PROLOG AND LANGUAGE ANALYSIS INTELLIGENT RESPONSE TO COMPREHENSION REPLIES

William Butcher & John Galletly with Andrew Wong

University of Buckingham

Introduction : The aim of the present article (1) is to describe a program for evaluating the correctness of simple English sentences in certain key contexts. Our initial thinking was influenced by a previous program (2), which grew in turn out of a conviction that many offerings in CALL and language-processing in general were prone to mechanicalness and rigidity. The user is aware that the computer is not carrying out any linguistic analysis or contextual response, and consequently the possibility of dialogue is lost.

Word-processing was a vital stage on the road towards real language processing. With the addition of multilingual spell-checkers and hyphenation, search and replace and conditional macros, together with CD-Rom-based or hard-disc-resident monolingual and bilingual dictionaries, word-processing has proved itself essential in humanities research and teaching. But because there is still no real interaction between the system and the single-word items input, there is little possibility for 'added value', for the machine to contribute positively to the process of communication.

Taking our cue from other areas, where some machine intelligence has been demonstrated (3), sometimes even creativity, where expert systems may at least attempt to solve problems holistically, we decided to seek some sort of flexible feedback.

This project used C-Prolog, a language which is highly suited to advanced applications. Although based on logic programming, Prolog is particularly adapted to non-numeric programming, including natural languages (4). It presents advantages of user-friendliness and concision, being above all descriptive (relational) rather than procedural: its general approach towards solving a given problem is to describe known facts and relationships in terms of goals to be satisfied rather than a particular sequence of steps.

---

(1) This article is based on a talk given at the conference on CALL and Evaluation at the University of Exeter in September 1989.

(2) J. E. Galletly & C. W. Butcher with J. Lim How, 'Towards an Intelligent Syntax-Checker', in Cameron, K. C. (ed.) (1989) *Computer Assisted Language Learning. Program Structure and Principles*, Intellect Press (Blackwell Scientific Publishers), pp. 81-100. This took the form of investigating areas of French syntax, rewriting the grammar rules in computer-comprehensible terms, and thus producing a program able to respond semi-intelligently to relatively free input.

(3) Despite R. Last's (1989) disillusionment about computer-based language learning and the general role of machine intelligence (*Artificial Intelligence Techniques in Language Learning*, Ellis Horwood, p. 99).

(4) Clocksin, W. F. & Mellish, C. S. (1987) *Programming in Prolog*, 3rd ed., Springer-Verlag.

Our project had relatively broad syntactical objectives while constraining the input context to that of replies to comprehension passage questions. The aim was to have a prototype program powerful enough for different sorts of comprehension exercises but also perhaps for processing free input; and at the same time precise enough to cope with most of the expected answers to the particular comprehension questions asked.

Spell-checking : The program was designed to run on an HLH Orion minicomputer using the Unix 4.2 BSD operating system. The first stage was dealing with the individual words input by the user, by means of a spell-checker. A commercially available dictionary was chosen, with ease of access and its low-to-medium number of 24,000 words, including some proper names.

The first problem was speed of access. Each word was sequentially searched for in the complete dictionary list, meaning up to 24,000 accesses, which took an unacceptable average of 3 minutes. Improvement was obtained by temporarily storing sections of the dictionary in memory. The first stage was to divide the dictionary into 26 'buckets', one for each letter of the alphabet. But each of the 26 buckets had then to be divided into four parts, based on the second letter of the word in question. The dictionary structure was organised in terms of an 'AVL tree' and even more sophisticated mathematics for estimating their efficiency (5). In practical terms, the main result was that access time was reduced in the first stage to about 12 seconds, and finally to about 5 seconds.

Morphology : Next came the realisation that it was extremely naive to believe that all English words appeared in dictionaries, whether in machine-readable or printed form. English is a semi-morphological language, with prefixes including *in-*, *im-*, *pre-* and *un-*, but also suffixes governing verb inflexions and plurality of nouns. There are also variations between British and American usages.

