Parsing Using the Role and Reference Grammar Paradigm

Elizabeth Guest
Innovation North, Leeds Metropolitan University
Leeds, LS6 3QS, UK

ABSTRACT

Much effort has been put into finding ways of parsing natural language. Role and Reference Grammar (RRG) is a linguistic paradigm that has credibility in linguistic circles. In this paper we give a brief overview of RRG and show how this can be implemented into a standard rule-based parser. We used the chart parser to test the concept on sentences from student work. We present results that show the potential role of this method for parsing ungrammatical sentences.

Keywords: Role and Reference Grammar, Parsing, Templates, Variable word order flexibility, Natural language processing

1. INTRODUCTION

Role and Reference Grammar (RRG) [7][8] is a promising theory for extracting the meaning from sentences from a computational viewpoint. It posits multiple projections where various aspects of a sentence can be dealt with separately. For example, words that modify other words are removed from the constituent projection and placed in an operator projection. As a result, only the main constituents of a sentence have to be parsed, simplifying the parsing process. RRG has a strong link with semantics, and the grammatical constructs are designed both to be cross-linguistically valid and to make the meaning relatively easy to extract. The grammatical constructs are based on templates rather than rules. This means that more information can be encoded into the grammatical construct, which in turn makes the meaning easier to extract.

However, there are aspects of RRG that make it harder to implement. It is much harder to parse with templates than with rules; RRG templates are particularly hard because lines are allowed to cross and the parse trees are not simply made up of parents and children, but nodes can have modifiers (such as PERIPHERY) attached to them. In addition, although RRG says nothing explicit about word order constraints, they are implicit in the templates in that the theory contains examples from many languages that include fixed and free word order, and varieties in between.
We describe how modifications can be added to the chart parsing algorithm to extend its functionality to variable word order flexibility and templates. These extensions should be applicable to any rule based parsing algorithm, thus making parsing according to this paradigm feasible. We believe that this would make RRG a better alternative to HPSG [9] [5] [6] and Dependency Grammar [2][4][1] which are currently the most popular parsing paradigms.

RRG posits algorithms to go from syntax to semantics, and semantics to syntax. The main contribution is the use of parsing templates and the notion of the CORE. A CORE consists of a predicate (generally a verb) and (normally) a number of arguments. It must have a predicate. Everything else is built around one or more COREs. Simple sentences contain a single CORE; complex sentences contain several COREs.

The fact that RRG focuses on COREs, means that the semantics is relatively easy to extract from a parse tree. You just have to look for the PRED and ARG branches of the CORE to obtain the predicate (PRED) and the arguments (ARG). Who did what to whom will depend either on the ordering of the ARG branches (in the case of English), or on their cases, or both.

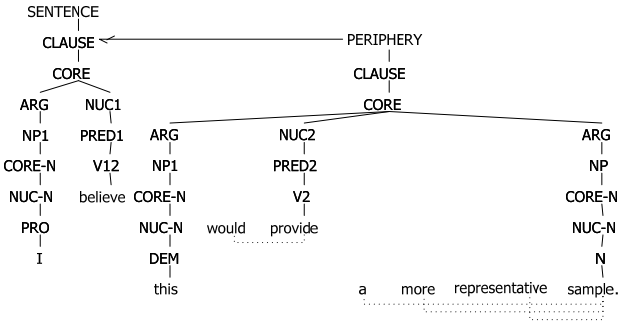


Figure 1: Example RRG parse tree
This diagram shows the *constituent* projection. Words that belong to the *operator* projection are linked to the words on which they operate.

