RUG: Regular Unification Grammar

Lauri CARLSON
University of Helsinki
Research Unit for
Computational Linguistics
Hallituskatu 11
SF-00100 Helsinki
Finland

Abstract

The paper describes a new *unification based grammar formalism* called **Regular Unification Grammar (RUG)**. The formalism is under development at the Research Unit of Computational Linguistics, University of Helsinki. In outline, RUG can be described as a combination of an extended graph unification formalism with a fixed minimal finite state syntax.

Section I of the paper outlines the RUG formalism. Section II describes some aspects of its current implementation. Section III describes an experimental RUG grammar for Finnish.

I. The RUG formalism

RUG constitutes a unification based grammar formalism /Shieber86/. In outline, RUG can be described as a combination of an extended graph unification formalism with a fixed minimal finite state syntax. It shares with categorial unification grammar (a) the use of graph unification as the basic descriptive mechanism and (b) association of combinatorial properties of words with lexical entries. It differs from categorial grammar in restricting string combinatorics to left associative concatenation.[1]

1. Combinatorial syntax

The combinatorial syntax of RUG consists of the following three rules[2]:

```
(S Words
        ((0)(1))
        ((0 PARSE) (1 initial next word))
        ((1 final) @FinalState)
        ((1 next) (1 final current))
        ((1 final preceding) (1 current)))

(Words Words W
                ((0) (2))
                ((2) @InternalState)
                ((2 preceding) (1 current))
                ((1 next) (2 current))
                ((1 initial)(2 initial)))

(Words W
        ((0) (1))
        ((1) @InternalState)
        ((1 initial) @InitialState)
        ((1 preceding) (1 initial current))
        ((1 initial next) (1 current)))
```

In brief, a grammatical string consists of words and words consist of one or more words. Each word has a state associated to it, i.e. a feature structure which can be used to store information about the word and the state of the parse up to that word. In addition, the syntax provides dummy initial and a final states which can be used to state constraints common to all strings. Each state has a pointer to its own current contents and to the contents of the preceding and next states. Using unification, features of neighboring words can be accessed directly and features of more remote states through sharing.

2. Feature structure

A characteristic of the RUG unification formalism is the use of *cyclic* feature structures. In general, the graph associated to a sentence in a RUG grammar is not a tree nor a dag, but a connected graph. Dependency relations are shown over the list of words in a sentence bottom up, each word pointing to its head. In addition, a head can constrain its subcategorized complements through appropriate attributes. This reflects predictability: a head selects its complements (constrains their number), while adjuncts are not subject to selection and hence cannot be identified on the basis of the head.[3]

The graph unification formalism used in RUG contains facilities for expressing *indeterminate functional dependencies* among words using *regular path expressions*. An example of an indeterminate functional dependence is the dependence of a preposed question or relative word on some verb complement to its right. The class of possible heads of the word can be defined in terms of a regular expression over attribute names, say `(verb main VCOMP*)` for "some verb complement of the main verb of the clause".[4]

RUG allows disjunctions and negations of atomic feature values. (ANYOF A B) unifies only with A and B and (NONEOF A B) with any atom except A and B.

Nonmonotonic extensions of unification are available for completeness checking. ANY values /Shieber 1986/ allow testing for the presence of obligatory constituent at the end of a parse. Analogous tests for feature instantiation after each successful unification are available.[5]

RUG allows specifying default values through the reserved attribute name DEFAULT. DEFAULT features are unified like any other features. During parse final completion, a dg is overwritten over the value of its own DEFAULT and the result replaces the original dg.

3. Tools for grammatical abstraction

The RUG grammar formalism starts out with lower level primitives than other unification based grammar formalisms. In particular, the notion of a phrase (constituent) built in to context free grammar must be reconstructed in terms of unification. On the other hand, the absence (or optionality) of the requirement of proper nesting can be a help in dealing with free word order.

The development of the grammar formalism involves defining suitable abstractions in terms of the primitives of the unification formalism which can be used in actual grammar writing. The template abbreviation facility of PATR /Shieber et al. 1983/ with a few extensions is used in RUG for this purpose.

One extension of the template formalism is the ability to define *disjunctive templates* using the reserved word OR. A specification of form (OR specList specList ... specList) is compiled into a list of dgs one for each disjunct. This helps keeping the lexicon simple as different uses of the same word can be listed under one template name.

Another extension is *parametrized templates* which allow defining abstract operations on paths, values, or other templates. A specification of form (@ Name arg_1 ... arg_n) is compiled into whatever Name would compile to after arg_1 ... arg_n replace corresponding placeholders in the definition of Name.

