PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1984,
edited by ICOT. © ICOT, 1984

MORE ON GAPPING GRAMMARS

Veronica Dahl

Department of Computing Science
Simon Fraser University
Burnaby, B.C. V5A 1S6
CANADA

ABSTRACT

Gapping grammars (GGs) are logic grammars that may explicitly refer to gaps in between constituents —i.e., to unidentified intermediate substrings of symbols. A gap G is referred to through a special, system-defined symbol: gap(G), and can be rewritten into any position in the right hand side of a rule. GGs include as special cases three other types of logic grammars: definite clause grammars, metamorphosis grammars and extraposition grammars.

GG rules have the form:

$a, \alpha_0, gap(G_1), \alpha_1, gap(G_2), \ldots, gap(G_n), \alpha_n \longrightarrow \beta.$

with $\alpha_i \in (V_N \cup V_T)^*$, $a \in V_N$, and $\beta \in (V_N \cup V_T \cup \Gamma)^*$

where $\Gamma = \{gap(G_1), gap(G_2), \ldots, gap(G_n)\}$,
G_i are variables ranging over $(V_N \cup V_T)^*$,

and V_T and V_N are respectively the terminal and non terminal vocabularies of the gapping grammar.

This article discusses how GGs can be exploited to produce natural and concise formal language descriptions, and how they can powerfully help capture several linguistic phenomena: coordination, free word order and right extraposition.

1 INTRODUCTION

Since the development of the first logic grammar formalism by A.Colmerauer in 1975 (Colmerauer 1978), and of the first sizable application of logic grammars by the author in 1977 (Dahl 1981), several variants of logic grammars have been proposed, sometimes motivated by ease of implementation (Definite Clause Grammars, DCG, (Pereira and Warren 1980)), sometimes by a need of more general rules with more expressive power (Extraposition Grammars, XGs, (F.Pereira 1981)), sometimes with a view towards a general treatment of some language processing problem such as coordination (Modifier Structure Grammars, MSGs, (Dahl and McCord 1983)) or of automating some part of the grammar writing process, such as the automatic construction of parse trees and internal representations (MSGs, op. cit, DCTGs, (Abramson 1984)). Generality and expressive power seem to have been the main concerns underlying all these efforts.

Gapping grammars (GGs) are logic grammars meant for analysis, whose rules may explicitly refer to gaps — i.e., to unidentified intermediate substrings. They were designed by the author, as a generalization of extraposition grammars, and further investigated in joint work with Harvey Abramson (Dahl and Abramson 1984), with emphasis on implementation details.

Here we examine some of the problems we feel them suited for, both for formal and for natural language processing. In particular, we propose a GG treatement of free word order. We argue that, although implementation details are not definitely settled (the current compiler being somewhat inefficient), the potential uses of GGs are worth studying.

Section 2 describes the gapping grammar formalism. Section 3 examines a small gapping grammar for natural language as an illustration on how to think in terms of gaps. Section 4 shows how, in some cases, GGs allow us to avoid artifices like adding extra grammar symbols that do not stand for constituents but serve some procedural-oriented goal, like counting symbols. Section 5 examines how the expressive power of GGs can be used to not only obtain more concise but also more efficient formulations than in other grammar formalisms. Efficiency here is considered with respect to the number of backtracks upon user-defined grammar symbols in our compiler formulation. Successive grammars for the same sample language are discussed. Section 6 proposes a GG formulation for representing free word order rules without multiplying the number of rules, and section 7 presents a small English GG with left and right extraposition. Some knowledge of Prolog is assumed.

2 DESCRIPTION

GGs make use of a system-defined, non-terminal grammar symbol, gap(G), to refer to a substring G whose actual composition is of no present interest, but that stands in between grammar constituents we are interested in. Other than for the fact that this special symbol is not user-defined, GGs look very much like metamorphosis grammars (Colmerauer, 1978): a GG rule has the form

$$a,\alpha \longrightarrow \beta.$$

where 'a' is a non-terminal grammar symbol (different from 'gap(G)'), and α and β consist of terminals, non-terminals, gaps and Prolog calls.

