Unification and the new grammatism

Steve Pulman
University of Cambridge Computer Laboratory
Corn Exchange Street
Cambridge CB2 3QG, UK.

What are we talking about?

The prototypical unification grammar consists of a context-free skeleton, enriched with a set of feature+value specifications on the grammatical symbols in the rules and associated lexicon. These feature specifications may involve variables, and may be recursive (i.e. the values may be interpreted as referring to a whole category). Whereas parsing and generating sentences using grammars with atomic grammatical labels involves a test for equality between symbols in a rule and those in a tree, in unification grammars the test is whether two non-atomic descriptions 'unify', i.e. can be made identical by appropriate mutual substitutions of terms. This mechanism can be used to enforce identity and coocurrence restrictions between feature values, and to 'percolate' such values between nodes. Thus, to take a simple example, a rule like:

$A \rightarrow B\ C$
$\mathrm{foo}(A) = \mathrm{foo}(B)$
$\mathrm{baz}(B) = \mathrm{baz}(C)$

where the equations specify constraints on the possible values of the features 'foo' and 'baz', can be understood as 'percolating' the value of foo from B to A, or vice versa, and as enforcing agreement between baz values on the daughter categories. But these are not separate operations or procedures: in fact, they are not operations or procedures at all. They are order independent, fully declarative statements about what must be true for a tree to be well formed whose implementation by unification has these desirable effects. Thus unification provides a neat and manageable solution to the problem of feature agreement, and feature percolation between nodes in a phrase structure tree: a problem which has produced more ad-hoc and underspecified 'solutions' than almost any other in linguistics. (Nor has the computational paradigm done any better: the usual solution in NLP has been to decorate CF rules with arbitrary bits of Lisp code).

This high level syntax in terms of equations may translate down into a form suitable for unification of terms, resulting perhaps in a rule like:

A[foo(X)] → B[foo(X), baz(Y)] C[baz(Y)]

or for unification of graphs, perhaps represented more like:

```
[  [mother     [category A [foo X]]   ]
   [daughter1  [category B [foo X]
                           [baz Y]]   ]
   [daughter2  [category C [baz Y]]   ]  ]
```

The actual 'unification' used may range from strict (first order) logical unification to more or less arbitrarily powerful pattern matching (Functional Unification Grammar, Kay 1979), with many other variants aimed at capturing specific types of linguistic regularity (Generalised Phrase Structure Grammar, Gazdar, Klein, Pullum and Sag, 1985; Lexical Functional Grammar, Bresnan, ed. 1982). Furthermore, a system of default values for features may be employed, so that, unlike ordinary term unification, it can make sense to unify two expressions like 'A[foo(a)]' and 'A[baz(b)]' as 'A[foo(a),baz(b)]'. Implicitly what is happening here is that we are assuming a default value for each feature (as a variable here), and thus actually employing ordinary term unification on 'A[foo(a),baz(X)]' and 'A[foo(Y),baz(b)]'. (This way of looking at it brings the set-theoretic and the pattern-matching sides of unification more obviously together.) However, if we had declared different default values unification might fail in such cases: a default value b for foo, for example, would prevent unification in this example.

What can you do with unification grammars?

Assume that we are talking about ordinary unification (i.e. allowing for category valued features, but not for negation, disjunction or other types of testing or pattern matching). A CFG enriched with such unification is still an extremely powerful system. Given the ability to manipulate category valued features we can generate some context-sensitive languages directly (at least). We can use the feature system to mimic a Turing machine (though this may not be reflected in weak generative capacity of the resulting grammar): and can simulate or implement many other apparently widely diverse grammatical formalisms: Fillmore type case grammars, categorial and dependency grammars, indexed grammars, some aspects of systemic networks, and so on. We can express such grammars in a wholly declarative way in the confidence that there is a theoretically clean and fairly efficient computational interpretation of them. Furthermore, viewed simply as a computational device, this type of unification can be used for several other types of linguistic structural manipulation beyond those involved in morphological or syntactic analysis:

(i) building up explicit, possibly decorated, parse trees themselves during the course of recognising a sentence, as is done in, e.g. Definite Clause Grammars (Pereira and Warren, 1980).

(ii) building up logical forms compositionally by using extra features to represent the function-argument structure of a constituent as is done in e.g. the PATR formalism (Shieber, forthcoming). With ingenuity, the effects of function application, composition and so on can be simulated using logical variables within a feature system. More simple predicate-argument structure can of course be built directly.

(iii) assigning prosodic contours on the basis of syntactic and lexical structure, by associating values for relative prominence and direction of pitch movement to the components of a constituent. Unification ties in all these values in such a way that the absolute pitch value of some constituent may ultimately depend on that of some higher level constituent, or of some some other phrase some distance away. To the extent that prosody is syntax driven at all, this is a theoretically clean way of deriving default intonation contours from the parse tree of a sentence.