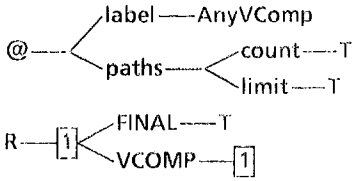
Using the template facility, higher level grammatical abstractions can be defined, for instance word or construction types such as subordinating connective, premodifier, etc. As all syntactic information is stored in templates, the property inheritance scheme implicit in the lexical template formalism can be used to express syntactic generalizations (say, to define a set of related clause types).

II. Implementation

RUG is currently implemented in REGDPATR, an extension of the D-PATR grammar development environment /Karttunen 1986/.

Regular path expressions are implemented by allowing dgs in attribute position of other dgs. Such an *attribute dg* (attrDg) is interpreted as the (possibly infinite) disjunction of the paths contained in it. For instance, the R attribute of the following attrDg is equivalent to the regular path expression (VCOMP*):

(1)

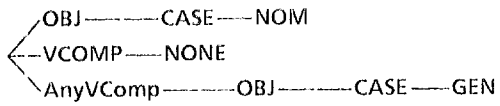


An attrDg can be defined and named in a template definition like any dg. Once defined, the name can be used in a path specification preceded by an @ sign. It is decoded and compiled into the corresponding attrDg. For instance, (1) can be defined in the lexicon as

(2) (AnyVComp ((VCOMP) NIL) (FINAL T))

We shall call dgs containing attrDgs *regular dgs* (regDgs). When a regDg is displayed, its attrDgs are labeled with their template names:

(3)



RegDgs are not guaranteed consistency by unification alone. The regDg in (3), for instance, is inconsistent, with the attrDg AnyVComp as shown in (1).

To supplement unification, another operation of *unit path resolution* is provided. Unit path resolution is very much like unit resolution in propositional logic. Recall that attrDgs are interpreted as disjunctions of paths. Likewise, we can interpret a simple dg as a conjunction of paths with given values. A regDg like (3) can thus be interpreted as a conjunction of disjunctions of paths, some of which (those consisting of atomic attributes) constitute unit disjunctions. This sets the stage for resolution.

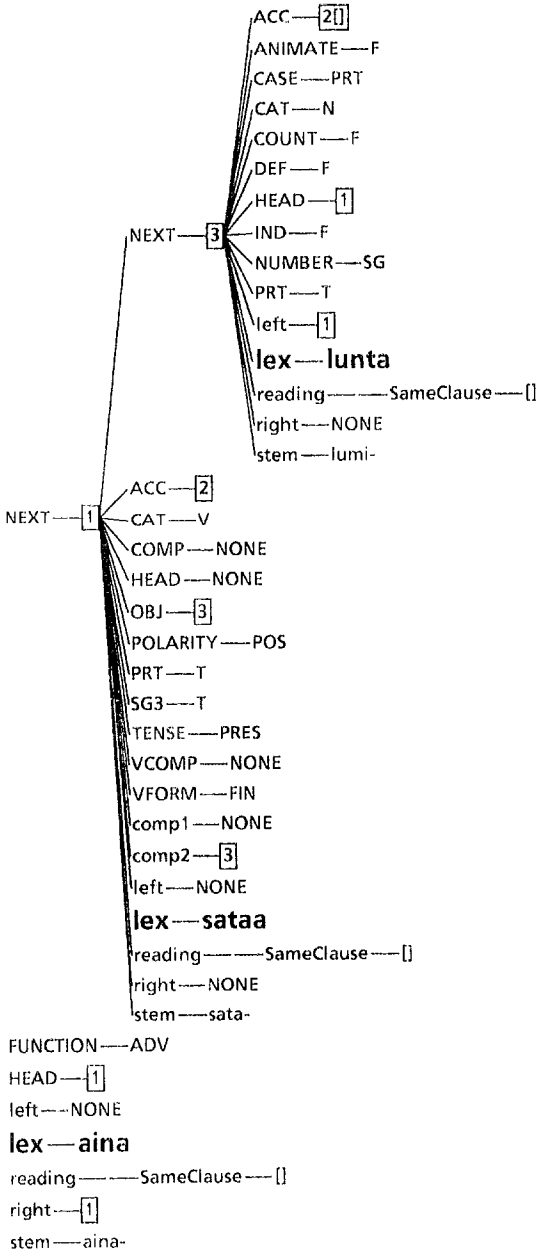
In unit path resolution, paths contained in each attrDg are matched with unit paths in the conjunctive "part" of the regDg looking for inconsistencies in the values at the end of identical paths. When an inconsistency is found, the corresponding path is removed from the attrDg. If all of the paths in the attrDg are thus removed, the regDg was inconsistent to start with. Otherwise, we obtain a consistent regDg with fever alternative paths left in it. This operation is undoable just as unification itself.