RRG makes extensive use of templates. These templates consist of whole trees and are thus harder to use in a parsing algorithm than rules. The templates can easily be reduced to rules, but only at a loss of much important information. The example in Figure 1 consists of one large template that gives the overall structure and some simple templates (which are equivalent to rules) so that elements such as NP and PP can be expanded. An NP is a noun phrase and in this theory consists of a noun, pronoun or question word. Templates are required to parse complex noun phrases, such as those with embedded clauses. A PP is a prepositional phrase and consists of a preposition followed by a NP. Clearly, if we reduce the large template in the example in Figure 1 to the rule CLAUSE → NP V1 DEM V2 NP, a lot of the information inherent in the structure of the template is lost. A further feature of RRG is that the branches of

the templates do not have to have a fixed order and lines are allowed to cross. The latter is important for languages such as German and Dutch, where the adverb that makes up the periphery normally occurs within the core.

In English, there is a strict word order to the constituents of the CORE: subject, verb, object. In questions, the object comes first if this is the subject of the question. For example, if we have the sentence "John ate pizza", we can ask the questions: "What did John eat?" and "Who ate pizza?" In the latter question, there is no change within the CORE. "Who" is the subject and we just need to fill in that slot. In the first question, however, "what" appears at the start of the sentence. As in English, this is the object and would normally appear at the end of the sentence; RRG theory says that this is outside the CORE. Since it is still really part of the CORE it goes in a 'pre-core slot' (PrCS). The PrCS simply tells us that something that is normally found within the CORE has been moved outside. This is useful information because we can then investigate why it has been moved outside and discover, for example, that we have a question.

The other RRG constituent that is important for English is the 'left detached position' (LDP). We can say "John ate pizza yesterday" and "Yesterday, John ate pizza". In RRG theory, the first sentence gives us the canonical form and the fact that the position of "yesterday" has changed in the second sentence is signalled by putting it in a left detached position. This is useful for working out the emphasis of a sentence. Note that "yesterday" is not considered to be part of the CORE, but is peripheral information. It therefore goes in a PERIPHERY.

The concepts of PrCS and left detached position (LDP) are illustrated in Figure 2. Other languages (such as Japanese) have Post Core Slots and Right Detached positions.

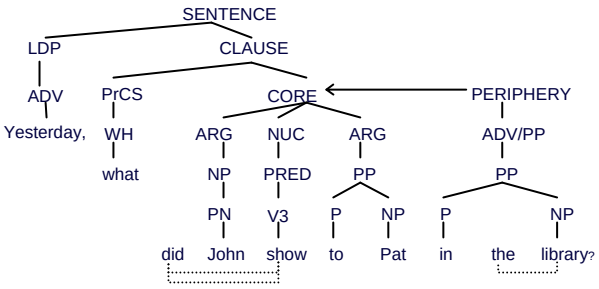


Figure 2: RRG tree showing the use of the left detached position (LDP), pre-core slot (PrCS) and PERIPHERY

2. Methods

2.1 Outline of the parsing algorithm:

Tagging is an important part of parsing, for this work tagging has been done semi-automatically using "toolbox", a program available from SIL. This was so that it was possible to experiment with the tags. However, once the tags have been finalised an appropriate automatic tagger can be used, or written using standard techniques. The main things to bear in mind when designing a tagging schema for RRG are:

1) It should be easy to separate the operators from the constituents.
2) It should be easy to distinguish between different classes of operators.
3) Words denoting discourse features is words that link sentences together, need to be handled in a sensible way as these do not feature in the standard RRG description.

As long as these conditions are met any tagging schema would work with RRG. Once the text has been tagged, there are three parts to the parsing algorithm:

1) Strip the operators. This part removes all words that modify other words. It is based on a correct tagging of head and modifying words. This stage uses methods from dependency grammar and the end result is a simplified sentence.
2) Parse the simplified sentence using templates. This is done by collapsing the templates to rules, parsing using a chart parser and then rebuilding the trees at the end using a complex manipulation of pointers. The chart parser has been modified to handle varying degrees of word order flexibility. This is done by working out all the possible combinations of the ordering using breadth first search. These options are then built into a complex data structure in such a way that relevant parts are deleted as parsing progresses, leaving the correct option according to the data.
3) Draw the resulting parse tree.