For instance, the rule

```
nominative, gap(G) -->
     gap (G), [W] , {nom(W)}.
```

expresses that a nominative constituent followed by some gap G can be rewritten into the gap followed by a word that satisfies the 'nom' property. We adopt the DEC-10 convention that terminals are enclosed in square brackets, and Prolog calls, in brackets. Variables start with a capital letter.

We can now define gapping grammars more formally. Let F be a set of functional symbols, including the unary symbol "gap", and V an enumerable set of variables. Let $\mathcal{A}$ be the set of terms constructible from F and V, i.e., the set of formulas consisting of either a variable, a constant (i.e., a functional expression of 0-arity) or an expression $f(t_1,\ldots t_n)$, where f is a functional symbol of arity n and the t_i are terms. Let H be the set of ground terms (i.e., terms containing no variables) in $\mathcal{A}$. Let $V_T \subset \mathcal{A}$, called the terminal vocabulary. Let $V_N \subset \mathcal{A}$, called the non-terminal vocabulary, $V_T \cap V_N = \phi$. Let $\Gamma \subset V_N$, called the set of gap symbols, be a set of the form: $\{gap(G_1), gap(G_2), \ldots, gap(G_n)\}$, where the G_i are variables. Let $V_S \subset V_N$ be the set of start symbols. Let P be a set of production (meta)-rules of the form:

$$nt,\alpha_0,gap(G_1),\alpha_1,\ldots,\\ gap(G_n),\alpha_n \longrightarrow \beta_0,gap\\ (G_{i_1}),\beta_1,\ldots,gap(G_{i_m}),\beta_m$$

where $m,n \geq 0, nt \in V_N; \alpha_i, \beta_i \in (V_N \cup V_T)^*$, $gap(G_i) \in \Gamma$, and for all j, $1 \leq i_j \leq n$. Let ==> be a rewriting relation on $(V_N \cup V_T)^*$, defined as follows:

u ==> v <==> there exists a production rule

$$\alpha_0,gap(G_1),\alpha_1,\ldots,\\ gap(G_n),\alpha_n \longrightarrow \beta_0,gap(G_{i_1}),\\ \beta_1,\ldots,gap(G_{i_m}),\beta_m$$

and some substitution θ of terms for variables, such that

$$u\theta = (\alpha_0 \ \gamma_1 \ \alpha_1 \ldots \gamma \ \alpha_m)\theta \text{ for some } \gamma_i \in (V_N \cup V_T)^*$$
$$v = (\beta_0 \ \gamma_{i_1} \ \beta_1 \ \ldots \ \gamma_{i_m} \ \beta_m)\theta$$

The quintuple $G=(V_N, V_T, \Gamma, V_S, P)$ is called a gapping grammar.

We now define $\overset{*}{==>}$ to be the reflexive, transitive closure of ==>. The language generated by G is

$$L(G)=\{t \in V^* / \exists S \in V_S \text{ with } S \overset{*}{==>} t\}$$

Notice that each of the metarules in

P stand for all instances of it — i.e., for all rules obtained from the metarule by substituting terms for variables.

Although terminal symbols may contain variables, in practice grammars are usually written such that they describe <u>ground</u> strings $t \in V_T^*$, i.e., strings $t \in V_T^* \cap H$. Also notice that we have left Prolog calls out of the formal definition, in order to keep it simpler. Prolog calls appearing in a metarule also become affected by the substitution used when applying the metarule, and become executed during this application. Their failure results in that rule's application being blocked.

Notice that XGs are a special case of GGs, where all gaps mentioned in the left-hand side are rewritten in sequential order at the end of the right-hand side. Also, XGs allow no Prolog calls in the left-hand side.

As in metamorphosis grammars, which are also included in GGs, for each non-terminal 'nt' contained in the left-hand side of a rule, we need a "normalizing" rule of the form

```
   nt --> [nt] .
```

where '[nt]' is a pseudo-terminal that does not occur in the original grammar. As these rules can be constructed automatically by a grammar preprocessor, we shall consider them transparent to the user, and disregard them in all that follows.