Path resolution can be incorporated as a stage in the unification of regDgs. Alternatively, it can be performed after each successful match or only after a parse is concluded. Unit path resolution is not complete, so all inconsistencies are not guaranteed to be detected by it.

III. FRED9: A RUG grammar for Finnish

1. Examples

(4)



FRED9 can be seen as an attempt to cast some of the grammatical ideas implicit in the procedural parser FPARSE of /Karlsson 1986/ into a declarative form.

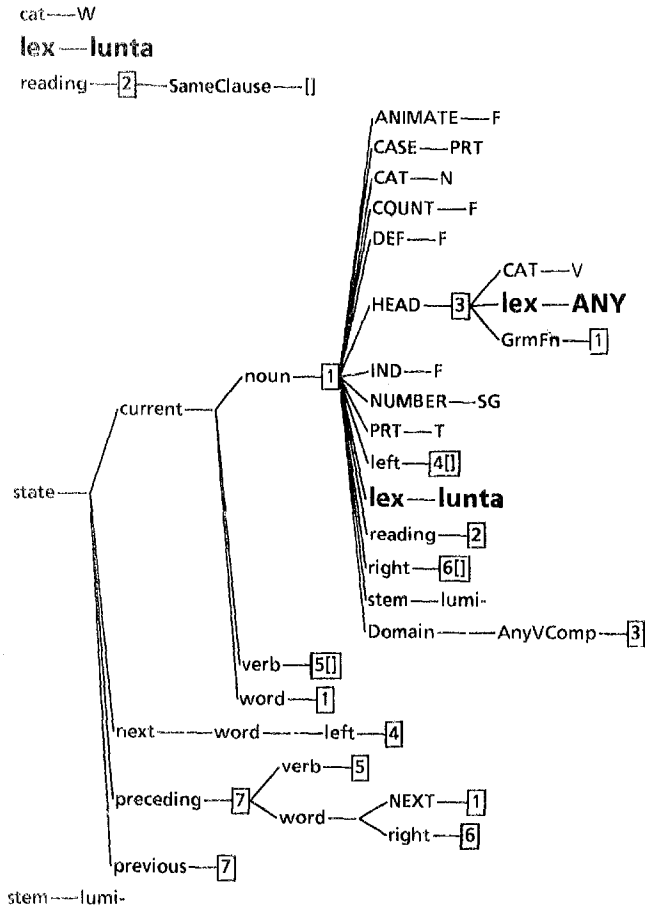
The structural descriptions produced by FRED9 resemble graphs used in traditional grammar. (4) describes the sentence *aina sataa lunta* "it always snows" (lit. 'always rains snow') . (4) is produced by unifying appropriate entries of the participant words one after another as directed by the syntax. (5) is the feature representation of the appropriate reading for *lunta* 'snow (prt)'.

The regular path expressions GrmFn and Domain AnyVComp characterize the ranges of possible functions and heads of *lunta* respectively. GrmFn and Domain have the definitions shown in (6).

2. Problems

Properties to account for in syntactic parsing include word order, dependency, consistence, completeness, and ambiguity. Word order and dependency together characterize what is commonly understood as syntactic surface structure. Notions of completeness and consistence describe two complementary constraints on grammaticality: consistence requires that a grammatical string must not contain too much information (too many or incompatible words), while completeness requires that a grammatical string must not contain too little information (missing or insufficiently specific words). The description of FRED9 below is organized around these five headings.

(5)



1. Word order

Free word order presents no inherent difficulty in RUG, as there is no built in connection between phrase structure and linear order like the proper nesting condition of context free grammar. For instance, Finnish allows scrambling dependents of a VCOMP chain anywhere inside the chain. This is described in FRED9 by the indeterminate head specification AnyVComp. For instance, in

(7) *Aina voi luntajoku alkaa luoda*
always can snow someone begin shoveling
"Someone can always begin shoveling snow"

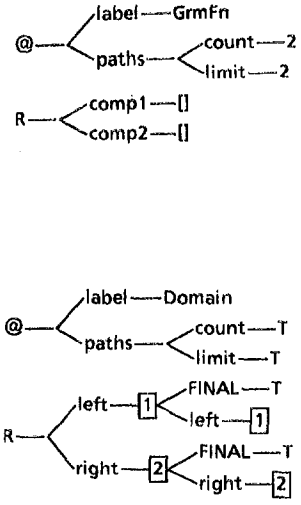
joku is the subject of *voi* and *lunta* is the object of *luoda*. What they have in common is that they depend on some verb on the VCOMP chain of the main verb *voi*.

