Effective Parsing With Generalised Phrase Structure Grammar

Allan Ramsay

Cognitive Studies Program, University of Sussex
Brighton, BN1 9QN, England

Abstract

Generalised phrase structure grammars (GPSG's) appear to offer a means by which the syntactic properties of natural languages may be very concisely described. The main reason for this is that the GPSG framework allows you to state a variety of meta-grammatical rules which generate new rules from old ones, so that you can specify rules with a wide variety of realisations via a very small number of explicit statements. Unfortunately, trying to analyse a piece of text in terms of such rules is a very awkward task, as even a small set of GPSG statements will generate a large number of underlying rules.

This paper discusses some of the difficulties of parsing with GPSG's, and presents a fairly straightforward bottom-up parser for them. This parser is, in itself, no more than adequate - all its components are implemented quite efficiently, but there is nothing tremendously clever about how it searches the space of possible rules to find an analysis of the text it is working on. Its power comes from the fact that it learns from experience: not new rules, but how to recognise realisations of complex combinations of its existing rules. The improvement in the system's performance after even a few trials is dramatic. This is brought about by a mechanism for recording the analysis of text fragments. Such recordings may be used very effectively to guide the subsequent analysis of similar pieces of text. Given such guidance it becomes possible to deal even with text containing unknown or ambiguous words with very little search.

1. Generalised Phrase Structure Grammar

There has been considerable interest recently in a grammatical framework known as "generalised phrase structure grammar" (GPSG). This framework extends the expressive power of simple context free grammars (CFG's) in a number of ways which enable complex systems of regularities and restrictions to be stated very easily. Advocates of GPSG claim that it enables concise statements of general rules; and that it provides precise descriptions of the syntactic properties of strings of lexical items. For the purpose of this paper I shall assume without further discussion that these claims are true enough for GPSG's to be considered interesting and potentially useful. The problem is that straightforward parsing algorithms for GPSG's can take a long time to run - the CFG which you get by expanding out all the rules of a moderately complex GPSG is so enormous that finding a set of rules which fits a given input string is a very time-consuming task. The aim of this paper is to show how some of that time may be saved.

The GPSG framework has been described in detail in a number of other places. The discussion in this paper follows Gazdar and Pullum [Gazdar & Pullum], [Gazdar et al.], though as these authors point out a number of the ideas they present have been discussed by other people as well. For readers who are entirely unfamiliar with GPSG I shall briefly outline enough of its most salient features to make the remainder of the paper comprehensible - other readers should skip to the next section.

GPSG starts by taking simple CF rules and noting that they carry two sorts of information. The CF rule

(1) S --> NP VP

says that whenever you have the symbol S you may rewrite it as NP VP, i.e. as the set NP, VP with NP written before the VP. GPSG separates out these facets of the rule, so that a grammar consisting of the single CF rule given above would be written as

(2a) S --> NP, VP
(2b) NP << VP

i.e. as an "immediate dominance" (ID) rule, saying that the set of symbols {S} may be replaced by the set of symbols NP, VP and a "linear precedence" (LP) rule which says that in any application of any ID rule involving a NP and a VP, the NP must precede the VP. There is some doubt as to whether they should be tied to specific groups of ID rules. It makes little difference to the algorithms outlined here one way or the other - for simplicity of exposition it will be assumed that LP rules are universal.

In the trivial case cited here, the switch from a CFG to ID/LP format has increased the number of rules required, but in more complicated cases it generally decreases the number of statements needed in order to specify the grammar.

ID/LP format allows you to specify large sets of CF rules in a few statements. GPSG provides two further ways of extending the sets of CF rules in your grammar. The first is to allow the elements of a rule to be complex sets of feature/value pairs, rather than just allowing atomic symbols. The rhs of rule 2a, for instance, refers to items which contain the feature/value pairs [category NP] and [category VP] respectively, with no explicit reference to other features or their expected values (though there will generally be a number of implicit restrictions on these, derived from the specification of the features in the grammar and their interactions). Thus 2a in fact specifies a whole family of CF ID rules, namely the set {all possible combinations of feature/value pairs which include [category NP]} X {all possible combinations of feature/value pairs which include [category VP]}. In theory this set could be expanded out, but it is not a tempting prospect - it would simply take a lot of effort, waste a lot of space, and lose the generalisation captured by 2a.

The other way of extending the grammar is to include metarules, i.e. rules which say that if you have a rule that matches a given pattern, you should also have another, derived, rule. For instance, the metarule

(3) VP $\rightarrow$ ..., NP $\Rightarrow$
VP [passive] $\rightarrow$ ..., PP[by]

says that for any rule stating that a VP may be made up of some set of items including a NP (the ... means any, possible empty, set of items), you should have a rule which states that a passive VP may be made up of the same set of items but with the NP replaced by a PP of type "by". Metarules are applied until they close, i.e. whenever a metarule is applied and produces a new rule, the entire set of metarules is scanned to see if any of them can be applied to this new rule.