3 THINKING IN TERMS OF GAPS

A gap may be thought of as a substring in the sentence to be analysed, that is separated and repositioned unanalysed, to be later parsed in its new location. Since a grammar relates terminal strings to strings of constituents, we can also imagine a gap as a string of constituents to be repositioned by application of the gapping rule.[1]

[1]Normalization, mentioned in the previous section, keeps gaps as strings of either terminals or pseudo-terminals, but, conceptually, we can disregard the normalization rules and refer to strings of constituents.

We next illustrate this through a simple grammar that handles sentence coordination and reconstitutes the meaning representation of an elided object.

```
sentence(and(S1,S2)) -->
    sent(S1),and,sent(S2).

sent(S) --> name(K),
    verb(K,P,S),object(P).

object(P) --> determiner
    (X,P1,P),noun(X,P1).
object(P) --> [nt-object(P)].
object(P),and,gap(G),
    object(P) --> [and] ,
    gap(G),object(P).

determiner(X,P,the(X,P)) -->
    [the].

noun(X,train(X)) --> [train].

   name(mary) --> [mary].
   name(john) --> [john].

   verb(X,Y,saw(X,Y)) -->
    [saw].
verb(X,Y,heard(X,Y)) -->
    [heard].
```

The third rule for 'object' elides an expected object followed by a gap and reconstructs its internal representation 'P' through unification with the meaning representation of the object in the second sentence. The gap in between 'and' and the object is recopied unanalysed.

A derivation graph for the sentence 'mary saw and john heard the train' might help visualize the working of the grammar, shown in fig. 1.

We have labelled each rule application with the substitutions used, and circled the gap. The final value obtained is:

```
S=and(saw(mary,the(X,train(X))),
    heard(john,the(X,train(X))))
```

Specific rules for coordination were, in fact, my motivation for developing GGs. Subsequent work, jointly with Michael McCord, resulted in a very specialized type of grammars — modifier structure grammars — that handle coordination

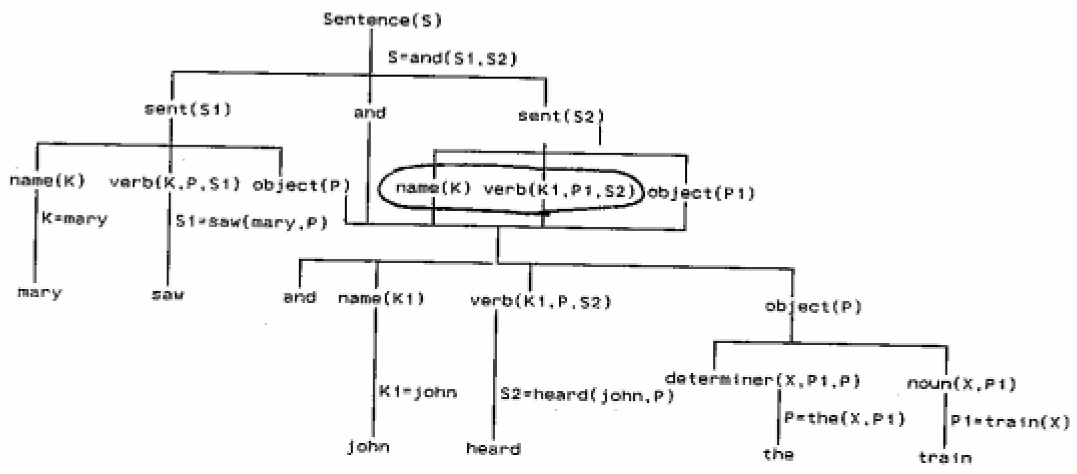

Figure 1.

in a more general, metagrammatical way (Dahl and McCord, 1983). Still, where specific rules are concerned, coordination is a good example of the need for repositioning gaps, as the elided fragments may not be full constituents, but be instead made up from fragments of different constituents.

4 AVOIDING ARTIFICES THROUGH STRAIGHTFORWARD USES OF GAPS

In (Dahl and Abramson, 1984), we presented the following GG for the language $\{a^n b^n c^n\}$:

```
s --> as,bs,cs.

as --> [ ].
as --> xa,[a],as.

bs --> [ ].
xa,gap(G),bs -->
     gap(G),[b],bs,xb.

xb,gap(G),cs --> gap(G),[c],cs.
cs --> [ ].
```

This grammar uses marker symbols 'xa' and 'xb', whose only function is to leave traces of right-extraposed a's and b's, in places where they can easily be evened out with b's and c's, respectively.

