CURRENT TRENDS IN LOGIC GRAMMARS

Veronica Dahl
Computing Sciences Department
Simon Fraser University
Burnaby, BC V5A 1S6

Abstract

This paper surveys several logic grammar formalisms, relates them to some recent trends in linguistics and advocates the use of logic grammars for natural language processing. Contrary to many recent approaches that resort to augmenting essentially context-free grammars, it also tries to make a case for not outruling context-sensitivity or transformations. Finally, it presents a new logic grammar formalism jointly developed by Michael McCord and the author, the main features of which are: a metagrammatical treatment of coordination that relieves the grammar writer from having to describe coordinating rules explicitly; a modular treatment of semantics based upon simple information given locally to each rule, and an automated building-up of the sentence's representation structure.

1. Introduction

Among the computational formalisms for describing and processing language, logic grammars have been drawing attention since their introduction in 1975 (Colmerauer 1975).

Logic grammars resemble type-0 grammars, except that the grammar symbols may have arguments, and that procedures may be invoked from the rules (e.g. to serve as applicability constraints). Derivations involve unifying (Robinson 1965) symbol strings rather than just replacing them. Since the logic grammar formalism is a part of the Prolog programming language (Colmerauer 1975, Pereira L et al, 1978), logic grammars written to describe a language can be interpreted by Prolog as analysers for that language. Thus relieved from the operational concerns of parsing, the user can develop very clear and concise "analysers" just by writing a set of logic grammar rules that describe a language and giving it to Prolog. Logic grammars have been favourably compared with a widely used formalism for processing language: augmented transition networks (ATNs) introduced in 1970 (Woods 1970). They have been argued to be clearer, more concise and in practice more powerful, while at least as efficient, as ATNs (Pereira & Warren, 1980).

The first sizable application for logic grammars was a Spanish/French consultable database system (Dahl 1977,1981,1982) which was later adapted to Portuguese by H. Coelho and L. Pereira

(1), and to English, by F. Pereira and David Warren (2); and has since inspired the development of several other applications (e.g. Coelho 1979, McCord 1980, F. Pereira & Warren 1981). This system has been shown to be comparable in efficiency with the LUNAR system (Woods et al., 1972), (cf. Pereira & Warren, 1980, p.276), thus joining the appeal of practical feasibility to the elegance and expressive power of the logic grammar approach. Further logic grammar applications include (Silva et al. 1979, Simmons and Chester 1979, Sabatier 1980, Pereira et al. 1982). However the experience gained in the aforementioned applications has motivated the development of alternative logic grammar formalisms, some restricting and others augmenting the power of the original formalism as described in (Colmerauer 1975).

This paper attempts to fill a gap by examining the evolution of logic grammars, comparing the alternative proposals, and discussing them with respect to recent trends in both theoretical linguistics and natural language processing. It also motivates and briefly presents a new logic grammar formalism, called "modifier structure grammars" (MSGs), developed jointly by Michael McCord and the author (Dahl and McCord 1983). Its main features are: a metagrammatical (user-invisible) treatment of coordination, a modular treatment of semantics, and an automatic build-up of the parsed sentence's representation.

---

(1) Personal Communication, 1978.
(2) Personal Communication, 1980.

Section 2 describes logic grammars in intuitive, user-biased terms. Section 3 presents different types of logic grammars; Section 4 compares them with respect to expressive power, in particular through the example of how they allow to express movement of constituents. Section 5 discusses pros and cons of choosing relatively evolved grammar formalisms, and makes a case for choosing logic grammars independently of the degree of evolution needed. Section 6 briefly presents our new logic grammar formalism (MSGs), and Section 7 contains some concluding thoughts.

2. What is a logic grammar?

Logic grammars can be thought of as ordinary grammars, in which the symbols may have arguments. These arguments are either constants, variables or functional expressions, and the fact that they may include variables implies that substitutions are sometimes needed in order to apply a grammar rule. For instance, consider the following grammar:

```
1)  Sentence (fact(F)) --> proper-noun(N), verb(N,F).

2)  proper-noun (mary) --> [mary].
3)  proper-noun (john) --> [john].

4)  verb (N, laughs(N)) --> [laughs].
5)  verb (N, smiles(N)) --> [smiles].
```

in which (as throughout this paper) constants are in lower-case

letters, variables start with a capital, consecutive terminal symbols are represented as square-bracketed lists, and non-terminals are in lower-case letters. The comma stands for "concatenation", and the end of a rule is signalled by a period. Having defined these rules to Prolog, if we now, for instance, want to analyse the sentence "Mary smiles", we merely write the question:

```
? sentence(X,"mary laughs", [ ])
```

This amounts to a request that Prolog find a value for X that represents the surface form "Mary smiles" with respect to this grammar. What happens in the Prolog execution of the parsing procedure can be summarized in the top-down, left-to-right derivation tree depicted in Fig. 1, where each rule application is labelled by the identification of the rule involved and by the set of substitutions of terms for variables that are needed in order to apply the rule.

