COMIT $\Longrightarrow^{*}$ PATR II

Gerald Gazdar
University of Sussex

Here is the history of linguistics in one sentence: once upon a time linguists (i.e. syntacticians) used augmented phrase structure grammars, then they went over to transformational grammars, and then some of them started using augmented phrase structure grammars again, <space for moral>. Whilst we are in this careful scholarly mode, let us do the same service for computational linguistics: once upon a time computational linguists (i.e. builders of parsers) used augmented phrase structure grammars, then they went over to augmented transition networks, and then many of them started using augmented phrase structure grammars again, <space for moral>. There are people who would have you believe in one or other of these stories (e.g. Chomsky 1983, p65, for the first). And, of course, there is an element of truth in each of them. If an unrestricted rewriting system is an "augmented phrase structure grammar", then we can say that Chomsky (1951) propounds an augmented phrase structure grammar [1].

Turning to computational linguistics, let us consider two fairly well-known exemplars, one for the old grammatism (COMIT - Yngve 1958) and one for the new (PATR II - Shieber 1984). Both are computer languages, both were designed for computational linguistic purposes, notably the specification of natural language grammars with a view to their use in parsers. The two general criteria that Yngve explicitly notes as having motivated the design of COMIT, namely "that the rules be convenient for the linguist — compact, easy to use, and easy to think in terms of" and "that the rules be flexible and powerful — that they not only reflect the current linguistic views on what grammar rules are, but also that they be easily adaptable to other linguistic views" (1958, p26) are indistinguishable from two of the three general criteria that motivate the design of PATR II (Shieber 1985, pp194-197) [the third — computational effectiveness — may have been too obviously pressing in the late 1950s for Yngve to have thought worth mentioning explicitly]. Both have been implemented on a variety of hardware, and substantial grammar fragments have been written in both.[2]

Both COMIT and PATR II are, in some sense, and not necessarily the same sense, augmented phrase structure grammar formalisms. In examining the differences between them, it will be convenient to divide the topic into (1) consideration of categories, and (ii) consideration of rules.

Looking at the category formalisms first, both formalisms allow categories to have an internal feature structure, but there the resemblance ends. A COMIT category consists of a monadic name (e.g. "NP"), an optional integer "subscript", and a set containing any number of attribute-value pairs (called "logical subscripts"). Attributes are atomic, but values are sets containing between 0 and 36 atomic members. This is a sophisticated and expressive feature system by contrast to the impoverished phonology-based binary systems that most transformational syntacticians seemed content to assume, though scarcely to use, during the 1960s and 1970s. A PATR II category, however, is an arbitrary directed acyclic graph (dag) whose nodes are labeled with atomic names drawn from some finite set. Thus it easy to see how to translate a set of COMIT categories into a set of PATR II categories: the only minor complication concerns how you choose to encode the COMIT integer subscripts. But translation in the other direction is in general impossible, for all practical purposes, since COMIT logical subscripts do not permit any

[1] The notation Chomsky used mostly suggests a context sensitive rewriting system which allows null productions (hence type 0 rather than type 1). However, one nonstandard augmentation that is employed throughout the work is the "sometimes" notation, as in the following example from page 30.

Y^2 --> y, sometimes

This remarkable innovation does not seem to have found favor in later work except, perhaps, as the precursor of the "variable rules" that became fashionable in sociolinguistics in the 1970s.

[2] For some example COMIT grammars, see Dinneen (1962), Fabry (1963), Satterthwait (1962), Weintraub (1970), and Yngve (1967).

recursive structure to be built.[3]

