in inference method of Prolog. We have used an efficient implementation of Prolog (7) on an IBM mainframe computer. In terms of coverage of a part of English, our results are promising. In terms of performance, we find that no question takes more than 1 second to answer, even when our computer is heavily loaded with work by other users. Our coverage and performance results are consistent with those reported by (3, 5, 9), namely, that efficient Prolog programs can be written for useful natural language access to knowledge bases.

Section 2 of this paper describes the input-output behavior of the KBO1 system by means of some annotated examples. Section 3 outlines the internal design of the syntactic and semantic components of the English interface of KBO1, while Section 4 describes the knowledge base. Section 5 consists of conclusions and directions for future work.

2. INPUT-OUTPUT BEHAVIOR OF KBO1

A user of the KBO1 system types English questions on a keyboard, and gets answers on a screen. In the following example session, a question starts after a prompt symbol ">>" and ends with a question mark "?". The answer produced by the system follows the question. Comments which we have added appear in parentheses after answers.

```
>>
what products do you sell ?

product A
product B
product C
product D
product E
product F
```

(the system has simply listed the products.
for examples with more products, it would print
just the kinds of products)

```
>>
what do you know about ?

product A kill weed
product B kill pest
product C kill pest
product D kill insect
product E kill insect
product F kill weed
product F fertilize lawn
```

(the system has listed the products in its knowledge base,
with one or more intended uses for each product. English
output generation is minimal)

```
>>
what is each product that you sell for ?

product A kill annualweed
product A kill perennialweed

product B kill gardenpest

product C kill gardenpest

product D kill crawlinginsect
product D kill flyinginsect
product D kill hoppinginsect
```

product E kill crawlinginsect
product E kill flyinginsect
product E kill hoppinginsect

product F kill annualweed
product F kill perennialweed
product F fertilize lawn

(this question has put more emphasis on intended use than the last, so the system has answered in more detail)

>>
when should I use product F ?

spring
fall

>>
what can I use to kill snails ?

product B
product C

>>
what can I use to kill snails on my vegetables ?

product C

(the product product B is not for use on food, and so is not suggested in this answer)

>>
what can I use to kill weeds ?

product A
product F

(these are the suitable products known to the system)

>>
what are the weeds ?

annualweed
perennialweed

(the system has listed, without much detail, the kinds of weeds)

>>
what are weeds ?

sorry, I don't know

(we have reached a limit of the knowledge in the system)

>>
what are the perennial weeds ?

bahiagrass
bull thistle
clover
dandelion
kentucky bluegrass
poison ivy
poison oak
yellow oxalis
oxalis

(the system has understood this as a request for a detailed answer)

>>
is there anything I can use that will fertilize my lawn ?

product F

(a suitable product is suggested)

>>
what can I use to kill weeds in my lawn in spring ?

product F

(the same product can be used)

>>
what can I use to kill weeds around my fence ?

product A

(product F is for lawns, but elsewhere we can use something stronger)

>>
do I need a sprayer to use product A ?

yes

>>
can I use product A without a sprayer ?

sorry, no

>>
what are the precautions I should follow in using product A ?

avoid contact with eyes
keep children and pets away till spray has dried

avoid contamination of food
if accidental spray then wash off with water

>>
what is the response time of weeds to product A ?

annualweed 5 days
perennialweed 15 days

>>
what is the response time of each product that kills annual weeds ?

product A 5 days
product F unknown

>>
what can I use on dandelions ?

product A
product F

>>
what can I use on dandelions that will kill them in less than 20 days ?

product A

>>
does product A kill dandelions in less than 20 days ?

yes

>>
does product A kill dandelions in less than 2 days ?

sorry, no

This session shows that the KBO1 system has considerable knowledge of the properties of a few products and their intended uses. As with all knowledge bases known to us, there are limits to the domain which is covered. However, when a question cannot be answered, the user sees reasonable replies such as "I'm sorry, I don't know", or "I don't understand the word: bus". Although the domain of competence is much smaller than that of most adult people, these phrases carry about the same information as an immediate reply from a person who does not know the answer to a question.

It would appear worthwhile to extend KBO1 so that it would explain its answers when asked to do so. For example, it would be helpful to know why product B cannot be used to kill snails on vegetables. The techniques described in (8) could be used to do this.

In this section, we have described the input-output behavior of KBO1. The next section outlines the design of the English interface, while section 4 describes the knowledge base.

3. THE ENGLISH INTERFACE

Our two main goals in designing the English interface were simplicity and modularity. We wanted to keep the interface simple so as to be able to get a system working within a short time. On the other hand, we made the interface modular so that it can form a basis more elaborate English input processing. The interface uses full dictionary lookup for every word in the sentence. Thus it is distinct from a 'keyword' interface, in which some words may simply be ignored.