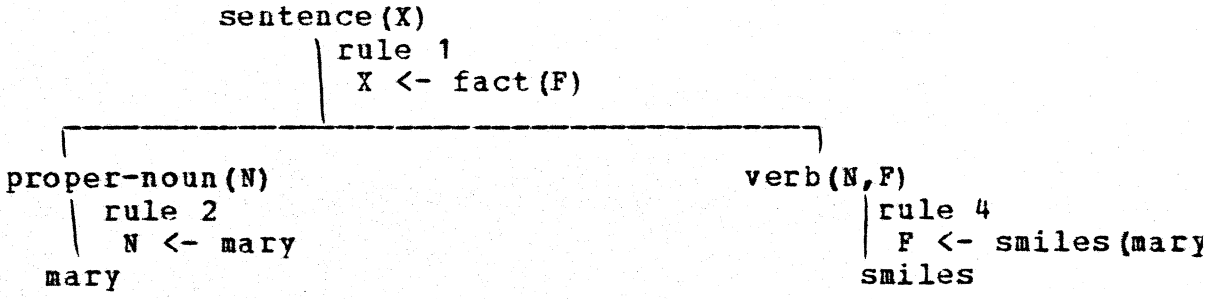

Figure 1. Derivation tree for "Mary laughs".

Only successful rule applications are shown here. Prolog backtracks upon unsuccessful ones. Through the substitutions employed, Prolog finds the representation

X = fact(smiles(mary))

for the sentence given.

Arguments allow for information to be shared by various grammar symbols, and to be carried along a derivation. In our example, they serve to build up a desired representation for a surface sentence. In procedural terms, grammar symbols can be thought of as producers and consumers of structure: the "proper-noun" symbol produces the value "mary", which is then consumed by "verb" in order to produce the structure "smiles(mary)", from which the final structure "fact (smiles (mary))" is constructed by "sentence".

Other uses in natural language processing include: syntactic and semantic checks (e.g. gender and number, semantic type or class), carrying extraposed constituents across phrases, etc.

For instance, we may check semantic accord by declaring Mary and John to be of type "human", and requiring that the arguments of "laughs" and "smiles" also be human. We use a functional symbol "-", in infix notation (allowed by Prolog) in order to introduce this semantic information. The above grammar becomes:

```
1)  sentence (fact(F)) --> proper-noun(N), verb(N,F)

2)  proper-noun (human-mary) --> [mary].
3)  proper-noun (human-john) --> [john].

4)  verb (human-N, laughs(N)) --> [laughs].
```

5) verb (human-N, smiles(N)) --> [smiles].

(Notice that variable names are local to each rule - i.e., variables with the same name are unrelated if they belong to different rules.)

We have enforced semantic agreement through unification, i.e., in the Prolog matching of terms. Proper nouns introducing non-humans now fail to be coupled with such verbs as "laughs" and "smiles".

Another way is through procedure calls, allowed in logic grammars in the form of Prolog calls (that we note between brackets). For instance, rule 4) could have instead been replaced by:

4) verb (N, laughs(N)) --> [laughs], human(N) .

and we would have added a Prolog definition for the procedure called, e.g.:

```
human(mary).
human(john).
```

More general procedures can, of course, be written in Prolog, e.g. "every child is human", noted:

```
human(x) :- child(x)
```

and read: "if x is a child then x is human".

3. Different types of logic grammars.

The first formulation of the parsing problem in terms of logic was obtained by A. Colmerauer and R. Kowalski, while trying to express Colmerauer's Q-System (Colmerauer 1973) in logic. This idea evolved into a very elegant and efficient Prolog implementation of metamorphosis grammars (Colmerauer 1975), that we shall call MGs.

An MG rule has the form:

$$S\ \alpha \text{ --> } \beta$$

where S is a nonterminal (logic) grammar symbol, α is a string of terminals and nonterminals, and β is like α except that it may also include Prolog procedure calls.*

Examples of such rules are:

```
a, [b] --> [b], a

verbroot (X), pluralmark --> [W], concat ([X],[s],W)
```

where a Prolog predicate concat (x,y,z) is assumed, that holds if z is the concatenation of x and y.

A special case of MGs was later included in DEC-10 Prolog

---

* The actual implementation in fact requires α to be a string of nonterminals, but we shall disregard the restriction since it has been shown (Colmerauer 1975) to involve no loss with respect to the full MG form.

(Pereira, Pereira & Warren 1978) and baptised definite clause grammars (DCGs). DCG rules have the form:

$$S \dashrightarrow \beta$$

where S and β are as above. All the rules presented in Section 2 are of this type.

The main motivation for introducing DCGs was ease of implementation coupled with no substantial loss in power (in the sense that DCGs can also basically describe type-0 languages - although less straightforwardly).

Extraposition grammars (XGs) (Pereira, to appear) were designed in order to refer to unspecified strings of symbols in a rule, thus making it easier to describe left extraposition of constituents.

XGs allow rules of the form

$$s_1 \ldots s_2 \text{ etc. } s_{k-1} \ldots s_k \dashrightarrow r$$

where the "..." specify gaps (i.e., arbitrary strings of grammar symbols), and r and the s_i are strings of terminals and non-terminals.