There are two further points about GPSG which are worth noting before we move on to see how to parse using the vast set of rules induced by a set of ID, LP and meta rules. Firstly, it is customary to include in the feature set of each lexical item a list containing the names of all the ID rules in which that item may take part. This induces a finer classification of lexical items than the one implied by the simple division into categories such as verb, noun, preposition, ... (this classification is often referred to as "lexical subcategorisation", i.e. splitting lexical items into subsets of the usual categories). Secondly, the inheritance of features when several items are combined to make a single more complex structure is governed by two rules, the "head feature convention" (HFC) and the "foot feature principle" (FFP). Very briefly: features are divided into "head features" and "foot features". The HFC says that head features are inherited from the "head", i.e. that sub-structure which has the same basic category (verb, noun, ...) as the complex structure and which is of lowest degree out of all the substructures of this type. The FFP says that foot features are inherited by studying all the other, non-head, substructures and copying those foot features on which they do not disagree (i.e. they need not all include a value for each foot feature, but a foot feature will not be copied if there are items which include different values for it).

The foregoing is very far from being a complete description of the GPSG framework. It should be detailed enough to give an idea of how rules are stated within the framework; and it should be detailed enough to make the rest of the paper comprehensible.

2. Parsing With GPSG's

Parsing with a GPSG is essentially the same as parsing with any of the other common grammatical systems. Given a string of lexical items, find some sequence of rules from the grammar which will combine items from the string together so that all that remains is a single structure, labelled with the start symbol of the grammar and covering the whole of the original text. The same decisions have to be made when designing a parser for GPSG as for the design of any parser for a grammar specified as a set of rewrite rules (this includes ATN's) - top down : bottom up, left - right : island building, depth first : breadth first : pseudo parallel. With GPSG there is yet another question to be answered before you can start to put your parser together: how far should the rule set be expanded when the rules are read in?

There are two extreme positions on this. (i) You could leave the rules in the form in which they were stated, i.e. as a collection of ID rules, plus a set of metarules which will generate new rules from the base set, plus a set of LP rules which restrict the order in which constituents of the rhs of a rule may appear. (ii) You could expand out the entire set of CF rules, first comparing the ID rules with the metarules and constructing new ID rules as appropriate until no new rules were generated; then generating all the ordered permutations of rhs's allowed by the LP rules; and finally expanding the specified feature sets which make up each constituent of a rule in all possible ways.

Neither of these options is attractive. As Thompson pointed out, (i) is untenable, since metarules can alter rules by adding or deleting arbitrary elements [Thompson 82]. This means that if you were working top down, you would not even know how the start symbol might be rewritten without considering all the metarules that might expand the basic ID rules which rewrite it; working bottom up would be no better, since you would always have to worry about basic ID rules which might be altered so they covered the case you were looking at. At every stage, whether you are working down from the top or up from the bottom, the rule you want may be one that is introduced by a metarule; you have no way of knowing, and no easy way of selecting potentially relevant basic rules and metarules.

On the other hand, expanding the grammar right out to the underlying CF rules, as in (ii), looks as though it will introduce very large numbers of rules which are only trivially distinct. It may

conceivably be easier to parse with families of fully instantiated rules than with rule schemas with underdetermined feature sets, e.g. with

(4a) S --> NP [num = sing], VP [num = sing]
(4b) S --> NP [num = plural], VP [num = plural]

rather than

(4c) S --> NP [num = NUM], VP [num = NUM]

However, complete expansion of this sort will definitely require orders of magnitude more space - one simple item such as NP could easily require 10 - 15 other features to be specified before it was fully instantiated. The combinatorial potential of trying to find all compatible sets of values for these features for each item in a rule, and then all compatible combinations of these sets, is considerable. It is unlikely that the possible gains in speed of parsing will be worth the cost of constructing all these combinations a priori.

To a large extent, then, the choice of how far to expand the grammar when the rules are first read is forced. We must expand the metarules as far as we can; we would rather not expand underdetermined feature sets into collections of fully determined ones. The remaining question is, should we leave the rules which result from metarule application in ID/LP format, or should we expand them into sets of CF rules where the order in which items occur on the rhs of the rule specifies the order they are to appear in the parse? For top down analysis, it is likely that CF rules should be generated immediately from the ID/LP basis, since otherwise they will inevitably be generated every time the potential expansions of a node are required. For bottom up analysis the question is rather more open. It is, at the very least, worth keeping an index which links item descriptions to rules for which the items are potential initial constituents; this index should clearly be pruned to ensure that nothing is entered as a potential initial constituent if the LP rules say that it cannot be.