This grammar was presented as an argument for freer repositioning of gaps than in XGs, and contrasted with the corresponding XG, which uses markers for left-extraposed symbols. Our GG formulation was found to work more efficiently than the XG one, with respect to the compiler implementation.

However, thinking in terms of markers is in many cases simply a residue from the constraints imposed by less powerful grammars. This language can in fact be simply described as follows:

```
1) s --> as,bs,cs.

2) as,gap(G1),bs,gap(G2),cs -->
      [a],as,gap(G1),[b],bs,
      gap(G2),[c],cs.

3) as,gap(G1),bs,gap(G2),cs -->
      gap(G1),gap(G2).
```

Rule (2) simply evens a's, b's, c's with a, b, and c, by skipping any intermediate strings as gaps G1 and G2, which are repositioned after regenerating as, bs, and cs in the right-hand side. Rule (3) simply makes as,bs and cs vanish by only recopying the gaps found in between.

Similarly, the language $\{a^n b^m c^n d^m\}$, for which we showed a GG formulation with markers in (Dahl and Abramson, 1984) can be more easily formulated as follows:

```
s--> as, bs, cs, ds.
```

```
as,gap(G),cs—>
    [a],as,gap(G),[c],cs.
as,gap(G),cs—> gap(G).

bs,gap(G),ds—> [b],
    bs,gap(G),[d],ds.
bs,gap(G),ds—> gap(G).
```

5 MORE EXPRESSIVE POWER NEED NOT MEAN LESS EFFICIENCY

In our compiler implementation of GGs, too much backtracking results due to the fact that the 'gap(G)' predicate is merely a version of 'append', which breaks the input string into a gap G followed by a remainder, in a non-deterministic way. After this experimental stage, we hope to present a more efficient implementation of gap determination, which does not sacrifice the conciseness of our compiler. In this sense, we are presently examining ways of modifying the strategy for executing the compiler's output, after having ruled out several other options.

For the time being, let us assume that the gap determination problem has been solved in the most efficient manner possible, and let us examine how having more expressive power affects the performance of the user-defined grammar constituents.

Take for instance the "scrambled $a^n b^n c^n$" problem, where sentences consist of an equal number of a's, b's, and c's, but mixed up in any order. In the first approximation to this problem, the power of XGs suffices. The idea is to write rules of the form:

```
as,gap(G),bs —> [b],as,gap(G).
```

which might be read as: if a 'b' appears instead of an expected 'a' at the beginning of the input string, skip any intermediate constituents until reaching a 'bs' symbol that does look for b's, and even them out, while recopying the unsatisfied 'as' and the unexamined gap.

The complete grammar follows.

```
s —>as,bs,cs,s.
s —> [ ].

as —>[a].
as,gap(G),bs —> [b],as,gap(G).
as,gap(G),cs —> [c],as,gap(G).

bs —>[b].
bs,gap(G),cs —> [c],bs,gap(G).
bs,gap(G),as —> [a],bs,gap(G).

cs —> [c].
cs,gap(G),as —> [a],cs,gap(G).

cs,gap(G),bs —> [b],cs,gap(G).
```

All these gapping rules rewrite gaps into rightmost positions, so they are strictly XG rules. Without leaving the XG formalism, we might collapse the three sets of rules for 'as, 'bs', and 'cs' (which are symmetrical) into just three rules plus the artifice of adding 'start' and 'end' symbols in order to keep in good terms with syntax. The rationale here is: upon hitting an 'a' as starting input symbol, skip any constituents until you reach the 'as' to even the 'a' out with, and then rewrite the skipped gap. If syntax permitted, we would write:

```
gap(G),as —> [a],gap(G).
```