2. Dependency

The converse side of the coin is that properly nested phrase structure does not come for free. Phrase structure has to be reconstructed using unification. One way to proceed is to use features acting as pointers to phrase heads, shared by the members of a phrase and linked to one another to form a phrase level projection of the string. Such projections form domains within which words can look for appropriate heads.

Center embedding can be managed with stack valued features. Proper nesting can also be enforced by a separate template Nested which requires that a word must not look for heads beyond its adjacent words' heads :[6]

(6)



(8) (Nested ((HEAD) (@Adjacent word @AnyHead))

In view of the difficulty of speakers to manage proper nesting deeper than one level or two, RUG seems to get into difficulties in the right direction.

3. Consistence

Maintaining consistence is in general easy given unification. For instance, the functional uniqueness principle (grammatical functions are unique per clause) is practically built in. For another example, a verb can have at most two grammatical case complements in Finnish. On the other hand, each grammatical case can have a number of functions (SUBJ, OBJ, PREDCOMP, OBL) depending on the verb. These constraints are maintained in FRED9 by allowing verbs two grammatical complement slots comp1, comp2 and specifying the function GrmFn of grammatical cases as the alternation of these slots. Further matching of verbs with cases is associated to the verbs. The two-way transfer of information through cyclic pointers between head and complement allows us to attach each constraint on the more informative member of the pair.

4. Completeness

Conversely, completeness is in general more difficult to ensure. Completeness cannot be expressed in terms of unification. Syntax can perform completeness checking by imposing suitable constraints on strings of category S. In particular, certain features can act as flags or stacks whose values at the final state are checked in the S rule. More directly, the nonmonotonic devices described in Section I.2 allow expression of obligatoriness or default values. As a general point, RUG grammars need not be restricted to parsing complete sentences or even constituents. A string of words is incomplete in some sense if the functions of some words in it remain unresolved. The string can still obtain a structural description specifying that fact in addition to whatever definite information can be gleaned from it.

5. Ambiguity

Since syntactic ambiguity is coded on lexical entries, multiplication of lexical entries for a given word is to be expected. In FRED9, the following policies are followed with regards to constraining lexical ambiguity. First, the use of unification makes it possible to replace some cases of ambiguity with underspecification.

Second, readings which are in complementary distribution can sometimes be coded into one entry which is accessed differently by the different contexts. FRED9 has just one entry for the uses of the copula *on* as an auxiliary and as a main verb in both predicative and existential constructions.

Third, ambiguities whose resolution has no effect on surrounding context can be localized into regular path expressions.

Fourth, an ambiguity which is resolved by immediate context can be left as a lexical ambiguity. The main consideration is that ambiguities do not begin to multiply during the parse.

Footnotes

[1]/Hausser 1986/ imposes a similar restriction on categorial grammar.

[2]The format is that of D-PATR /Karttunen 1986/. Lists of form ((...)(...)) represent path equations and atoms of form @... refer to grammar specific template definitions.

[3]This is why in categorial grammar, adjuncts are construed as functors. Complements usually come out as arguments. Cf. however Karttunen /1986/.

[4]Cf. /Kaplan and Zaenen 1986/. REGDPATR allows expressing alternation and iteration of paths. Complementation is not implemented.

[5]Such checks can sometimes do the job of features acting as flags.

[6]Cf. the adjacency principle in /Hudson 1984/.

References

Hausser, R. (1986) *NEWCAT: Parsing Natural Language Using Left-Associative Grammar.* Springer Verlag, Berlin/Heidelberg/New York, 1986.

Hudson, R. *Word Grammar.* (1984) Basil Blackwell, Oxford.

Kaplan, R. and Zaenen, A. (1986) Long-Distance Dependencies as a case of Functional Uncertainty. In Baltin (ed.), *Alternative conceptions of Phrase Structure*, New York.

Karlsson, F. (1986) Process grammar. in Dahl (ed.) *Papers from the IX Scand. Conf. of Linguistics*, Stockholm.

Karttunen, L. (1986a) D-PATR: a development environment for unification-based grammars. *Proceedings of COLING-86*, Bonn, pp. 74-80.

Karttunen, L. (1986b) Radical lexicalism. in Baltin (ed.), *Alternative Conceptions of Phrase Structure*, New York.

Shieber, S. (1986) *An Introduction to Unification-Based Approaches to Grammar.* CSLI Lecture Notes Series, No. 4 (distributed by the University of Chicago Press, Chicago, Illinois).

Shieber, S.M., H. Uszkoreit, F.C.N. Pereira, J.J. Robinson, and M. Tyson (1983). The Formalism and Implementation of PATR-II. In *Research on Interactive Acquisition and Use of Knowledge.* Artificial Intelligence Center, SRI International: Menlo Park, California.