LIMITED DOMAIN SYSTEMS FOR LANGUAGE TEACHING

S G Pulman,
Linguistics, EAS
University of East Anglia,
Norwich NR4 7TJ, UK.

This abstract describes a natural language system which deals usefully with ungrammatical input and describes some actual and potential applications of it in computer aided second language learning. However, this is not the only area in which the principles of the system might be used, and the aim in building it was simply to demonstrate the workability of the general mechanism, and provide a framework for assessing developments of it.

BACKGROUND

The really hard problem in natural language processing, for any purpose, is the role of non-linguistic knowledge in the understanding process. The correct treatment of even the simplest type of non-syntactic phenomena seems to demand a formidable amount of encyclopedic knowledge, and complex inferences therefrom. To date, the only systems which have simulated understanding to any convincing degree have done so by sidestepping this problem and restricting the factual domain in which they operate very severely. In such limited domains semantic or pragmatic processing to the necessary depth can be achieved by brute force, as a last resort. However, such systems are typically difficult to transpórt from one domain to another.

In many contexts this state of affairs is unsatisfactory - something more than fragile, toy, domain dependent systems is required. But there are also situations in which the use of language within a limited factual domain might well be all that was required. Second language learning, especially during the early stages, is one, where quite a lot of the time what is important is practice and training in correct usage of basic grammatical forms, not the conveying of facts about the world. If someone can be taught to use the comparative construction when talking about, say, lions and tigers, he is not likely to encounter much difficulty of a linguistic nature when switching to talk about cars and buses, overdrafts and bank loans, etc., even thought the system he was using might.

Several existing limited domain systems might lend themselves to exploitation for these purposes: one example might be the program described by Isard (1974) which plays a game of noughts and crosses with the user and then engages in a dialogue about the game. Although the domain is tiny the program can deal with much of the modal and tense system of English, as well as some conditionals. Also dealing with noughts and crosses is the program described by Davey (1978), which is capable of (and therefore capable of detecting) correct uses of conjunctions like 'but' and 'although'. Other examples of systems geared to a particular domain and often to a particular syntactic construction will spring readily to mind. Embedded in educationally motivated settings, such systems might well form the basis for programs giving instruction and practice in some of these traditionally tricky parts of English grammar. Such, at any rate, is the philosophy behind the present work. The idea is that there is scope for using limited systems in an area where their limitations do not matter.

ERROR DETECTION AND REPORTING

Of course, such an application carries its own special requirements. By definition, a language learner interacting with such a system is likely to be giving it input which is ill-formed in some way quite often. It is not a feature of most NL systems that they respond usefully in this situation: in a language tuition context, an efficient method for detecting and diagnosing errors is essential.

The problem has of course not gone unnoticed. Hayes and Mouradian (1981), Kwasny and Sondheimer (1981) - among others - have presented techniques for allowing a parser to succeed even with ill-formed or partial input. The ATN based framework of the latter also generates descriptions of the linguistic requirements which have had to be violated in order for the parse to succeed. Such descriptions might well form the basis for a useful interaction between system and learner. However, the work most directly related to that reported here, and an influence on it, is that by Weischedel et al (1978) and Weischedel and Black (1980), (see also Hendrix (1977). They also describe ATN based systems, this time

specifically intended for use in language tutoring programs. The earlier paper describes two techniques for handling errors: encoding likely errors directly into the network, so that the ungrammatical sentences are treated like grammatical ones, except that error messages are printed; and using 'failable' predicates on arcs for such things as errors of agreement. The disadvantages of such a system are obvious: the grammar writer has to predict in advance likely mistakes and allow for them in designing the ATN. Unpredicted errors cannot be handled.

The later paper describes a generalisation of these techniques, with two new features: condition-action pairs on selected states of the ATN for generating reports (1980:100) and the use of a 'longest path' heuristic (101) for deciding between alternative failed parsings. Although impressive in its coverage, Weischedel and Black report two major problems with the system: the difficulty of locating precisely where in a sentence the parser failed, and the difficulty of generating appropriate responses for the user. Those derived from relaxed predicates for the meanings of states were often fairly technical: some helpful examples of usage were given in some cases, but these had to be prestored and indexed by particular lexical items (103).

The problem of accurately locating ungrammaticality is one that is extremely difficult, but arguably made more difficult than it need be by adopting the ATN framework for grammatical description. The ATN formalism is simply too rich: a successful parse in general depends not only on having traversed the network and consumed all the input but on having various registers appropriately filled. Since the registers may be inspected at different points this makes it difficult to provide an algorithmic method of locating ungrammaticality.

The problem of generating error reports and helpful responses for the learner is also made more difficult than it need be if this is conceived of as something extra which needs to be added to a system already capable of dealing with well-formed input. This is because there is a perfectly straightforward sense in which this problem has already been solved if the system contains an adequate grammar. Such a grammar, by explicitly characterising well-formedness, automatically provides an implicit characterisation of how far actual inputs deviate from expected inputs. It also contains all the grammatical information necessary for providing the user with examples of correct usage. These two types of information ought to be sufficient to generate appropriate reports.