This last problem has received poor treatment in general from computer-based methods. At worst, American usage is forced on one; at best, one has a British version or else a choice. What would be ideal of course would be to have either usage separately, both usages together, or 'translation' from one 'language' to the other. In the present case, however, the quick solution was adopted of simply adding the most common British spellings and usages to the American dictionary.

As for prefixes, the rules are weird and wonderful, often causing problems to native speakers. To have machine-explicit rules for all cases would clearly be a boon, especially to foreign learners. Our solutions were governed by the lacunas in existing systems, by difficulties in taking the existing systems apart, but also by a desire to emphasise this rule-based aspect of language-learning.

It is reasonable to hope that a big dictionary (100-120,000 words) could cope with relatively rare prefixes like *im-* or *ante-*. But the combination of *un-* + adjective is still productive, with the result that no dictionary can list all possibilities. Nor can one simply allow any *un-* combination, with or

(5) Knuth, D. E. (1973) *The Art of Computer Programming*, Vol. 3, *Sorting and Searching*, Addison-Wesley, p. 453.

without a hyphen, for cases like **ungreen* (6) or **uneach* are clearly unacceptable. We are not aware of any satisfactory solution to this problem...

The problem of irregular verbs was solved by explicitly including all forms, e.g. *take, took, taken, taking* and *takes*. The biggest obstacle encountered was that of morphological endings like *-s, -ly, -er, -est, -ing* and *-ed*. This general problem of suffixes is clearly finite, for each English word has at most 10 or 15 forms, and one solution is the sledgehammer one of listing all forms explicitly. Unfortunately, this was not the solution adopted by dictionaries accessible to us, undoubtedly for reasons of data compression.

A major benefit of the explicit, rule-based approach we adopted instead was to pinpoint that forms like **comed* and **comeed* are attempts to form the past of *come* and thus to be able to display a precise explicatory message to the user. As regards *-ing* and *-ed*, the general solution forms what has been called 'junction analysis'. Words ending with *-eing* are usually incorrect, with exceptions, however, like *seeing* and *shoeing*. The most efficient solution was to list the cases where the infinitive ending simply receives the suffix *-ing*, from *agreeing* through to *whingeing*. Then any other string *xxxing* was well-formed if *xxxe* was an infinitive. But words like *thing* are also well-formed. We used a wildcard search on an existing dictionary, and hence listed all words (except verbs...) ending with *-ing*.

Similar methods were used to deal with double consonant problems, both in cases like *hopping* and *hoping* (7) and in the past-tense forms in *-ed*. The distinction between *rodeos* and *potatoes* could clearly only be treated by an exhaustive listing. Terminal *-x*, *-ch* and *-y* also required explicit rules and sub-rules (8).

The above problems are encountered in many natural language projects; our aim was to have a working system which would make explicit the practical rules of spelling morphology in English — and thus present clear advantages in an educational environment. Users are impressed if errors like *comming* or *carryed* are detected within free input and corrected with reference to the particular word.

Sentence analysis using Prolog and Definite Clause Grammar: The way was then open for parsing the sentence. The choice here was between top-down and bottom-up techniques, with top-down ones seeming preferable for ease of writing and speed of implementation. What we sought was a grammar as a collection of 'rewrite rules' specifying which sequences of words are syntactically acceptable.

One sort is 'Context-Free Grammar' (see Figure 1). Very briefly, in CFG the individual words are specified as 'terminals'; the Chomskian rewrite rules successively break down the sentence into a noun phrase and a verb

(6) The * indicates an ungrammatical form.

(7) The word *xxxkking* was judged correct if *xxxk* (where k = consonant) was a permissible infinitive. The word *xxxking* was judged correct if *xxxke* existed.