Switching our attention now to rules, we observe that both COMIT and PATR II allow one to write rules that say that an expression of category A can consist of an expression of category B followed by an expression of category C. But a COMIT rule is a rewriting rule whose primary concern is that of mapping strings into strings, whereas a PATR II rule is a statement about a permissible structural configuration, a statement that concerns itself with strings almost incidentally. A rule with more than one symbol on the left-hand side makes no sense in the PATR II conception of grammar, but it makes perfectly good sense when the function of a rule is to change one string of categories into another string, as in the COMIT conception. COMIT rules give you unrestricted string rewriting, PATR II rules permit concatenation only. Thus COMIT rules cannot, in general, be translated into PATR II rules, and PATR II rules, thanks to the category system employed, cannot, in general, be translated into COMIT rules. COMIT rules are inextricably embedded in a procedural language: the rules are ordered in their application, every rule has an address, every rule ends with a GOTO-on-success, and rules can set and consult global variables in the environment (the "dispatcher"). PATR II rules, by contrast, are order independent, side effect free, and pristinely declarative. Both languages allow the user to manipulate features in rules, but whilst COMIT offers the user a small arsenal of devices — deletion, complementation, merger — of which the last-named appears to be the one most used, PATR II offers only unification. But are "merger" and "unification" two names for the same concept? The answer here is no: merge(A,B), where A and B are attribute values (hence sets), is the intersection of A and B if the latter is nonempty, and B otherwise.

There is nothing too surprising in any of the foregoing: as one might expect from the chronology, PATR II stands in much the same relation to COMIT as Scheme does to Fortran. If anyone wanted to do COMIT-style computational linguistics in 1987, then they would probably be better off using Icon than they would be using PATR II. What is distinctive about the new grammatism, as canonically illustrated by PATR II (but also exemplified in CUG, DCG, FUG, GPSG, HPSG, JPSG, LFG, UCG, ...) is (i) the use of a basically type 2 rule format (single mother, unordered, no explicit context sensitivity) under (ii) a node admissibility rather than a string rewriting interpretation, with (iii) a recursively defined tree or dag based category set, and (iv) unification as the primary operation for combining syntactic information.

It would be interesting to learn of any computational linguistic work done in the 1950s or 1960s that exhibits more than one of these characteristics.

Acknowledgements

I am grateful to Geoff Pullum for some relevant conversation and to Victor Yngve for making copies of unpublished COMIT work available to me. This research was supported by grants from the ESRC (UK) and SERC (UK).

References

Chomsky, Noam (1951). *Morphophonemics of Modern Hebrew.* MA thesis, University of Pennsylvania, Philadelphia, Pennsylvania. Published by Garland, New York, in 1979.

Chomsky, Noam (1983). *Noam Chomsky on the Generative Enterprise* [in conversation with Riny Huybregts and Henk van Riemsdijk]. Dordrecht: Foris.

Dinneen, David A. (1962). A Left-to-Right Generative Grammar of French. Unpublished PhD dissertation, MIT.

[3] The concern in this paper is only with what each formalism can naturally express, not with what you can do if you start playing tricks. Since both COMIT and PATR II are Turing equivalent, anything that can be expressed in the one, can be coded up somehow or other in the other one. Thus, for example, given some Godel-numbering scheme for dags, every PATR II feature structure could be mapped into a COMIT integer subscript (ignoring the 2^15 upper bound on the latter). But nobody in their right mind would do this.

Fabry, Robert S. (1963). Sentence Generation and Parsing with a Single Grammar. Unpublished MA dissertation, MIT.

Satterthwait, Arnold C. (1962). Parallel Sentence-Construction Grammars of Arabic and English. Unpublished PhD dissertation, Harvard University.

Shieber, Stuart M. (1984). The design of a computer language for linguistic information. *Proceedings of COLING84* 362-366.

Shieber, Stuart M. (1985). Criteria for Designing Computer Facilities for Linguistic Analysis. *Linguistics* 23, 189-211.

Weintraub, D. Kathryn. (1970). The Syntax of some English Relative Clauses. Unpublished PhD dissertation, University of Chicago.

Yngve, Victor H. (1958). A programming language for mechanical translation. *Mechanical Translation* 5, 25-41.

Yngve, Victor H. (1967). MT at MIT 1965. In A.D. Booth, ed. *Machine Translation.* Amsterdam: North-Holland, 452-523.