We can summarise our discussion of how to parse using GPSG's as follows. (i) Metarules should be expanded out into sets of ID rules as soon as the grammar is read in. (ii) It may also be worth expanding ID rules into sets of rules where the order of the rhs is significant. (iii) It is not a good idea to expand ID rules into families of CF rules with all legal combinations of feature: value pairs made explicit. We also note that if we are simply going to treat the rules as ways of describing constituent structure then some sort of chart parser is likely to be the most appropriate mechanism for finding out how these rules describe the input text [Shieber 84].

These are all reasonable decisions. However, once we come to work with non-trivial GPSG grammars, it appears that general purpose parsing algorithms, even efficient ones, do rather a lot of work. We need some way of converting the declarative knowledge embodied in the rules of the grammar into procedural knowledge about how to analyse text. The approach described in this paper involves using two parsing algorithm together. We have a standard bottom-up chart parser, which simply tries out grammatical rules as best it can until it arrives at some combination which fits the text it is working on; and a "direct recogniser", which uses patterns of words which have previously been analysed by the chart parser to suggest analyses directly.

There is not much to say about the chart parser. It uses the rules of the grammar in a form where the metarules have been applied, but the permutations implied by the LP rules have not been explicitly expanded. This means that we have fewer rules to worry about, but silghtly more work to do each time we apply one (since we have to check that we are applying it in a way allowed by the LP rules). The extra work is minimised by using the LP rules, at the time when the grammar is first read in, to index ID rules by their possible legal initial substructures. This prevents the parser trying out completely pointless rules.

It is hard to see many ways in which this parser, considered as a general purpose grammar applying algorithm, could be improved. And yet it is nowhere near good enough. With a grammar consisting of about 120 rule schemas (which expands to about 300 schemas by the time the metarules have been applied), it takes several thousand rule applications to analyse a sentence like "I want to see you doing it". This is clearly unsatisfactory.

To deal with this, we keep a record of text fragments that we have previously managed to analyse. When we make an entry in this record, we abstract away from the text the details of exactly which words were present. What we want is a general description of them in terms of their lexical categories, features such as transitivity, and endings (e.g. "-ing" or "-ed"). These abstracted word strings are akin to entries in Becker's "phrasal lexicon" [Becker 75]. Alongside each of them we keep an abstracted version of the structure that was found, i.e. of the parse tree that was constructed to represent the way we did the analysis. Again the abstraction is produced by throwing away the details of the actual words that were present, replacing them this time by indicators saying where in the original text they appeared.

It is clearly very easy to compare such an abstracted text string with a piece of text, and to instantiate the associated structure if they are found to match. However, even if we throw away the details of the particular words that were present in the original text, we are likely to find that we have so many of these string: structure pairs that it will take us just as long to do all the required comparisons as it would have done to use the basic chart parser with the original set of rules.

To prevent this happening, we condense our set of recognised strings by merging strings with common initial sequence, e.g. if we have two recognised fragments like

(3) det, adj, adj, noun ---→ NP(det = [1], adjlist = [2 3], n = [4])
(4) det, adj, noun --------→ NP(det = [1], adjlist = [2], n = [3])

we take advantage of their shared structure to store them away like

adj, noun ---→ NP(det = [1], adjlist = [2 3], n = [4])
(5) det, adj, noun --------→ NP(det = [1], adjlist = [2], n = [3])

Merging our recognised fragments into a network like this means that if we have lexically unambiguous text we can find the longest known fragment starting at any point in the text with very little effort indeed - we simply follow the path through the network dictated by the categories (and other features, which have been left out of (3), (4) and (5) for simplicity) of the successive words in the text.

This "direct recognition" algorithm provides extremely rapid analyses of text which matches previously analysed input. It is not, however, "complete" - it is a mechanism for rapid recognition of previously encountered expansions of rules from the grammar, and it will not work if what we have is something which is legal according to the grammar but which the system has not previously encountered. The chart parser is complete in this sense. If the input string has a legal analysis then the chart parser will - eventually - produce it.

For this reason we need to integrate the two mechanisms. This is a surprisingly intricate task, largely because the chart parser assumes that all rules which include completed substructures are initiated together, even if some of them are not followed up immediately. This assumption breaks down if we use our direct recogniser, since complete structures will be entered into the chart without their components ever being explicitly added. It is essential to be very careful integrating the two systems if we want to benefit from the speed of the direct recogniser without losing the completeness of the chart parser. Our current solution is to start by running the direct recognition algorithm across the text, repeatedly taking the longest recognised substring, adding all its known analyses to the chart, and then continuing from the position immediately following this string. If we do not recognise anything at a particular point, we simply make an entry in the chart for the current word and move on. When we have done this there will be a number of complete edges in the chart, put there by the direct recogniser, and a number of potential combinations to follow up. At this point we allow normal chart parsing to take place, hoping that the recognised structures will turn out to be constituents of the final analysis. If they are not, we have to go back and successively add single word edges wherever we jumped in with a guess about what was there.