(8) *-x* and *-tch* take *-es* in the plural, unadorned *-ch* normally plain *-s*; carrying causes no problem; terminal *-ky* (*k* = consonant) gives *-kied* in the past, whereas *-vy* (*v* = vowel) gives *-vyed*.

phrase, and eventually into determiners, nouns, verbs, etc. (the terminals). The left-hand side of each rule consists of exactly one term. The tree diagram (with the 'leaves' as terminals) shows clearly the underlying logical structure of the sentence; and is therefore especially appropriate for recursive forms like 'The key of the door of the house that Jack built...'.

```
sentence = noun phrase + verb phrase
noun phrase = determiner + noun
verb phrase = verb + noun phrase
determiner = the
noun = cat
noun = fish
verb = eats
```

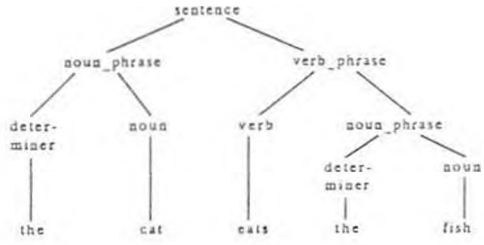


Figure 1

To sum up the desirable characteristics of CFG: 1) the grammar rules are described in a modular way; 2) there is a feature allowing the representation of the recursive embedding of phrases; and, 3) there is an established body of results on CFG which is very useful in designing parsing algorithms.

But CFG is 'context-free': it is difficult for contextual information to be taken into account. In particular, number arguments (singular and plural), agreements and tenses cannot easily be integrated.

Fortunately, there exists a category of grammar which retains the three desirable characteristics, while integrating contextual information and reproducing the essential structure of Prolog: Definite Clause Grammar (9). The advantages of Definite Clause Grammar are clear on reading the two versions of the same program below(10).

```
sentence(S0,S) :- noun_phrase(S0,S1), verb_phrase(S1,S).
noun_phrase(S0,S) :- determiner(S0,S1), noun(S1,S).
verb_phrase(S0,S) :- verb(S0,S1).
verb_phrase(S0,S) :- verb(S0,S1), noun_phrase(S1,S).
```

(9) Pereira, F. C. N. & Warren, D. H. D. (1980) 'Definite Clause Grammars for Language Analysis', *Artificial Intelligence*, 13, pp. 231-78.

(10) Derived from Clocksin and Mellish (1987).

```
adjective(S0,S) :- adj(S0,S1), adjective(S1,S).
adjective(S0,S).
determiner([the|S],S).
noun([boy|S],S).
noun([apple|S],S).
verb([eats|S],S).
adj([young|S],S).
```

```
sentence ⟶ noun_phrase, verb_phrase.
noun_phrase ⟶ determiner, noun.
verb_phrase ⟶ verb.
verb_phrase ⟶ verb, noun_phrase.
adjective ⟶ adj, adjective.
adjective ⟶ [].
determiner ⟶ [the].
noun ⟶ [boy].
noun ⟶ [apple].
verb ⟶ [eats].
adj ⟶ [young].
```

Figure 2

The first program is in ordinary Prolog, whereas the second is in DCG. We would claim that DCG is especially well-organised, readable and concise. Two details confirm this impression. Unlike standard Prolog programs, DCG does not require 'arguments'; and its treatment of recursion is particularly elegant. Thus the way of coping with an indefinite number of preceding adjectives is simply to have the clause 'adjective' invoke itself until no further adjectives are found.

DCG can, more generally, not only provide a description of some of the basic grammar of English, but it is, above all, extremely powerful in use since it is an executable program of Prolog(11). By means of a well-proven standard Prolog compiler, DCG can be compiled into efficient code. It is difficult to overemphasise the practical advantages of this additional simplification to what is already a user-friendly language. The programmer can think in familiar terms of the Chomskian diagrams, convert this to grammatical forms like those in Program B, and his work is finished. The system directly implements the program by converting it successively to standard Prolog and machine code.