2.2 Parsing templates

The reason for parsing with templates rather than rules is that templates contain a lot more information. In addition, RRG contains peripheries and links that do not fit into trees in the normal way but via arrows, as illustrated in Figure 3, which shows an automatically generated parse tree. Also, by using a template, it is easier to ensure that in sentences with a pre-core slot (PrCS), for example, an argument really is missing from the CORE. However, parsing with templates is much harder than with rules.

Templates are parsed by collapsing all the templates to rules and then rebuilding the correct parse tree once parsing is complete. This is done by including the template tree in the rule, as well as the left- and right-hand sides. When rules are combined during parsing, we make sure that the right-hand side elements of the instantiated rule, as represented in the partial parse tree, point to the leaves of the appropriate rule template tree. This is especially important when the order of the leaves of the template may have been changed. The reference number for the rule that has been applied is also recorded so that it can be found quickly.

Modifying nodes, such as PERIPHERY, cause problems with rebuilding the tree. This is because such nodes can occur anywhere within the template, including at the root and leaf levels. Also, if we are dealing with a sub-rule whose root node in the parse tree has a modifying node, it is not possible to tell whether this is a hangover from the previous template, or part of the new template. To solve this problem, modifying nodes have flags to say whether they have been considered or not. There is a potential additional problem with repeated nested rules: if processing is done in the wrong order, the pointers to the rule template tree get scrambled. To overcome this problem, each leaf of a template is dealt with before considering sub-rules.
The algorithm for building the tree is:

1) Get the appropriate rule and rule template tree.
2) If the rule tree is of depth 1 and has no embedded modifying nodes (that is modifying nodes that point to a

node other than the root), then simply continue by looking at each of the children in turn, starting at step 1.

3) If the rule tree is of depth greater than 1 or there are embedded modifying nodes, then make the rule template tree point to the appropriate places in the parse tree. This is done using the links made from the parse tree to the rule template tree during parsing. Note that the parse tree will consist of simple rule structures of depth 1 and modifying nodes will show up as children.
4) Clear all the children in the parse tree. This will have the effect of removing any embedded modifying nodes.
5) Copy all the children of the template tree and copy into the appropriate place in the parse tree.
6) If the template has modifying nodes, copy that part of the template tree and insert into the appropriate place in the parse tree.
7) Replace the leaves of the copied template trees with the original leaves. This is possible because the template leaves are pointing to the original leaves (step 3).
8) Consider each leaf in turn, modifying the parse tree as above (start at step 1 for each leaf).

2.3 Parsing with fixed, free and constrained word order

There were two main problems to solve in order to modify the chart parser to handle varying degrees of word order flexibility:

- Working out a notation for denoting how the word order can be modified
- Working out a method of parsing using this notation.

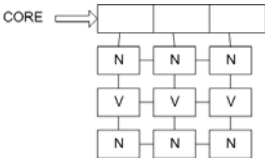
The first was achieved by the following notation on the ordering of the leaves of the template, treating the template as a rule:

- Fixed word order: leave as it is [N V N].
- Free word order: insert commas between each element [N, V, N]. (Note that case information is included as an operator so that the undergoer and actor can be identified once parsing is complete.)
- An element has to appear in a fixed position: use angular brackets: [N, <V>, ADV] this means that N and ADV can occur before or after v, but that V *must* occur in second position. Note that this is second position counting constituents, not words.

Other kinds of variation can be obtained via bracketing. So, for example, [(N, V) CONJ (N, V)] means that the N's and V's can change order, but that the CONJ must come between each group. If we had [(N,V),CONJ,(N,V)] Then the N's and V's must occur next to each other, but the group do not have to be separated by the CONJ, which h can occur at the start, in the middle or at the end, but which cannot break up an [N,V] group.