Unification as a contribution to linguistic theory

It should be clear from the foregoing that there are at least two different ways in which we could assess the notion of unification, and unification grammars. Given a particular notion of unification as incorporated in a linguistic theory (say, that defined in GKPS85), we can ask whether it enables us to express linguistic generalisations clearly, and whether it meshes in satisfactorily with other mechanisms used within that theory (for example, default feature specifications, or cooccurrence restrictions). In short, we are assessing unification as a claim about human linguistic ability. Depending on the formal properties of the type of unification at issue, this may or may not be a meaningful thing to do: if we have a system that can do anything, it is not news to be told it is adequate for syntactic description.

In fact, it strikes me as most unlikely that everything one would want to say about the syntactic structure of language would turn out to be expressible cleanly within one particular flavour of unification grammar (pace LFG, FUG, GPSG). Even if this did turn out to be the case, it would actually be mildly disturbing, given current dogma about modularity, to find that a mechanism conceived of originally for syntax turned out to be so easily adaptable for other possibly unrelated kinds of symbolic manipulation. It is almost certain that any theory with unification as a component will be capable of simulating most of the features of any other such theory, when regarded merely as a notation. Unification is a very powerful symbol manipulation tool, apart from anything else. Thus arguments about whether XUG is better that YUG, for many values of X and Y, are liable to be as ultimately unproductive (except perhaps of entertaining rhetoric) as most other competitive linguistic arguments.

Unification grammars as a linguistic *lingua franca*

In my view, a more useful way of thinking about unification, from the point of view of computational linguistics, at least, is to see it as merely a useful computational procedure with a well defined semantics and efficient implementations. It seems to me that the real role of unification in grammatical formalisms will be to provide a kind of normal form of guaranteed computational tractability, or perhaps better, an assembly language into which different linguistic theories can be compiled. Take the example of feature percolation given above: as far as the linguist is concerned, the first statement involving explicit equalities says everything there is to say about when a tree is or is not well formed. The linguistic theory need not be committed to any of the concepts presupposed in the translation to the second format where the equations are implemented via unification: for example, the notion of a logical variable need not figure in the linguistic theory at all, although it must figure in the implementation.

Thus the relation between a prototypical unification grammar of the type outlined above, and a particular linguistic theory, would be akin to that between the compiled code which executes a program. and the original raw form of the program (or even some higher level description). We have already seen that many different linguistic theories can be translated into a unification grammar format: the original insights and theoretical content of the theories is presumably independent of this translation. The grammarian says what he wants to say about the structure of the language, in some high level declarative formalism, and for the purposes of parsing or generation, this high level description is compiled out into a lower level, simple unification formalism, for which there exist well understood computational interpretations.

This compilation process serves several purposes: the practical one of ensuring that you can actually do something with the grammar: parse, or generate sentences. A second purpose, somewhat akin to the requirement of Turing machine reducibility on psychological theories, is achieved by the fact that the existence of such a compilation serves as a guarantee that the original theory is consistent, and has a coherent computational interpretation. (The version of GPSG in GKPS85 turns out to have some problematic features in this respect). Finally, in supplying such a normal form, a unification formalism provides a rational basis for the comparison of different grammatical theories: you have to compare like with like, and this can more easily done via translation into a common format. This is an aspect of unification formalisms which has been explored for formalisms like LFG and GPSG using the PATR formalism by the CSLI foundations of grammar group (Shieber 1985).

References

Bresnan, J. (ed) 1982 *The Mental Representation of Grammatical Relations*, Cambridge, Mass: MIT Press.

Gazdar, G., Klein, E., Pullum, G. and Sag, I. 1985, *Generalised Phrase Structure Grammar*, Oxford: Basil Blackwell

Kay, M. 1979 *Functional Grammar*, in Proceedings of 5th annual meeting of the Berkeley Linguistics Society, ed. C. Chiarello et al, University of California, Berkeley, 142-158.

Pereira, F. and Warren, D. 1980 *Definite Clause Grammars for Language Analysis*, Artificial Intelligence 13, 231-278.

Ritchie G. D. 1984 *A Rational Reconstruction of the Proteus Sentence Planner*, in Proceedings of the 10th International Conference on Computational Linguistics/22nd Annual meeting of the Association for Computational Linguistics: Standord, ACL, 327-329.

Ritchie G. D. 1986 *The computational complexity of sentence derivation in functional unification grammar*, in proceedings of the 11th International Conference on Computational Linguistics, Bonn: ACL, 584-586.

Shieber, S. M. 1985 *Separating Linguistic Analyses from Linguistic Theories*, in Transcripts of the Alvey/ICL workshop on Linguistic Theory and Computer Applications, ed. P Whitelock et al., Centre for Computational Linguistics, University of Manchester Institute of Science and Technology.

Shieber, S. M. (forthcoming) *An Introduction to Unification-Based Approaches to Grammar*, University of Chicago Press, Chicago, Illinois.