In sum, Definite Clause Grammar formalism provides for three important linguistic mechanisms: 1) the building of structures such as parse trees; 2) the treatment of context dependency; and, 3) allowing general conditions on the constitution of words and phrases.

As a simple example of the second facility of contextual information, consider the two ungrammatical sentences: *The boys eats an apple; and, *The boy eat an apple.

---

(11) Colmerauer, A. 'Metamorphosis Grammars', in Bolc, L. (ed.) (1978) *Natural Language Communication with Computers*, Springer-Verlag, and Kowalski, C. A. (1979) *Logic for Problem Solving*, North-Holland.

To introduce the concept of singular/plural, one adds to the second program:

noun (singular) ⟶ [boy]	noun (plural) ⟶ [boys]
noun (singular) ⟶ [apple]	noun (plural) ⟶ [apples]
verb (singular) ⟶ [eats]	verb (plural) ⟶ [eat]

In a similar way, further number arguments or other agreements can be 'sent down' the sentence by specifying the appropriate logical arguments.

The third facility, of allowing general conditions, enables new lexical items to be added, not singly, which would be very tedious, but by specifying their shared information (plurality, etc.), and then listing all the words concerned.

Dealing with word groups : With the aid of the powerful tools provided by DCG, quite extensive numbers of syntactical features were identified by our program. The present section describes the ways in which certain codifiable features of English word groups of the highest frequency were implemented.

After dealing with one clause, the program clearly needs to know when to begin its parsing again, that is when a new clause is beginning. We defined an end-of-clause marker to be connectors like *but, although*, etc., any punctuation mark (except apostrophe), or both together. Clearly this heuristic requires a great deal of refinement; but it was found to work in practice in nearly all students' replies.

Let us assume the basic sentence to be defined as a noun phrase (NP) followed by a verb phrase (VP). The NP itself can be composed of different items: either nouns with any number of adjectives and with or without articles or subject pronouns, or proper nouns with or without articles. The rules governing the different possibilities are distinctly messy to express, but the state transition diagram below neatly summarises most of them.

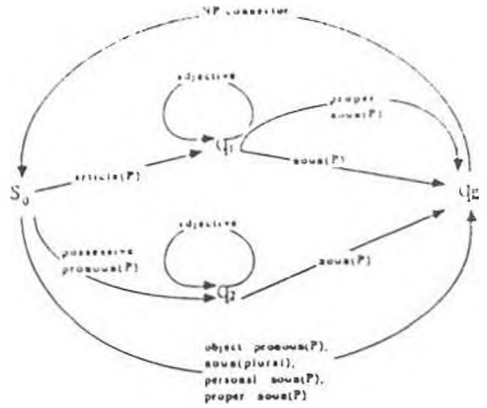


Figure 3

After the state S_0 (beginning of sentence), a possessive pronoun, for instance, will be followed by zero or any number of adjectives, then by a noun, before reaching the end state (q_E). Specifying the order of these adjectives is particularly satisfying. *A small yellow Japanese plastic racing car* is correct, but *A yellow small racing plastic Japanese car* sounds distinctly odd. The order of adjectives that the program checked, then, was: general, colour, origin, material, purpose. Each of the five elements can be recursive within itself; and any or all of them may be omitted except that object pronouns like *me* are allowed in the middle of a clause, and subject pronouns like *I* are in general not.

Of course, the verb itself may not be a single word but of form *might shake*, *had been shaken* or even *might have been being shaken*. The situation is again relatively complex, for one can distinguish four different functions of auxiliaries, making up a total of sixteen different types of basic verb phrases.