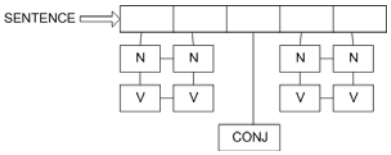
2.4 Modifications to the parsing algorithm

Parsing was achieved via a structure that encoded all the possible orderings of a rule. So, for example, the rule CORE→N, V, N would become:



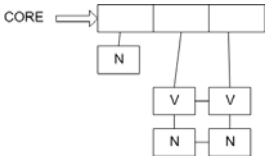
This means that N or V can occur in any position, and N has to occur twice. The lines between the boxes enable the "rule" to be updated as elements are found.

Using this schema, SENTENCE→(N,V) CONJ (N,V) would become:

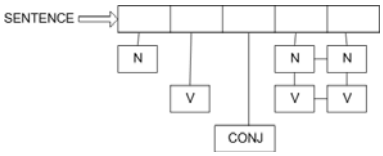


In this case, the CONJ in the middle is by itself because it has to occur in this position as the grouping word order is fixed. The groupings of N's and V's show where the free word ordering can occur.

To apply a rule, the first column of the left-hand side of the rule is searched for the token. Any tokens that do not match are deleted along with the path that leads from them. In the first example, after an N is found, we would be left with:



In the second example, after an N is found we would be left with:



Note that in order for the rule to be satisfied, we *must* find a V and then a CONJ: there are no options for position 2 once the element for position 1 has been established.

In this way, we can keep track of which elements of a rule have been found and which are still to be found. Changes in ordering with respect to the template are catered for by making sure that all instantiated rules point back to the appropriate leaves of the rule template, as described above.

The different possibilities for each rule are obtained via a breadth first search method that treats tokens in brackets as blocks. Then the problem becomes one of working out the number of ways that blocks of different sizes will fit into the number of slots in the rule.

3. Results

The above algorithm was tested on student texts that formed part of a statistics assignment. The task was to identify biases in a certain scenario and to suggest a less biased way of collecting data. There are over 100 of these texts, but as tagging is fairly time consuming, to date only 15 of these texts have been considered. Some texts were discarded because they had some formatting in them, such as bulleted lists. Of those texts we could use, we used the first 45 sentences (six complete texts) to design a set of templates to parse these sentences. Note that this is the first attempt to use RRG theory to parse texts. We managed to parse all but one sentence, and this sentence has

ungrammaticality in it, which is hard to work around. We then applied these templates to the next six texts (47 sentences) to see how well our current set-up works. Of these sentences, 15 were parsed correctly. This is about 32%, or 36% if you discount the ungrammatical sentences that did not parse. Many of the sentences that did not parse did not do so because their structure had not been modelled. This is not a surprise given that the templates had only been determined using six texts.

In addition, the lengths of sentences vary from seven words to 63 words, and the distribution can be seen in the graph in Figure 5, which shows the frequency of the number of words in the sentences. This shows that most of the sentences are long and thus have a high degree of complexity.

Although, the preliminary results of applying these algorithms to student texts are very promising, some issues have been highlighted. The method parses relatively simple sentences correctly, and the main arguments and verbs are found. In addition, some very long and complicated sentences are parsed correctly, and many kinds of grammatical errors do not cause any problems.

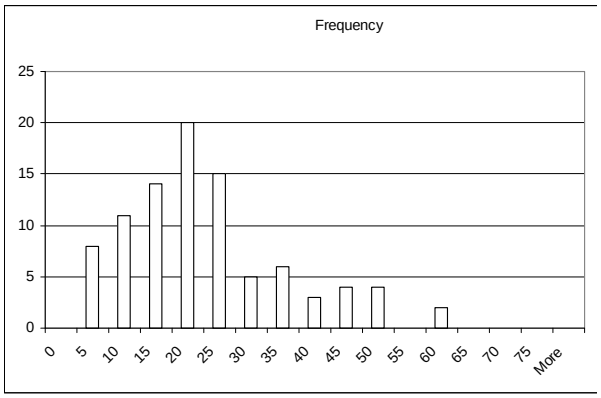


Figure 3: Graph showing the frequency of sentence lengths in the analysed texts. Sentence lengths are along the x-axis.