But, since rules should start with a non-terminal symbol other than gap(G), the complete grammar reads:

```
s —> start,sl,end.

sl —>as,bs,cs,sl.
sl —> [ ].

start,gap(G),as —> [a],
    start,gap(G).
start,gap(G),bs —> [b],
    start,gap(G).
start,gap(G),cs —> [c],
    start,gap(G).
start,end —> [ ].
```

While in the first formulation, scrambled sequences of m letters (in the example, m=3) require m^m rules plus the two rules for 's', in the second one we get away with just m rules plus the four rules for 's', 'sl', and 'end'. Alas, while the first formulation parses strings in the language without ever backtracking upon 'as', 'bs', or 'cs', the second one backtracks on the 'start' predicate upon all possible string rearrangements, until it finally hits the good one and succeeds.

If we use the full power of GGs, however, we can arrive at the exceedingly simple formulation:

```
   s --> as,bs,cs,s.
   s --> [ ].

   as,gap(G) --> gap(G),[a].
   bs,gap(G) --> gap(G),[b].
   cs,gap(G) --> gap(G(,[c].
```

which not only makes the number of rules linearly rather than exponentially dependent on m without any artifice like 'start' and 'end', but also happens to head directly to the solution without even once backtracking on any user-defined grammar symbols. Rearranging gaps in an arbitrary fashion allows us a very efficient, lexicon-- driven formulation: when trying to expand 'as', simply skip any intermediate symbols until you reach an 'a', and retain the gap for futher analysis.

6 GGs AND FREE WORD ORDER

The above suggests a very concise and efficient treatment of free word order, a problem that many languages exhibit to some extent. The problem is that, since inflections rather than position are used to indicate case or grammatical function, position is used to indicate emphasis or focus, and almost any possible ordering becomes acceptable. For instance, the Sanscript phrase 'Ramauh pashyati Seetam' (Ramauh sees Seetam) can also appear as:

```
   pashyati Ramauh Seetam
   pashyati Seetam Ramauh
   Seetam Ramauh pashyati
   Seetam pashyati Ramauh
   Ramauh Seetam pashyati
```

This kind of free order of sister constituents where each retains its integrity is easily handled within gapping grammars, in a similar way as in our last example (e.g. sample rule in Section 2). More interesting is the case in which even the contents of constituents appear to be scrambled up with elements from other constituents (e.g. as in the Warlpiri language). Even in Latin or Greek, phenomena such as discontinuous noun phrases, which would appear as extreme dislocation in prose, are very common in verse (and not unusual even in certain prose genres (e.g. Plato's late work, such as the Laws). A contrived example for Latin would be:

```
   Puella bona puerum parvum amat.
       (Good girl loves small boy)
```

where the noun and adjective in the subject and or object noun phrase may be discontinued, e.g.:

```
   Puella puerum amat bona parvum.
```

In fact all 5! word permutations are possible, and we certainly do not want to write a separate rule for each possible ordering. In GGs, we can simply write:

```
   sentence --> noun-phrase(nom),
       noun-phrase(acc), verb.

   noun-phrase(Case) -->
       adjective(Case), noun(Case).

   noun(Case), gap(G) --> gap(G),
       [Word],
       {dict(noun(Case),Word)}.

   adjective(Case),gap(G) --> gap(G),
       [Word],{dict(adjective
       (Case), Word)}.

   verb,gap(G) --> gap(G),
       [Word], {dict(verb,Word)}.

    dict(verb, amet).
    dict(noun(acc), puerum).
    dict(noun(nom), puella).
    dict(adjective(acc), parvum).
    dict(adjective(nom), bona).
```

Another approach to free word order is the augmented phrase structure one [Pullum, 1982]. In Pullum's formulation, phrase structure (meta) rules only indicate immediate dominance, and are supplemented with linear precedence restrictions to indicate what orderings are allowed. For instance, the meta rule A --> B,C,D, together with on empty set of linear precedence restrictions, stands for all rules where A rewrites into B, C and D in any order. With the restriction: {D C}, on the other hand, it represents only the rules: {A --> BDC, A --> DBC, A --> DCB}.

While this notation is concise and expressive for free or relatively free ordering problems, it becomes costly as more orderings are fixed. Also, since precedence restrictions are attached to the whole set of phrase structure rules, it can only deal with grammars in which any two constituents that have been stated to have an order appear in that order no matter what their origin. Gazdar and Pullum make the hypothesis that grammars of natural language will all possess this property.