The general meaning of such a rule is that any sequence of symbols of the form

$$s_1 x_1 s_2 x_2 \text{ etc. } s_{k-1} x_{k-1} s_k$$

with arbitrary x_i s, can be rewritten into $r\ x_1\ x_2 \ldots x_{k-1}$

(i.e., the s_i's are rewritten into r, and the intermediate gaps (x_i's) are rewritten sequentially to the right of r. For instance, the XG rule:

relative-marker ... complement --> [that].

allows to skip any intermediate substring appearing after a relative marker * in the search for an expected complement, and then to subsume both marker and complement into the relative pronoun "that", which is placed to the left of the skipped substring.

The next section shows this rule at work in a parsing context.

Restriction grammars (RGs) (Hirschman & Puder, 1982) are not, strictly speaking, logic grammars, since the grammar symbols may not include any arguments. But they are implemented in Prolog and provide an instance of what seems to be a popular tendency in natural language processing nowadays: they involve sets of context-free definitions augmented with grammatical constraints or restrictions. In RGs, these appear in the form of procedures interleaved among the context-free definitions.

For instance, the RG rule

---

* i.e., a symbol that announces the beginning of a relative clause.

predicate ::= verb, object, verb-object

states that "predicate" can be rewritten into "verb object", provided that the "verb-object" restriction is satisfied (this restriction could for instance state that if the object is nil, the verb must be intransitive). Restrictions need to be defined separately, using such available primitives to traverse the tree as "up", and "down"; and explicitly stating parameters for its starting point in the tree and in the word stream.

4. Expressive power of each grammar formalism.

What are the consequences of choosing one of these grammar formalisms to write a natural language processor? From a theoretical point of view, the power of MGs, DCGs and XGs is similar in that they can all serve to describe type-0 languages. From a practical point of view, however, their possibilities differ. In this section we shall illustrate this point by studying how easily and concisely each of these formalisms allows to describe those rules involving constituent movement and ellision. RGs, although not dealing specifically with movement, are also considered.

4.1 Movement rules and MGs.

Let us consider for instance the noun phrase:

the man that John saw

which can be thought of as the surface expression of the more

canonical form:

the man [John saw the man],

where the second occurrence of "the man" has been shifted to the left and subsumed into the relative pronoun "that".

A simple grammar for (very restricted) sentences in canonical form could be:

(1) sentence --> noun-phrase, verb-phrase.

(2) noun-phrase --> determiner,noun,relative.

(3) noun-phrase --> proper-name.

(4) verb-phrase --> verb.

(5) verb-phrase --> trans-verb,direct-object.

(6) relative --> [ ].

(7) direct-object --> noun-phrase.

(8) determiner --> [the].

(9) noun --> [man].

(10) proper-name --> [john].

(11) verb --> [laughed].

(12) trans-verb --> [saw].

We shall successively modify this grammar (referred to as G in all that follows) in order to describe the relativization process within various logic grammar formalisms.

Within MGs, all we need is to add the following rules:

(6') relative --> relative-marker, sentence.

(5') verb-phrase --> moved-dobj, transitive-verb.

(13) relative-marker, noun-phrase,moved-dobj --> rel-pronoun,noun-phra

(14) relative-pronoun --> [that].

Figure 2 depicts the derivation tree for our sample noun phrase "the man that John saw". We abbreviate some of the grammar symbols. Rule numbers appear as left-hand side labels.