An example of a correctly parsed sentence is: "I would target main areas populated by students and would attend the same place at different times and during the day." The parse tree for this example is given in Figure 6. Note that the complex object "main areas populated by students" has been parsed correctly and that the tree attaches the qualifying phrase to "area" so that it is clear what is being qualified. An important source of ambiguity in English sentences is caused by prepositional phrases, and this is a main cause of multiple parses of a sentence. In this example, the phrases "at different times" and "during the day" are placed together in the periphery of the CORE, although arguably they should have a different structure. This is a design decision to limit the number of parses. This kind of information needs semantic information to sort out what attaches to what. This cannot be obtained purely from the syntax.

An example of an ungrammatical sentence that is correctly parsed is: "Results from the observations would be less bias if the sample again was not limit the students in the labs between 9:30 and 10:30 on a Thursday morning." (The parse tree for which is given in Figure 7.) This sentence parses correctly because the affix that should be on "limit" is an operator and the correctness of the operators is not checked during the parsing process. The word "bias" is labelled as a noun and gets attached as the second argument to "would be", although it should be "biased", which would get it labelled as an adjective. Despite these errors, the meaning of the sentence is clear and the parse will enable the meaning to be deduced.

The following sentence produces two parses (one correct and one incorrect): "Therefore, asking only the students present on a Thursday morning will exclude all the students that either have no lessons or are not present." The incorrect parse breaks up "Thursday morning" to give two clauses: (1) "Asking only students present on a Thursday" and (2) "Morning will exclude all the students that either have no lessons or are not present."

In the first clause, the subject is "asking only students", the main verb is "present" and the object is "on a Thursday morning". This does not make sense, but it is syntactically correct as far as the main constituents are concerned. Similarly, the second clause is also syntactically correct, although it does not make sense. There are two ways of eliminating this parse. The first is to do a semantic analysis; the second is to not allow two clauses juxtaposed next to each other without punctuation such as a comma. However, students tend to not be very good at getting their punctuation correct. The current implementation of the parsing algorithm ignores all punctuation other than full stops for this reason. In fact, there is a trade-off between allowing the system to parse ungrammatical sentences and the number of parse trees produced. More flexibility in grammatical errors increases the number of parse trees.

An issue that makes parsing problematic is that of adverbs. These tend to be allowed to occur within several places within the core and some, such as "yesterday", modify groups of words rather than a single word. The best solution, given their relative freedom of placing and the fact that sorting out where best to put them is more a meaning than a syntactic issue, would be to remove them and work out where they belong once the main verb and arguments have been identified.

Most of the above issues have to be left to an analysis of meaning to sort out the correct parse. There is no clear division between syntax and semantics. However there is another issue that has been highlighted to do with grammar and punctuation. How tolerant of errors should the system be? We have shown that errors in the operators do not cause problems for the parser, and errors in the placing of adverbs are relatively easy to deal with, but errors in the main constituents are not handled. For example, the phrase "the main people you need to ask will not be in the labs so early unless that have got work to hand in" occurs in one of the texts. The current algorithm will not handle these kinds of mistakes. Should the system be able to handle these kinds of mistakes, or should students be encouraged to improve their writing skills?

Figure 4: An example of a correctly parsed sentence

Figure 5: An example of a correctly parsed ungrammatical sentence

4. Conclusions and future work

Although, the initial results are promising, a lot more work needs to be done before this work can be included in an automatic marking system. Determining a good set of templates according to RRG theory is hard, and it is extremely difficult to keep the number of parses down to a small number. However, the use of statistical approaches should help to alleviate this problem as will some attention to semantics.