GGs, on the other hand, can describe different orders of same constituents coming from different rules quite straightforwardly, so they will be appropriate even if the above mentioned hypothesis turns out to be wrong, or if we want to process formal language grammars that do not satisfy it. They also seem more versatile in being able to deal with both fixed and changing order with no significant change in cost.

Another difference is that GG rules, like all logic grammar rules, may stand for infinite sets of rules. It suffices for them to contain a variable, and all rules generated from them by substituting a term for the varible are represented.

If the variable in question happens to be a gap argument, moreover, all rule instances where the gap has been replaced with any parsing substate become represented.

In this sense, GGs are also more powerful than the augmented phrase structure approach. It is important to realize what kind of a meta-grammar formalism the GG one is. It is not a device that will generate all specific rules for a grammar before the actual parse. It will process the GG rules written by the user, unmodified, and discover, during parsing, only those rule instances that are relevant to the particular sentence being analysed. Virtual rules are thus made present on demand through unification, while all that is actually stored is the concise, gapping formulation of the grammar.

6.1 Lexicon-Driven Free Word Order with Options

Another interesting problem that can be handled in GGs is free order of constituents that may or may not be present, as determined by other constituents. Let us assume that each particular verb requires its own set of constituents. Then we could include an argument in the verb symbol, telling us about its specific requirements. Modulo notation, this would look like:

```
sentence --> verb(R),R.

verb(R),gap(G) -->
    gap(G),[W],{ver(W,R)}.

nominative,gap(G) -->
    gap(G),[W],{nom(W)}.

accusative,gap(G) -->
    gap(G),[W],{acc(W)}.

ver(pashyati,[nominative,
    accusative]).

nom(ramauh).

acc(seetam).
```

These examples, although oversimplified, suggest that GGs might be an appropriate computational tool for processing lexical functional grammars (Kaplan and Bresnan, 1981).

7 GGs AND RIGHT EXTRAPOSITION

In Section 4, we have argued that we should not only be allowed right-extraposing formulations of a grammar as a matter of personal preference or efficiency considerations, but we should also be given the liberty to reposition the gaps referred to arbitrarily, in order to do away with artifices, like adding extra grammar symbols that do not stand for constituents, but serve some procedural— oriented goal, like counting symbols.

A further argument for expressing right-extraposition directly is the fact that, in natural language, some movement phenomena are more naturally viewed as right rather than left-extraposition, although they could perhaps be forced into left-extraposing formulations.

We next show a small English grammar with a relativization rule expressed as left-extraposition and another rule for extraposing the whole relative clause to the right.

Both rules can interact in the same sentence, as the example shows.

Notice that, despite our advocacy of the non-addition of extra symbols, in this example we are using symbols such as 'trace'. We do not, however, consider these an artifice, because, as opposed to the markers 'xa', and 'xb' in Section 4, traces have linguistic reality in natural language and can be viewed as constituents in their own right, as part of the grammar, outside any considerations on parsing. The following grammar parses sentences such as 'The man is here that Jill saw' into logical structures such as:

```
the(X,and(man(X),saw(Jill,X)),
    here(X))


sentence(P) -->
    np(X,P1,P),vp(X,P1).

np(X,P1,P) -->
    det(X,P2,P1,P),noun(X,P3),
    relative(X,P3,P2).

np(X,P,P) --> name(X).

vp(X,P) --> trans-verb
    (X,Y,P1),object(Y,P1,P).
vp(X,P) -->
    aux(be),comp(X,P1,P).

relative(X,P1,and(P1,P2)) -->
    rel-marker(X),sentence(P2).
relative(X,P1,P),gap(G) -->
    gap(G),rightex(X,P1,P).
relative(X,P,P) -->[ ].

rel-marker(X),gap(G),
    trace(X,P1,P) -->
    rel-pronoun,gap(G).

object(X,P,Q) --> np(X,P,Q).
object(X,P,P) --> trace(X,P,P).

comp(X,P,P) --> adverb(X,P).

trace(X,P1,P) -->
    [trace(X,P1,P)].

rightex(X,P1,and(P1,P2)) -->

    rel-marker(X),sentence(P2).

noun(X,man(X)) --> [man].

aux(be) --> [is].

adverb(X,here(X)) --> [here].

det(X,P1,P2,the(X,P1,P2,)) -->
    [the].

rel-pronoun --> [that].

name(jill) --> [jill].

trans-verb(X,Y,saw(X,Y,)) -->
    [saw].
```