Our main simplification is to make lexical analysis deterministic. This means that only one morphological token is associated with each word, an assumption which does not hold in general, but which which turns out not to be a serious limitation in our present application. This simplifying assumption allows us to complete the lexical analysis before starting a syntactic parse. In a later version of the interface we would expect to drop the deterministic assumption, but to retain modularity and efficiency by using coroutining techniques in Prolog (6).

Some other design desicisions which we made to keep the English inteface simple were as follows.

We only deal with fairly simple ellipsis, namely an elided subject in a conjunction of verb phrases. We feel that any serious treatment of ellipsis would need to manipulate information from several sentences in a dialog. The present system works one sentence at a time.

We do not generate a syntactic tree. Rather, we construct a semantic representation directly during parsing. This is efficient, since semantic checks are made early in the anaylsis of a sentence, helping to prune unfruitful parses.

We do not treat extraposition, although some common cases of left extraposition can be handled by an easy extension of the present system.

We also do not make a syntactic check of gender and number agreement, since we have found that it can only be used constructively in some rare cases in which it may disambiguate an attachment problem. (Even then, semantic checks may be enough.) In the same vein, no analysis is made of verb tense.

We made the English interface modular by clearly separating a number of functional components, and by classifying each component as either specific to our present application, or general. The main components are: a lexical analyser, a dictionary, a syntactic parser, semantic rules, and an output package. The lexical analyser and the syntactic parser are general purpose. The dictionary and the semantic rules each have a general subcomponent and an application-specific subcomponent, while the output package is application specific.

The lexical analyzer reads a question from the terminal, groups the characters into words, and looks up the words in the dictionary to find the corresponding morphological tokens (e.g. noun, verb, preposition).

As mentioned above, the dictionary consists of a general and a specific part. The general part contains entries that are likely to be needed in any application in English, such as articles and the forms of the verb 'to be'. The specific part contains entries for items such as product names for the KBO1 application.

The syntactic analyzer consists of a set of rules, written in the manner of a definite clause grammar (4). The rules are executed top-down by Prolog's built-in inference mechanism, hence there is no distinct syntactic parser component in KBO1. The bodies of the grammar rules contain the usual calls to nonterminal symbols, and also contain calls to the semantic rules that construct a representation of the meaning of the sentence which is being parsed. If a fruitless parse is being attempted, the calls to the semantic rules will fail, causing a new parse to be sought before too much effort has been wasted.

The function of the semantic rules is to translate groups of syntactic items into a semantic representation of a question. The semantic representation, which we describe in detail below, is a Prolog statement roughly equivalent to 'the set of all X such that p(X) is true in the knowledge base', where X and p may be structured terms. In particular, p may contain further set-formation terms. The general part of the semantic rules component treats items such as quantification and the verb 'to be' which would be needed in most applications, while the specific part deals with items such as the kinds of objects to which it is reasonable to apply domestic chemical products.

The output component uses the information which is retrieved from the knowledge base, together with syntactic information about the question which caused the retrieval, to generate the answer that is displayed on the screen.

Since KBO1 is designed to answer questions, the syntactic component of the English interface only accepts questions. These may be wh-questions, (such as 'what...', 'when...', 'how...' etc.) or nexus questions ('does...', 'is there...', etc). Verbs are accepted both in the active and the passive forms, and they can be followed by any number of complements. Nouns can be preceded by any number of adjectives, and double nouns are accepted (e.g. 'poison ivy'). Nouns can have simple complements ('persistence of product A') or double complements ('response of bluegrass to product A'). The syntax covers the usual articles and prepositions, a few pronouns ('you', 'them', 'anything', ...) and some auxiliary verb forms ('can', 'should', ...). Relative clauses are accepted, and there can be conjunctions of relative clauses, verb phrases, or verb complements.

The concepts represented inside the parser are of two types, which we call Entities and Properties. A complete meaning representation, in terms of Entities and Properties, is constructed from the morphological token stream by the syntactic and semantic rules. Entities are either Objects, or sets of Objects, the latter being represented by 'set(Q, O)', where Q is a quantifier ('each', or 'all') and O is an Object.

Objects can be named, or can be defined. A named Object is represented as 'T:X', where T is the name of the type of the Object. If X is a free variable, then we say that the Object is abstract, that is, it is an unspecified Object of type T. If X has a value, then we say that the Object is concrete, and that X is its name. Thus 'perennialweed:X' is an abstract Object, while 'perennialweed:clover' is a concrete Object.

Defined Objects have the form

Abstract_Object ! Property

(read 'Abstract_Object such that Property') where the Abstract_Object is T:X and X appears in the Property.

A simple Property is just a Prolog predicate. Properties may also be quantified, in the form

for(Qantifier, Object, Property)

Conjunctions of Properties are also Properties.