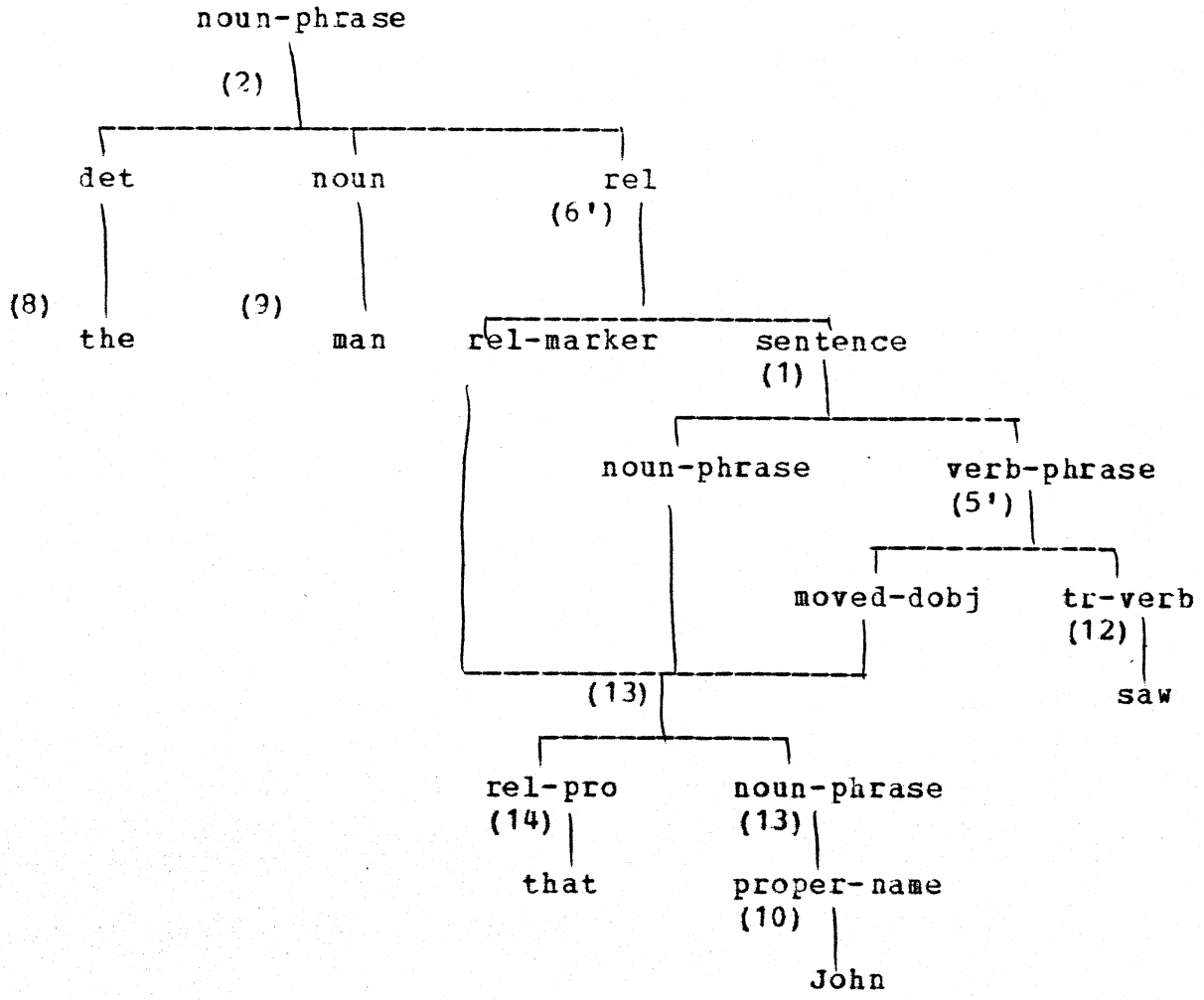

Figure 2. MG derivation tree for "The man that John saw".

Of course, for such a parse to be of any use, we need to construct a representation for the sentence while we parse it. But for the time being we shall ignore symbol arguments in order to concentrate upon the particular problem of moving constituents.

4.2. Movement rules and DCGs.

In terms of DCG rules, the simplest possible modification to the original grammar G is to allow a direct object to be ellided, e.g. by adding the rule:

(7') direct-object --> [ ].

But, because this rule lacks the contextual information found in (13), a direct object is now susceptible of being ellided even outside a relative clause. In order to prevent it, a usual technique is to control rule application by adding extra arguments. In our example, we only need to add a single argument that we carry within the sentence, verb-phrase and direct-object symbols, and that takes the value "nil" if the direct object in the verb phrase of the sentence is not ellided, and the value "ellided" if it is. The modified rules are the following:

(0) sentence --> sent(nil).

(1) sent(E) --> noun-phrase, verb-phrase(E).

(4) verb-phrase(nil) --> verb.

(5) verb-phrase(E) --> transitive-verb, direct-object(E).

(6') relative --> relative-pronoun, sent(E).

(7) direct-object(nil) --> noun-phrase.

(7') direct-object(ellided) --> [ ].

(13) relative-pronoun --> [that].

Figure 3 shows the DCG derivation tree for "The man that John saw laughed". Substitutions of terms for variables are shown as right-hand side labels.