3. Ambiguous And Unknown Words

The combination of chart parser and direct recogniser is sufficiently effective that we can afford to use it on text that contains ambiguous words without worrying about the extra work these will entail. This is fortunate, given the number of words in English which are ambiguous as to lexical category - "chart", "direct", "can", "use", "work" and "entail" from the first sentence of this paragraph alone!

Lexical ambiguity generally causes problems for bottom-up parsers because each interpretation of a given word will tend to indicate the presence of a different type of structure. It will often turn out that when all the possibilities have been explored only one of the interpretations actually contributed to a complete, consistent parse, but it may take some time to check them all. By looking for structures cued by strings of words we get a strong indication of which is the most promising interpretation - interpretations which are not going to be part of the final analysis are not likely to appear inside substantial recognised strings. To take a simple example, consider the two sentences "I don't see the use" and "I will use it". In the first the interpretation of "use" as a noun fits easily into wider patterns of the sort we will have stored away, such as {det, noun} → NP or {verb, det, noun} → VP, whereas its interpretation as a verb does not. In the second the interpretation as a verb fits into plausible patterns like aux, verb → VSEQ or {aux, verb, pronoun} → VP, while the interpretation as a singular noun does not seem to fit well into any surrounding patterns.

These cues are effective enough for us to be able to follow [Thorne et al. 68] in merging the "open" lexical categories, i.e. noun, verb, adj and adv. In the vast majority of cases, the final analysis of the text will tell us which of the various subclasses of the category "open" a particular instance of a given word must have belonged to. We do, of course, make heavy use of the connections between these categories and the suffix system - if a word has had "-ing" added to it, for instance, then it must be functioning as a verbal form. Not only does the final analysis usually determine uniquely the interpretation for each open category word in the input, the combined recogniser and parser produce this final analysis with comparatively little search. We are thus able to deal with input that contains ambiguous words just about as effectively as with input that doesn't. The disambiguation is performed largely by having the system recognise that it has never seen, say, an open category word functioning as a verb surrounded by the current local configuration of words, whereas it has seen something in this context which was eventually interpreted as a noun. This has the added advantage of enabling us to produce a syntactic analysis of text containing previously unknown words - they are immediately assigned to the open category, and their particular function in the current context is discovered at the end of the analysis. How you construct a meaning representation from

such an analysis is another matter.

5. Conclusions

The parser and rule learner described above perform far far better than the parser by itself - on complex cases, the parser may find the correct analysis several hundred times as quickly using learnt rules as it would have done with just the basic set. Experience with the system to date indicates that the introduction of new rules does not slow down the process of selecting relevant rules all that much, partly because the indexing of patterns against initial elements cuts out quite a lot of potentially pointless searching. It is conceivable that when the system has been run on large numbers of examples, the gains introduced by abstracting over long, unusual strings will be outweighed by the extra effort involved in testing for them when they are not relevant. If so, it may be a good idea to put a limit on the length of string for which compound rules should be recorded. There is no indication as yet that this will be necessary.

It is of interest that the compound rules the system creates are akin to the productions used in Marcus' deterministic parser [Marcus] - patterns of descriptions of items which the parser is prepared to react to, combined with packets of simple actions to be taken when a pattern is recognised. There is no suggestion here that the system described above could ever be fully deterministic - there are just too many possibilities to be explored for this to be likely - but it certainly explores fewer dead ends with learnt compound rules than with the initial basic ones.

Acknowledgments

My understanding of GPSG owes a great deal to discussions with Roger Evans and Gerald Gazdar. The idea of using recognisable sequences of categories to find shortcuts in the analysis arose partly out of conversations some time ago with Aaron Sloman. Gerald Gazdar and Steve Isard read and commented on this paper and an earlier, even more misguided one. Steve Isard implemented the basic chart parser which was adapted for the work reported here. Any remaining errors, etc. are as usual the author's responsibility.

References

Becker, The Phrasal Lexicon. TINLAP, 1975.

Gazdar, G. Klein, E., Pullum, G.K., Sag, I.A., Generalised Phrase Structure Grammar. Blackwell, Oxford (in press - 1985).

Marcus, M., A Theory of Natural Language Processing PhD thesis, MIT, 1980.

Shieber, S.M., Direct Parsing of ID/LP Grammars Linguistics & Philosophy 7/2, 1984.

Thorne, J.P., Bratley, P. & Dewar, H., The Syntactic Analysis of English By Machine in Machine Intelligence 3, ed. Michie, Edinburgh UP, 1968.

Thomson, H. Handling Metarules In A Parser For GPSG DAIRP 175, University of Edinburgh, 1982.