Table 1

	VP				
	MODALITY *(Mod)*	PERFECT ASPECT *(Perf)*	PROGRESSIVE ASPECT *(Prog)*	PASSIVE VOICE *(Pass)*	MAIN VERBS *(Mv)*
1					shook
2	might				shake
3		had			shaken
4			was		shaking
5				was	shaken
6	might	have			shaken
7	might		be		shaking
8	might			be	shaken
9		had	been		shaking
10		had		been	shaken
11			was	being	shaken
12	might	have	been		shaking
13	might	have		been	shaken
14	might		be	being	shaken
15		had	been	being	shaken
16	might	have	been	being	shaken

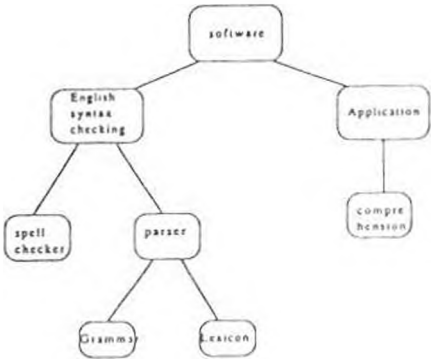
Figure 4

Figure 4 shows that in row 6, for instance, the word *might* can be followed by *have* or *shake*. In general, although the auxiliaries must be in the correct order, they are all optional - except, precisely, when other auxiliaries are present. Thus *might shaken* is incorrect; and although the sixteen types can all be listed (with, of course, *might* replaceable by *will, shall, can,* etc., and with any verb at all replacing *shake*), an algorithm to detect each of these sixteen types and reject all other combinations would be extremely complicated. Instead, certain regularities were observed, such as the fact that *had* and *have* are necessarily followed by *been* or *shaken*. By observing the general form which must follow each of the five functions, highly efficient rules were in fact finally implemented. Another general problem encountered was that of exceptions to rules. Thus, as we have seen, initial NPs and mid-sentence NPs have distinct forms; sentences and

the personal pronoun *I* begin with a capital letter; a changes to an before a vowel; *I* and *you* are singular pronouns but are not followed by 'singular' verbs, and so on. In each case, however, Prolog's flexibility allowed us either to adapt the program state, so that the stage a given sentence had got to could be explicitly indicated, or else to introduce extra arguments and hence make the program branch to the exception codes.

The Finished Program : For the finished product, attention was paid to what might seem merely cosmetic features of the human-computer interface. The simpler and the more pleasant the environment, the more likely the user to consider the interaction positive. A standard IBM-style keyboard was used, and a printer was not in evidence.

After an initial menu for the choice of comprehension passage, the passage chosen is displayed in the top half of the screen, with questions prompted one at a time in the bottom half. After a complete answer has been given by the user, the individual words and the structure are checked; and then a report together with model answers is provided. If mistakes are detected at any stage, the user is given up to two further chances to produce a correct answer. At the end all questions can be answered again if wished. The user can at any moment scroll up or down the passage, invoke the menu or exit the system, these commands being displayed in a separate window, which disappears when no longer necessary. Screen messages are brief and indicate clearly what processing is currently going on. Figures 5 and 6 below presents the overall structure of the program from the user's point of view:



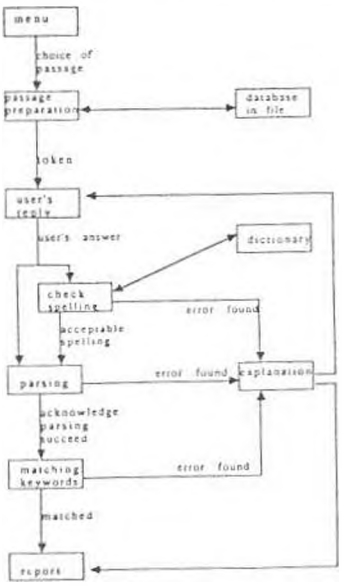


Figure 5 *Figure 6*

Evaluation : We propose the following criteria as a reasonable set for the evaluation of CALL software, even if in practice it is often the 'feel' that is the most important. We also attempt to evalue our program in the light

of these criteria:

functionality, i.e. user interface - the system provides an easy-to-use interface. The user may obtain help or quit at any point, and the commands allowed and their meanings are displayed on a status line. Particular effort is devoted to simplicity of display.
user-worthiness, i.e. degree of testing - the system (which in fact contains a much larger number of types than presented here) was tested with a wide variety of sentences. Users' productions in front of a whirring machine are in fact often highly stereotyped; and in practice, we found, normally fell into categories recognisable by the machine.
help and elucidation - as we have seen, one of the strong points, in terms of both spelling and clause structure, is the explicitness and transparency of many of the strategies used, and hence the ease of their transfer to users. The whole project was designed around the ambitious concept of analysing free input, rather than forcing the interaction into pre-defined grammatical situations. Clearly a further enhancement would be some sort of demonstration mode where the possibilities of the system were demonstrated - which might lead the users to be more adventurous in the structures of their answers.
machine responsiveness - the system response is slowed by the need to search a large dictionary which is not memory-resident. The dialect of Prolog used is not endowed with efficient file handling primitives. An obvious improvement would be to check for common words first using a much smaller dictionary.
augmentation ease - new comprehension passages with appropriate questions are very easily entered as ASCII files, identified by the file extension. Small-scale extensions which might be envisaged include improvement in the treatment of prefixes and suffixes, in the spell-checker speed and the addition of fuzzy-matching of mis-spelled words. (The simplest way is simply to extract all vowels and then compare consonants between the doubtful word and the dictionary.) Having categorically specified many of the grammar rules, the way is clearly open for re-introducing a degree of fuzziness in them, so as to reproduce the flexibility of actual language usage. A useful major extension would involve more reliably identifying ends of clauses. Unusual word order cannot at present be coped with by the system. Homograph disambiguation would not in general present any insuperable difficulties, but would be lengthy to implement.
transferability - the system is currently written in Prolog (Edinburgh syntax) and requires VT100 terminal emulation. The transfer to a big IBM PC will become feasible shortly.
authoring - the use of Prolog allows the development of programs on a higher plane. What is more, DCG allow grammar rules to be written in a succinct and readable form. Our program proved in the end relatively readable. On the other hand, graphics are difficult to implement in Prolog.

Conclusion : The practical detailed implementation of English grammar can never be expected to proceed smoothly. One reason is the sheer quantity of information. Foreign learners and others use analogical processes to

a huge extent, but nearly always with implicit conditions of operation and numerous exceptions and 'sub-exceptions'. The complexity of the problem is often underestimated; two-thousand-page grammars are still very far from providing a complete description.

Nevertheless, our linguistic intuition tells us that nearly all simple sentences can be divided into well-formed and not well-formed, and that an explicit reason can often be found - even if one must attach very considerable scepticism to spontaneous explanations by most native-speakers, despite a commonly-held but erroneous view.

If the rules can be formulated, then they can surely be translated into machine-readable form. The present project attempted to give body to that optimistic leap-in-the-dark. It might be objected that the power of many of the tools proposed was greater than that necessary for analysing simple comprehension replies. This is perhaps the case, but the aim was not to produce a compact, totally robust program for the commercial market; but rather to try out innovative techniques in prototype form, concentrating on real problems that are at present unsolved even by multi-million-ecu endeavours. It was felt that 'you can never have too many horsepower'.

The sticking-point of our sort of approach may reside in its very individuality. The more complex a program, the higher the risk of internal contradiction; but above all, the more difficult for the program to be subsequently added to. There is clearly a huge gap between prototypes in limited contexts and the robustness and reliability required for a successful commercial product.

At present, many of the language-processing packages widely available are at a linguistically impoverished level. This is despite the huge increase of numbers of non-native speakers of English learning the language and, very often, producing written documents in it. We hope that this dual challenge, of meeting the needs of both language learners and 'real' language users, can be tackled simultaneously. We also emit a plea for increased communication between computer scientists with an interest in language matters and language specialists with a knowledge of computing. This must be a viable way forward!