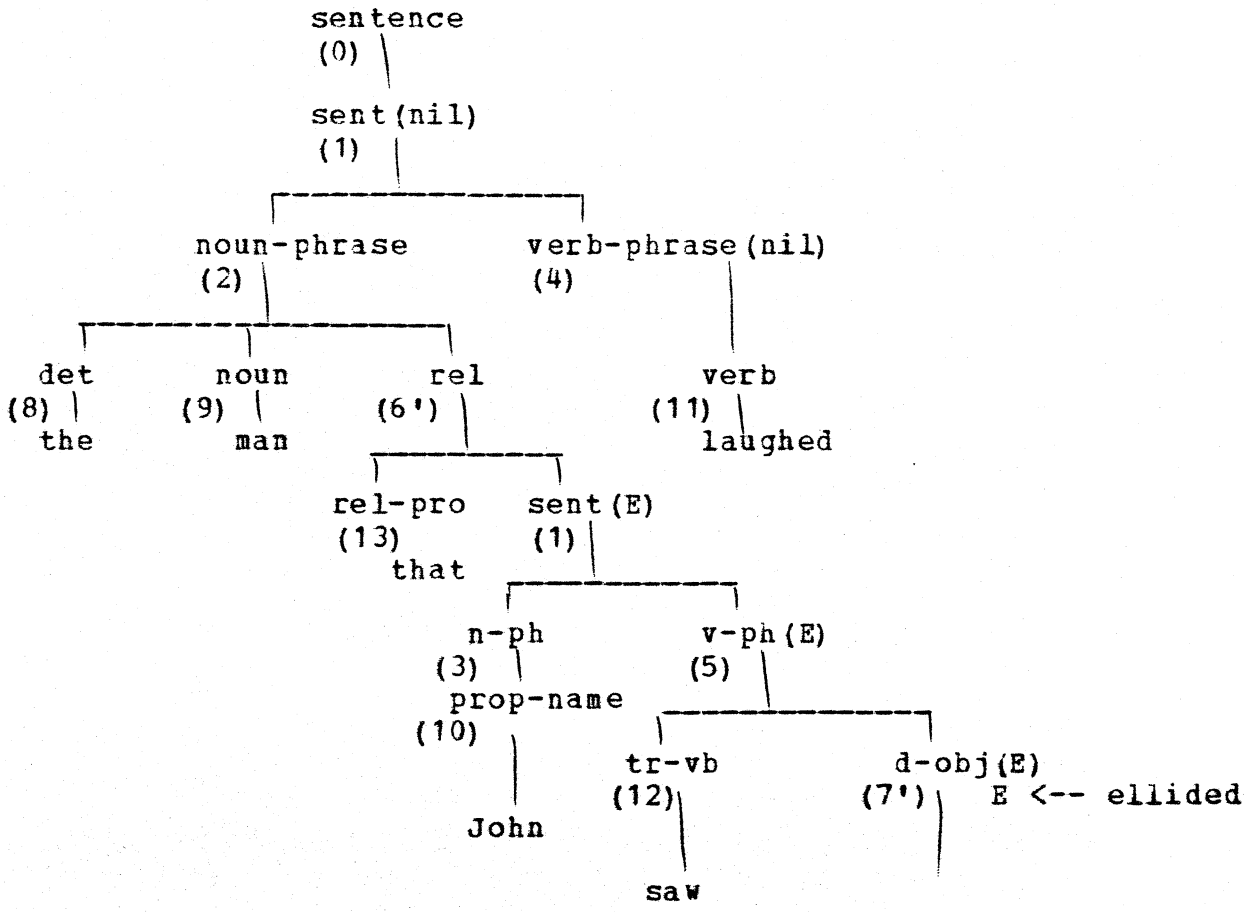

Figure 3. DCG derivation tree for the sentence "The man that John saw laughed".

4.3. Movement rules and XGs.

While, as we have seen, MGs express movement by actually moving constituents around, DCGs must carry all information relative to movements within extra arguments. XGs, on the other hand, can capture left extraposition in an economical fashion: by actually skipping intermediate substrings rather than shifting the constituents that follow. Thus, our initial grammar can be modified to handle relativization simply by adding the XG rules:

(6') relative --> relative-marker, sentence

(13) relative-marker ... direct-object --> relative-pronoun.

(14) relative-pronoun --> [that].

Figure 4 shows the XG derivation tree for "The man that John saw laughed".