One particularly difficult aspect of parsing has been the noun phrases and the forms of arguments of verbs. When an argument is a straightforward noun phrase, parsing is straightforward, but we have numerous examples where the arguments are very complex noun phrases or another complex structure. An example of a complex noun phrase is: "possible sources of bias introduced into finding out students clubbing habits by asking those in the labs between 9:30 and 10:30 on a Thursday morning". This noun phrase is parsed correctly, but so far other kinds of complex arguments are not, such as "by randomly selecting an equal amount of students out of each year on varying courses by using their student ID email and sending them a questionnaire or just asking at popular locations within the campus, keeping a tally in order to meet the specification of the sample". All the latter is a single argument to "This could be done". Clearly the sentence would be easier both to parse and for other people to understand if it was broken up into several sentences. If the computer is to be able to handle any text automatically, however, this level of complexity really should be handled.

One line of attack to handle complexity would be to compare these examples with texts where (unlike for these examples) students are given marks for presentation and grammar. It may be that these texts are much easier to parse, and if this is the case, then we may be able to put some constraints on the complexity of sentences that students are allowed to input to an automatic marking system. One constraint that it would be useful to add would be to constrain the formatting that students are allowed to use. Formatting in the form of bulleted lists, especially of paragraphs, is a topic we have not even considered. It should be possible to handle other types of formatting such as simple lists or the use of colons, which are not handled at present, but even this adds a significant level of complexity to the parsing problem.

At present, verbs are simply categorised depending on how many arguments they take. Parsing may be easier if we could take into account the kind of arguments they take. For example, the verb "eat" is generally found with relatively simple arguments, whereas "suggest" or "recommend" generally take much more complex arguments. In the future, we plan to use this work as the first stage in a system that uses a new semantic framework, ULM (Universal Lexical Metalanguage) [3] to compare the meaning of student texts with a (single) model answer. A core part of ULM is to link syntactic structure and semantics together, and part of this is to specify the types of arguments in more detail along with the syntactic structure that goes with them. For example, the verb "see" takes simple arguments when used with standard simple sentence structure (such as "I see lots of trees"), but more complex arguments when used in a more figurative sense, as in "I see that Jim has gone home".

The ultimate aim of using ULM would be to enable us to convert text to a meaning representation. The aim is to build up a meaning representation from several sentences and then compare the meaning of the student text with the model answer – even when the words used are not the same. Clearly much work needs to be done before this aim is achieved.

5. References

[1]. Chung, H. and Rim, H.-C. "Unlexicalized Dependency Parser for Variable Word Order Languages based on Local Contextual Pattern***", Lecture Notes in Computer Science: Computational Linguistics and Intelligent Text Processing*** *(5th International Conference CICLING): Seoul 15-21 Feb*. 2945, 2004 pp 112-23.

[2]. Covington, M. A. "A Free Word Order Dependency Parser in Prolog**".** http://www.ai.uga.edu/mc/dparser/dparser.pdf [last accessed Sep 2008]

[3]. Guest, E. and R. Mairal Usón, "Lexical Representation Based on a Universal Metalanguage*"*. **RAEL, Revista Española de Lingüística Aplicada,** 4: 2005 pp. 125-173

[4]. Holan, T. "Dependency Analyser Configurable by Measures"**.Text, Speech and Dialogue 5th International Conference TSD***: Brno, Czech Republic, Sep 9-12*,.2002 pp *81-8.*

[5]. Hou, L. and Cercone, N. "Extracting Meaningful Semantic Information with EMATISE: an HPSG-Based Internet Search Engine Parser". **IEEE International Conference on Systems, Man, and Cybernetics***: Tucson, AZ Oct 7-11*. vol 5, 2001 pp2858-66.

[6]. Kešelj, V. "Modular HPSG". **IEEE International Conference on Systems, Man, and Cybernetics***: Tucson, AZ Oct 7-11*.vol 5, 2001 pp 2867-72.

[7]. Van Valin, R. D. J. **Exploring the Syntax-Semantics Interface.** Cambridge: Cambridge University Press. 2005

[8]. Van Valin, R. D. J. and LaPolla, ***R.Syntax: Structure, Meaning and Function.*** Cambridge, Cambridge University Press. 1997

[9]. Wahlster, W. **Verbmobil: Foundations of Speech-to-Speech Translation.** Berlin: Springer. 2000