THE SYSTEM

The syntactic theory underlying the present system is that of Generalised Phrase Structure Grammar, of the vintage described in Gazdar (1982). This is a more constrained grammatical formalism than that of an ATN, and hence it was possible to develop a relatively simple procedure for almost always accurately locating ungrammaticality, and also for automatically generating error reports of varying degrees of complexity, as well as examples of correct usage. All this is done using no information over and above what is already encoded in the grammar: nothing need be anticipated or pre-stored.

Briefly, on the GPSG theory, the syntactic description of a language consists of two parts: a basic context-free grammar generating simple canonical structures, and a set of metarules, which generate rules for more complex structures from the basic rules. The result of applying the metarules to the basic rules is a large CFG.

The system contains a suite of pre-compilation programs which manipulate a GPSG into the form used by the parser. First, the metarules are applied, producing a large, simple, CFG. The metarule expansion routine is in fact only fully defined for a subset of the metarules permitted by the theory. Roughly speaking, only metarules which do not contain variables which could be instantiated more than one way on any given rule application will be accepted. This is not a theoretically motivated restriction but simply a short cut to enable a straighforward pattern matching production system already available in Pop-11 to be transferred wholesale. A set of filters can be specified for the output by the same means if required.

Next, the resulting CFG is compiled into an equivalent RTN, and finally this RTN is optimised and reduced, using a variant of a standard algorithm for ordinary transition networks (Aho and Ullman (1977:101). The intention behind this extensive preprocessing, apart from increased efficiency, is that the eventual system could be tailored by teachers for their own purposes. All that would be needed is the ability to write GPS grammars, or simple CF grammars, with no knowledge needed of the internal workings of the system.

To give an example of the effect of this pre-processing, the grammar used by the system in the interchanges below contained about 8 rules and 4 metarules. These expand to a simple CFG of about 60 rules; this compiles to an RTN of over 200 states, and the final optimised RTN contains about 40 states.

The parser is a standard RTN parser operating breadth first. The error detection routine is part of the main loop of the parser and works as follows: when no transition can be taken from a particular state in the network, a record is taken of the overall state of the machine. This contains information about how much of the sentence has been successfully parsed, the tree built, a list of states to POP to etc. If this record represents a more successful parse than any record so far it is preserved. This means that at the end of an unsuccessful parse the system has a record of the most successful path pursued, and this record is passed to the error reporting routine.

If desired, all such records could be preserved during a parse and some procedure for choosing between then defined. This would mean that ambiguous parses can be treated independently, whereas at present only one record representing the most successful path through the input on any reading is retained.

The error reporting routine is based around an RTN generator, which simply picks up from the point in the network indicated by the record handed to it, using the information in that record, as well as the RTN, and a special sub-lexicon described below. It is capable of generating error reports of several different types:
(i) it can say what constituent(s) it was trying to complete
(ii) it can say what type of item it was expecting to find at the point of failure - either using the terminology of the grammar, or by example
(iii) it can say what would be necessary to continue the sentence correctly, by generating example continuations.

Here are some transcriptions of typical exchanges with the system using the small grammar mentioned above:

```
:go();
** ready
? william put the book on the shelf
ok
? did william put the book on the shelf
yes
? was the book put on the shelf
yes
? who put the book on the shelf
william
? what did william put on the shelf
the book
? what was put on the shelf
the book
? who was the book put on the shelf by
william
? what did william put
sentence ok up to here:
what did william put ...
expecting to find
one of the following
preposition (in, on, etc)
examples of grammatical continuations
what did william put ...
with something
? what did the read
sentence ok up to here:
what did the ...
expecting to find
one of the following
adjective (nice, big etc.)/ noun (boy, girl etc.)
examples of grammatical continuations
what did the ...
whatdoyoucallit hit
? william hit jumble with a stick big
sentence ok up to here:
william hit jumble with a stick ...
expecting to find
end of sentence
(NB this response is not as helpful as it could
be, since the system does not look at the input
after the point of failure).
? who did was hit
sentence ok up to here:
who did ...
expecting to find
one of the following
noun phrase
examples of grammatical continuations
who did ...
something's thing hit
? who william did hit
sentence ok up to here:
who ...
expecting to find
one of the following
verb1 (did, was, etc.)/ verb2 (hit, read, etc.)
examples of grammatical continuations
who ...
read something
put something with something
```

An attraction of this mechanism, apart from its simplicity, is that it is defined for the whole class of CFGs; this class of grammars is currently believed to be more or less adequate for English and for most of most other languages (Gazdar 1982). The two problems faced by the system of Weischedel and Black seem to have been overcome in a reasonably satisfying way: since after optimisation, the only non-determinism in the RTN is due to genuine ambiguity, we can be sure that the system will, given the way it operates, almost always locate accurately the point of failure in all non-ambiguous cases. And of course, when working with such limited domains we can control for ambiguity to a large extent, and deal with it by brute force if necessary.