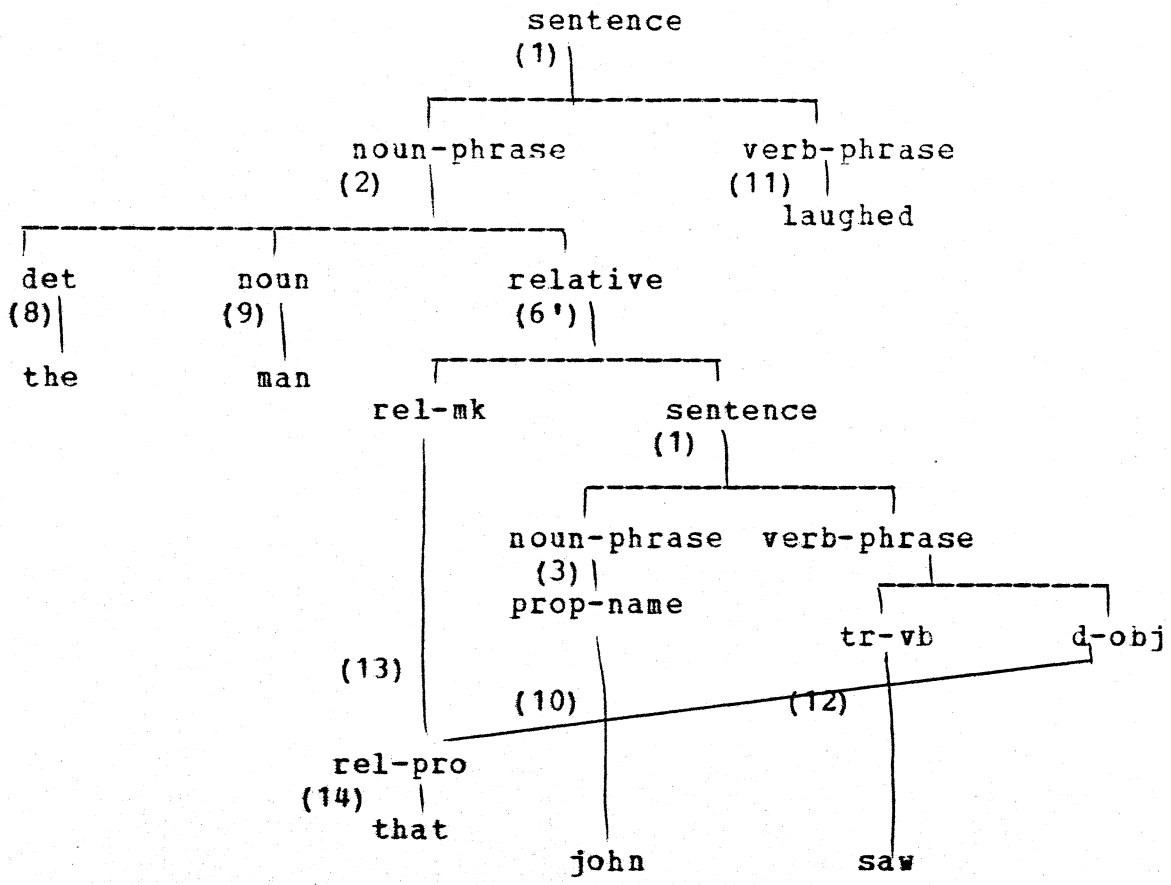

Figure 4. XG derivation tree for "The man that John saw laughed".

4.4. RGs

Restrictions can be used in RGs for the purpose of enforcing context sensitive constraints, but transformations seem to require an RG extension - possibly in the form of an additional component - , which is presently under study. (Hirschman & Puder, 1982)

An interesting feature of RGs is that a parse tree is automatically constructed during the parse (i.e., the tree-building parameters are hidden from the user). This makes a grammar clearer, but at the same time less flexible: only the history of rule applications is recorded, whereas in any other logic grammar the user may build up (through explicit parameters) any desired representation for the sentences parsed. The effects of context sensitivity, on the other hand, are ensured by giving each restriction access to the entire previously constructed parse tree. This need is the main difference between restrictions and the standard Prolog calls allowed in logic grammars (which are also, after all, procedure calls interspersed within the rules).

In short, where DCGs accommodate context-sensitive constraints within user-controlled parameters, RGs enforce them through restrictions placed upon a system-controlled parse tree. This concept would result in a higher level formalism if it gave the user a fairly complete independence from parse tree concerns. However, efficient exploitation of XGs requires some knowledge of

the parse tree, and the user needs to express restrictions in the lower level terms of tree traversal rather than in the typically declarative, operationally independent fashion of logic grammars.

5. How evolved a grammar formalism do we need?

Work in theoretical linguistics has lately been departing from transformational theory (Chomsky 1965), largely because of the subtelty of rules involved and the supplementary devices needed (e.g. co-indexing, filters, etc.) and because of the complexity of dealing with semantics within the transformational panadigm. Work by Montague (Montague 1976) and Gazdar (Gazdar 1981) resulted in a simpler and more intuitive formalization of semantics based upon the rule-to-rule hypothesis (Bach 1976): to each syntactic rule corresponds a structually analogous, semantic rule for building up logical representations.

Gazdar's framework, in particular, can deal with a wide range of syntactic phenomena within a phrase-structure theory that has a node admissibility interpretation rather than a generative one. This new outlook, augmented by metagrammatical devices (such as categories with gaps, metarules and rule-schemata) elegantly captures such important constructs as coordination and unbounded dependencies. Gazdar's "augmented phrase structure" approach has influenced research in AI, where the transformational approach had also been losing adepts, as it was also felt to deal insufficiently with semantics and, moreover, with sentence analysis - AI's main concern in natural language processing.

Among the systems inspired by this approach are (Joshi 1982, Robinson 1982, Schubert & Pelletier 1982).

Logic grammars, from all our previous discussion, would seem to provide an adequate computational framework within which to implement the augmented phrase structure approach. From the descriptive point of view, as we have seen, "logical" context-free rules are more powerful than standard ones because of parameters in grammar symbols, and unification. Procedure calls are moreover an inherent feature that is useful for representing constraints.

But, although any logic grammar supports at least this, the user need not be restricted to context-free type rules. MGs or XGs will moreover provide for generalized type-0 rules, and even for the handling of gaps, while maintaining high standards of efficiency.

Extra power available, therefore, can only represent a gain, since it does not preclude resorting to more elementary approaches as a special case. In this respect we support Berwick and Weinberg's contention that there is a possible tradeoff between parsing efficiency and descriptive apparatus, and that "a language that is quite 'high up' in the Chomsky hierarchy - e.g. a strictly context-sensitive language - may in fact be parsed more rapidly than languages lower down in the hierarchy - e.g. faster than some context-free languages - if the gain in succinctness is enough to offset the possible increase in parsing

time" (Berwick & Weinberg, 1982).