Here are some concepts and their internal representations:

product_A	item:product_A
a weed	weed:X
the weeds	set(each, weed:X)

all weed killers

```
set(all, item:I ! use(I, kill-weed:W, S))
```

persistence of each product

```
time:T ! for(each, item:I, use(I, *, persistence(T)))
```

There are semantic predicates that link concepts to form new concepts. They link subject and verb, verb and object, verb and complement, and adjective and noun. The domain-independent part of the definition of these predicates deals with the handling of quantification and the verb 'to be', while the domain-dependent part contains only quantifier-free definitions.

After a syntactic and semantic parse succeeds in producing a data strucure corresponding to an input question, the data structure is transformed into a Prolog query which can be applied to the knowledge base. The essence of the transformation is to insert a call to the Prolog meta-predicate 'all', which computes the set of all items which satisfy a given property.

For example, for the question

'what is the response time of weeds to product A ?'

the query

```
set(response):R !

   all(T,use(product_A,kill-weed:any,response(T)), R)
```

is generated. This query, when executed against the knowledge base, retrieves the answer to the original question and binds the answer to the variable R.

Special care was taken to give informative, rather than yes-no, answers to nexus questions. To see that this is essential, rather than simply desirable, consider the question

'do you sell anything that kills bluegrass ?'

Very few people would be satisfied with only the answer 'yes', so the system generates the query

```
set(item):I ! all(P, use(P, kill-weed:bluegrass, M), I).
```

which retrieves a list of suitable products.

On the other hand, a yes-no question such as

'does product A kill dandelions in less than 20 days ?'

is translated into the query .

```
yesno ! use(product_A, kill-weed:dandelion, response(T)) &

typedIt(T ,20.days)
```

This section has described the design of the parts of the KBO1 system that translate a question in English into a query for the knowledge base. The next section describes the knowledge base.

4. THE KNOWLEDGE BASE

The last section described the mapping of an English question into either a form

yesno ! p(X)

or a form

set(items):X ! p(X)

where p(X) is, in general, an abitrary Prolog goal expression. This section describes the underlying knowledge base which provides answers to the mapped queries.

The knowledge base consists of two components, both of which are domain-specific. The first component contains an is-a hierarchy, while the second contains knowledge about the products and how they may be used.

The hierarchy contains assertions such as

```
setname(weed)
setname(annualweed)
weed(annualweed)
annualweed(bluegrass)
```

together with an immediate membership predicate 'mem', a transitive membership predicate 'member', and an 'isa' predicate. Thus mem(bluegrass, annualweed) holds, while mem(bluegrass, weed) fails, but member(bluegrass, weed) holds. Similarly isa(bluegrass, bluegrass), isa(bluegrass, annualweed), and isa(bluegrass, weed) all hold. The 'mem' predicate is also written in infix form as ':', e.g.

mem(bluegrass, annualweed)

is written annualweed:bluegrass.

The type-hierarchy is actually a directed acyclic graph rather than a tree, as there are statements such as

```
homepest(fly)
flyinginsect(fly)
```

The hierarchy allows questions to be answered at an appropriate level of detail. For example

'what can I kill with product A ?'

yields the answer 'weeds', while

'what weeds can I kill with product A ?'

yields

annual weeds
perennial weeds

The main body of knowledge about the products is stored in the second component of the knowledge base. It consists of

(i) an input component, which maps sub-queries of the form

use(Subject, Verb-Object, Modifiers)

into calls to some basic clauses about the products,

(ii) the basic clauses themselves, e.g.

can_use(product_A, kill, weed:Y:Z) <- weed:Y:Z

and,

(iii) an output component which contains information about which kinds of questions (yesno, set) require which kinds of answer format.

The knowledge base is used as follows. A query such as

'what should I use in spring to kill weeds in my lawn ?'

is presented to the knowledge base as

```
set(item):X !
      all(S, use(S, kill-weed:any,
          environment(plant:lawn).season(spring)), X).
```

Here all(S, use(S,....), X) returns in X the set of all subjects S such that one can use S for the indicated purpose. The call

use(S, kill-weed:any, environment(plant:lawn).season(spring))

is mapped by the knowledge base into

```
can_use(S, kill, weed:any) &
environment(S, kill, weed:any, plant:lawn) &
season(S, kill, weed:any, spring)
```

Thus a subject S is retrieved if it can be used to kill some Object O, the Object O matches weed:any in the type hierarchy, and the modifiers environment and season are satisfied.

The knowledge base contains basic clauses such as

```
can_use(product_F, kill, weed:Y:Z) <- weed:Y:Z
environment(product_F, kill, weed:*, plant:lawn).
season(product_F, *, *, spring).
```

which cause the original call to use(S, ...) to succeed with S = product_F.