However, no such procedure can be wholly learner-proof, (as one of our referees has pointed out). A user might, for example, misspell his intended word and accidentally produce another legitimate word which could fit syntactically. Under these circumstances the parser would proceed unknowingly past the real point of error.

The error reports delivered by the system can be as technical or informal as the grammar writer wants, or simply be prompts and examples of correct usage. In practice, simple one word prompts seem to be as useful as any more elaborated response. As will be clear from the examples, both for prompts and continuations, the system uses a restricted sub-lexicon to minimise the likelihood of generating grammatical nonsense. This sub-lexicon contains vague and general purpose words like 'thing' and 'whatsit'. This apart, no extra work has to be done once the grammar has been written: the system uses only its knowledge of what is grammatical to diagnose and report on what is ungrammatical.

DEVELOPMENTS

The mechanism is currently embedded within two small domains. The one illustrated here is 'told' a simple 'story' and then asks or answers questions about that. The sample grammar was intended to demonstrate the interaction of wh questions with passives, among other things. Although we are not here concerned with the semantics of these domains, they are fairly simple, and several different types of semantic components are used depending on the nature of the domain. For some domains a procedural semantics is appropriate, manipulating objects on

a screen or asking and answering questions about them. In the 'William' program here a production system again based on the Pop-11 matching procedures is used, currently being coupled to a simple backwards chaining inference mechanism.

Neither the grammatical routines nor any embodiment of them constitute a complete tuition system, or anything approaching that: they are merely frameworks for experimentation. But the syntactic error detection routines could be used in many other environments where useful feedback of this type was required, say in database interrogation or machine translation. Within a language tuition context the mechanism could be used to advantage without an associated semantics, in some of the more traditional types of computer aided EFL teaching programs: for example, gap-filling, drill and practice, sentence completion, or grammatical paraphrase tasks. Only trivial adjustments would be needed to the overall mechanism for this to become a powerful and sophisticated framework within which to elaborate such programs.

However, there are several ways in which the general mechanism might be improved upon, most immediately, the following:

(i) if a parse fails early in the sentence, the user only gets a report based on that part of the sentence, when there may be more serious errors later one (or some praiseworthy use of the language). In these cases a secondary parse looking for well-formed sub-constituents, in something like the way a chart parser might do, would provide useful information. (I am grateful to Steve Isard and Henry Thompson for this suggestion).

(ii) the quality of the example continuations could be improved. Eventually it would be desirable to have the generator semantically guided, but this is by no means trivial, even in a limited domain. There are several heuristics which can produce a better type of continuation, however: using a temporary lexicon containing words from the unparsed portion of the sentence, or from the most recently parsed sentences, or combinations of these with the restricted sub-lexicon. In the best cases this type of heuristic can be spectacularly successful, producing a grammatical version of what the user was trying to say. However, they can also flop badly: more testing on real students would be one way of disceovering which of these alternatives is best.

(iii) as suggested in Weischedel and Black, it might be profitable to explore the use of semantic grammars - grammars using semantically rather than syntactically motivated categories - in the system. Although of dubious theoretical status, they are a useful engineering tool: the non-terminals can be labelled in a domain-specific way that is transparent for the user, and, being semantically motivated, the system could appear as if it were doing semantic diagnosis of a limited type as well as syntactic diagnosis. For example, instead of being prompted for an adjective, the user might be prompted for 'a word describing the appearance of a car', or something equally specific. Furthermore, the availability of the pre-compilation programs means that it should be possible to use the metarule formalism for these grammars also: this should go some way towards minimising their linguistic disadvantages, namely, a tendency to repetition and redundancy in expressing facts about the languages they generate.

The system is written in Pop-11 (a Lisp-like language) within the POPLOG programming environment developed by the University of Sussex. At UEA POPLOG runs on a VAX 11/780 under VMS.

REFERENCES

Aho, A. and Ullman, J. (1977) Principles of Compiler Design,
London, Addison Wesley Publishing Co.

Davey, A. (1978) Discourse Production
Edinburgh University Press

Gazdar, G. (1982) Phrase Structure Grammar
in P. Jacobson and G.K. Pullum (eds) The Nature of Syntactic Representation, Dordrecht: D.Reidel Publishing.

Hayes, P.J., and Mouradian, G.V. (1981) Flexible Parsing
AJCL 7, 232-242

Hendrix, G. (1977) Human Engineering for Applied NL Processing
IJCAI 5, Cambridge MA.

Isard, S.D. (1974) What would you have done if...?
Theoretical Linguistics 1, No 3.

Kwasny, S. and Sondheimer, N. (1981) Relaxation Techniques for Parsing Ill-formed Input
AJCL 7, 99-108

Weischedel, R. et al. (1978) An Artificial Intelligence Approach to Language Instruction
Artificial Intelligence 10,3

Weischedel R. and Black, J. (1980) Responding Intelligently to Unparsable Inputs
AJCL 6, 97-109