Our approach is therefore that of continuing research on logic grammar extensions that may be useful in view of a more powerful and elegant, while still efficient, treatment of some natural language processing phenomena. In the next section we describe a new logic grammar formalism developed in particular for dealing metagrammatically with coordination, but that exhibits several other features that are interesting by themselves.

6. Modifier structure grammars (MSGs)

Research by Michael McCord (McCord 1980, 198 has resulted in interesting ideas for processin natural language through logic grammars. In particular, the notion of modifier structure and the treatment of semantic interpretation presente there seemed a promising framework for solving nontrivial language-processing problems, such as coordination. Joint research with Michael McCord in view of a logic grammar, metagrammatical treatment of coordination, resulted in the development of a system consisting of: a) a new formalism for logic grammars, which we call modifier structure grammars (MSGs), useful for making modifier structure implicit in the grammar, b) an interpreter (or parser) for MSGs which also takes all the responsibility for the syntactic aspects of coordination, and c) a semantic interpretation component which produces logical forms from the output of the parser and deals with scoping problems, (as in McCord 81) which also includes specific rules for semantic interpretation of coordination. The whole system is implemented in Prolog-10 (Pereira, Pereira & Warren 1978). Here we make a brief presentation of this system. A complete description can be found in (Dahl & McCord, 1983).

MSG rules are of the form:

A : Sem --> B

where A --> B is an XG rule and Sem is a term called a semantic item, which plays a role in the semantic interpretation of a phrase analysed by application of the rule. The semantic item is (as in (McCord 1981)) of the form

Operator - LogicalForm

where, roughly, LogicalForm is the part of the logical form of the sentence contributed by the rule, and Operator determines the way in which this partial structure combines with others. Sem may be a "trivial" Sem if nothing is contributed.

When a sentence is analysed, a structural representation, in tree form, called "modifier structure" is automatically formed by the parser. Each of its nodes contains not only syntactic information but also the semantic information Sem supplied in the grammar, which determines the node's contribution to the logical form of the sentence (this contribution is for the node alone, and does not refer to the daughters of the node, as in Gazdar's approach (Gazdar, 1981)).

The semantic interpretation component first reshapes this tree into another MS tree where the scoping of quantifiers is closer to the intended semantic relations than to the (surface) syntactic ones. It then takes the reshaped tree and translates it into logical form. The modifiers actually do their work of modification in this second stage, through their semantic items. It should be noted that the addition of simple semantic

indicators within grammar rules contributes to maintain, from the user's point of view, a simple correspondence between syntax and semantics. This is similar in intention to the rule-by-rule hypothesis mentioned before (Bach 1976), but is differently realized: instead of a rule-to-rule correspondence, we have a correspondence between each non-trivial expansion of a non-terminal and a logical operator. That is, each time the parser expands a non-terminal symbol into a (non-empty) body, a logical operator labels the expansion and will be later used by the semantic component, interacting with other logical operators found in the parse tree obtained. The complexity of dealing with quantifier scoping and its interaction with coordination is screened away from the user.

With respect to coordination, the MSG grammar should not mention conjunction at all. The interpreter has a general facility for treating certain words as "demons" (cf. Winograd 1972), which trigger a backing up in the parser history that will help reconstruct ellisions and recognize the meaning of the coordinated sentence.

This proceeds in a manner similar to that of the SYSCONJ facility for augmented transition networks (Woods 1973, Bates 1978), except that, unlike SYSCONJ, it can also handle embedded coordination and interactions with extraposition. The use of modifier structures and the associated semantic interpretation component, moreover, permits in general a good treatment of

scoping problems involving coordination. Finally, the system seems reasonably efficient (cf. timings for our sample grammar in (Dahl and McCord, 1983)).

7. Concluding remarks.

Logic grammars, as we have seen, need not sacrifice efficiency to the goals of power and elegance. They seem to be evolving - like other computational formalisms - into higher level tools which allow the user to spare mechanizable efforts in order to concentrate on as yet unmechanizable, creative tasks.

We view MSGs as a step in that direction, with the main advantages of automatising the treatment of coordination, providing a modular treatment of semantics, and allowing the user not to worry over structure building.

The latter feature may be an attractive one for logic grammars in general to retain, since it makes a grammar easier to write and read, and more concise.

Since, moreover, logical structure desired for a sentence's final representation is also automatically built up, from a few simple semantic indicators in the grammar rules, it becomes easier to adapt a grammar to alternative domains of application: modifying the logical representation obtained need only involve the semantic components of each rule.

This modular isolation of structure lends grammars a

syntactico-semantic flavour. It may be viewed as a way out of the dilemma on whether the semantic component should be separate or intermingled with the syntactic one. Compromising on manipulating static semantic indicators during the syntactic parse while using them dynamically during the semantic one may well prove to be the way of combining the advantages of both approaches while minimizing the disadvantages.

ACKNOWLEDGEMENT

This work was completed under NSERC Operating Grant 12436 and PRG Grant 6-4240.