As mentioned above, the general form of an internal query to the knowledge base is

```
use(Subject, Verb-Object, Modifiers)
```

where Modifiers is a list made up from some of the predicates: environment, season, persistence, response (e.g. response time of a weed to a weed-killer), ingredient, assume, precaution, directions (i.e. directions for use, and how (precautions and directions). The modifiers in the list may be negated, and the list denotes a conjunct.

The predicates for how, precaution and direction are special in that they store English text which can be retrieved, but which cannot be checked in detail. Thus one can ask 'what are the directions for using product A ?' which yields the query

```
set(direction):S !
      all(D, use(product_A, V-O, direction(D), S)
```

and the answer

```
apply with hand trigger sprayer
one application kills most weeds
less effective if rain within 24 hours
```

However the question 'does product A kill most weeds in one application ?' yields 'Sorry, I don't understand'.

The remaining predicates can be used either for retrieval or for checking, and there is some overlap between these and the retrieval-only predicates. Thus the question 'what vegetables can I spray with product D ?' yields the query

```
set(vegetable):S !
      all(V, use(product_D, V-O,
            environment(vegetable:V).
            assume(equipment(sprayer))), S)
```

which retrieves into S the answer 'any'.

One can also ask the checking question 'can tomatoes be sprayed with product D ?' which yields a yesno query similar to the one above, but for

```
environment(vegetable:tomato)
```

and leads to the answer 'yes'.

The use of retrieval-only predicates to store English sentences is mainly a matter of convenience. If detailed questions about, e.g., directions for

the use of a product, were expected, then the knowledge could be moved to predicates which could be used both for retrieval and checking.

To summarize, the knowledge base consists of a hierarchy together with specific knowledge about products and their uses. An incoming internal query from the English interface is transformed into a Prolog goal, the goal is executed against the knowledge, and the result is sent to the output component of the system.

5. CONCLUSIONS AND DIRECTIONS FOR FURTHER WORK

The KBO1 system is at present a prototype. Our experience in bringing it to its present level of behavior indicates that Prolog is well-matched to the task of building a knowledge based natural language system. The system answers non-trivial questions in under one second of real time on an IBM mainframe computer.

While certain simplifications were made in order to build a demonstrateable system in a short time, the English language interface performs full dictionary lookup and parsing. The system was built in a modular manner, and we have separated the reusable parts from the domain dependent parts.

Adding new words and their meanings the system is rather straightforward. Many extensions to the syntactic parser could be made without having to change other parts of the system. For example. we could improve the present treatment of left extraposition just by modifying the parser. In fact, 'what' is already treated like an extraposed noun, and 'when' like an extraposed complement.

A major improvement would be to handle anaphora, mainly ellipsis and pronoun reference beyond the scope of one sentence. However, this is still a research area needing much work. An interesting point is that people sometimes make outside references not only to a previous question, but also to previous answers. To resolve such references, we must have access to a representation of the previous answers.

Another interesting enhancement would be the treatment of cardinal and fuzzy quantifiers. This would require that we modify the semantic rules that handle quantification, and that we define the equivalent of the 'all' meta-predicate for the new quantifiers. A nice possibility is a generalized 'all' meta-predicate with an extra argument which would impose conditins on the number of solutions.

In summary, the direct representation of syntax, semantics and knowledge in the language Prolog appears to be a good approach to the construction of useful knowledge based systems which can answer questions in ordinary English.

6. REFERENCES

(1) Clocksin, W. F. and C. S. Mellish, Programming in Prolog. Springer-Verlag, 1982.

(2) Kowalski, R. Logic for Problem Solving. North-Holland Publishing Co., 1979.

(3) McCord, M. Using slots and modifiers in logic grammars for natural language. Artificial Intelligence 18, (1982), 327-367.

(4) Pereira, F. C. N. and D. H. D. Warren. Definite clause grammars for language analysis - a survey of the formalism and a comparison with augmented transition networks. Artificial Intelligence 13, 1980, 231-278.

(5) Pereira, L. M., P. Sabatier and E. Oliveira. Orbi: an expert system for environmental resources evaluation through natural language. Proc. 1st Int. Logic Programming Conference, University of Marseilles, France, 1982, 200-209.

(6) Porto, A. Epilog: A language for extended programming in logic. Proc. 1st Int. Logic Programming Conference, University of Marseilles, France, 1982, 31-37.

(7) Roberts, G. M. An implementation of Prolog. M.S. thesis, Department of Computer Science, University of Waterloo, 1977.

(8) Walker, A. Prolog/EX1: An inference engine which explains both yes and no answers. Proc. 8th Int. Joint Conf. on Artificial Intelligence, Karlsruhe, to appear.

(9) Warren D. H. D. and F. C. N. Pereira. An efficient easily adaptable system for interpreting natural language queries. DAI Research Paper No. 155, Department of Artificial Intelligence, University of Edinburgh, 1981.