REFERENCES

Bach, E. (1976). An extension of classical transformational grammar. Mimeo, Univ. of Massachusetts, Amherst, MA.

Bates, M. (1978). The theory and practice of augmented transition networks. In: L. Bolc (ed.) Natural Language Communication with Computers. (Springer, Berlin, May 1978).

Berwick, R. C. and Weinberg, A. S. (1982). Parsing Efficiency, Computational Complexity, and the Evaluation of Grammatical Theories. Linguistic Inquiry, vol. 13, No. 2, Spring 1982, pp. 165-191.

Chomsky, N. (1965). Aspects of the Theory of Syntax MIT Press, Cambridge, Massachusetts.

Coelho, H. M. F. (1979). A program conversing in Portuguese providing a library service. Ph.D. Thesis, Univ. of Edinburgh.

Colmerauer, A. (1973). Les systemes-Q ou un formalisme pour analyser et synthetiser des phrases sur ordination. Publication interne No 43, Dept. d'Informatique, Universite de Montreal.

Colmerauer, A. (1975). Les grammaires de metamorphose. Groupe d'Intelligence Artificielle, Univ. de Marseille-Luminy. As "Metamorphosis Grammars" in: L. Bolc (ed.), Natural Language Communication with Computers (Springer, Berlin, May 1978).

Dahl, V. (1977). Un systeme deductif d'interrogation de banques de donnees en espagnol. These de Doctorat de Specialite, Univ. d'Aix-Marseille.

Dahl, V. (1981). Translating Spanish into logic through logic. American Journal of Computational Linguistics, Vol. 7, No. 3, pp. 149-164.

Dahl, V. (1982). On database systems development through logic. ACM Transactions on Database Systems, Vol. 7, No. 1, March 1982, pp. 102-123.

Dahl, V. and McCord, M. (1983). Treating coordination in logic grammars. Internal report, Univ. of Kentucky.

Gazdar, G. (1981). Unbounded dependencies and coordinate structure. Linguistic Inquiry, Vol. 12, No. 2, Spring, 1981.

Hirshman, L. and Puder, K. (1982). Restriction grammar in Prolog. Proc. First International Logic Programming Conference, Marseille, pp.85-90.

Joshi, A. (1982). Phrase Structure Trees Bear More Fruit than You Would Have Thought. American Journal of Computational Linguistics. Vol.8, No.1, Jan-March 1982.

Kowalski, R. A. (1979). Logic for problem solving. North-Holland Elsevier, New York, 1979.

McCord, M. (1980). Using slots and modifiers in logic grammars for natural language. In: Artificial Intelligence.

McCord, M. (1981). Focalizers, the scoping problem, and semantic interpretation rules in logic grammars. University of Kentucky.

Montague, (1974). English as a formal language. In: Thomason, R. H. (ed.), Formal Philosophy: selected papers of Richard Montague. Yale Univ. Press, New Haven, CT., pp.188-221.

Pasero, R. (1982). A dialogue in natural language. Proc. First International Logic Programming Conference, Marseille, France, pp.231-239.

Pereira, F. (to appear). Extraposition grammars. To be published in: American Journal of computational linguistics.

Pereira, F. and Warren, D. (1980). Definite clause grammars for natural language analysis - a survey of the formalism and a comparison with augmented transition networks. Artifical Intelligence 13 (1980), pp.231-278.

Pereira, F. and Warren, D. (1981). An efficient easily adaptable system for interpreting natural language queries. Dept. of AI, Univ. of Edinburgh.

Pereira, L., Pereira, F. and Warren, D. (1978). User's guide to DECsystem-10 Prolog. Div. de Informatica, LNEC, Lisbon and Dept. of AI, Univ. of Edinburgh.

Pereira, L. M. et al. (1982). ORBI - An expert system for environmental resource evaluation through natural language. Universidade Nova de Lisboa.

Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. J. ACM 12, pp.25-41.

Robinson, J. (1982). Diagram: A grammar for dialogues. Comm. ACM, January 1982, Vol.25, No.1.

Sabatier, P. (1980). Dialogues en francais avec un ordinateur. Groupe d'Intelligence Artificielle, Univ. d'Aix-Marseille.

Silva, G. M. T. et al. (1979). A knowledge-based automated message understanding methodology for an advanced indications system. Rome Air Development Center, Report RADC-TR-79-133.

Simmons, R. F. and Chester, D. (1979). Relating sentences and semantic networks with clausal logic. Dept. of Computer Science, Univ. of Texas.

Schubert, L. and Pelletier, F. (1982). From English to Logic: Context-Free Computation of "Conventional" Logical Translation. American Journal of Computational Linguistics, Vol.8, No.1, Jan-March 1982.

Winograd, T. (1972). Understanding natural language. Cognitive Psychology, 3, pp.90-93.

Woods, W. A. (1970). Transition network grammars for natural language analysis. C. ACM 13.

Woods, W. A., Kaplan, R. M. and Nash-Webber, B. (1972). The lunar sciences natural language information system: final report, BBN Report 2378.