8 CONCLUDING REMARKS

We have examined several formal and natural language parsing utilizations of GGs, and shown how GGs can be exploited to arrive at concise formulations that (gap determination excluded) are also largely deterministic.

As pointed out in (Dahl and Abramson, 1984), this power can be misused if not kept in mind. We there showed a naive GG to accept expressions balanced with respect to parenthesis, which happened to also accept unbalanced strings, by way of eating up the unbalanced parenthesis into the gaps. As has also been pointed out, implementation details are not definitely settled.

However, as the most inclusive type of rewriting[1] logical grammar to date, GGs seem to hold promise in flexibility and expressive power. Efficiency in gap determination, moreover, does not seem crucial to us, since the most obvious applications for GGs concern natural language processing, and will therefore deal with relatively small substrings anyway (the largest possible being the whole sentence or paragraph).

Some work on the complexity of GGs could be useful at this point, but this should probably best be studied once implementation issues have become better defined, and, insofar as GGs are a computational tool rather than a linguistically exhaustive theory, maybe their complexity should be studied in

----

[1] Recent work by Paul Sabatier (Sabatier, 1984) presents a non-rewriting type of logic grammar — puzzle grammars —based on tree-assembling. Here we only refer to logic grammars based on rewriting rules.

connection with the specific grammars they will be used for in actual systems — e.g., if used for lexical function grammars, complexity results (e.g. Berwick 1982) might carry over.

All examples chosen here have been kept simple in order to provide clear illustrations for our discussion. We hope, however, that they have covered enough interesting concepts that they might induce further research on GGs within wider scale applications.

ACKNOWLEDGEMENTS

I am grateful to Brian Newton, Martti Penttonen and the referees for useful comments on this article's first draft. I also appreciate the assistance of Wilson Leung and Fred Popowich with some of the testing.

This work was supported by the National Science and Engineering Research Council of Canada.

REFERENCES

Abramson, H. Definite Clause Translation Grammars. Proc. IEEE Logic Programming Symposium, Atlantic City, New Jersey, 1984.

Berwick, R.C. Computational Complexity and Lexical— Functional Grammars. American Journal of Computational Linguistics, Vol 8, No. 3-4, 1982.

Colmerauer, A. Metamorphosis Grammars. In:Bolc, ed., Natural Language communication with computers. Springer-Verlag, 1978.

Dahl, V. Translating Spanish into logic through logic. American Journal of Computational Linguistics, Vol. 13, pp. 149-164, 1981.

Dahl, V., McCord, M. Treating coordination in logic grammars. American Journal of Computational Linguistics, Vol. 9, No. 2, pp. 69-91, 1983.

Dahl, V., Abramson, H. On gapping grammars. Proc. Second International Conference on Logic Programming, pp. 77-88. Uppsala, Sweden, 1984.

Kaplan, R., Bresnan, J. Lexical-functional grammar: a formal system for grammatical representation. MIT Center for Cognitive Science Occasional paper No. 13, Cambridge, MA., 1981.

Pereira, F.C.N., Warren D.H.D. Definite Clause Grammars for Language Analysis. Artificial Intelligence, Vol. 13, pp. 231-278, 1980.

Pereira, F.C.N., Extraposition grammars. American Journal of Computational Linguistics 7, 4, pp. 243-256, 1981.

Pullum, G.K. Free word order and phrase structure rules. In: J. Pustejovsky and P. Sells, eds., Proc. of the Twelfth Annual Meeting of the North Eastern Linguistic Society, pp. 209-220., 1982.

Sabatier, P. Puzzle grammars. In: Proc. First International Workshop on Natural Language and Logic Programming, Rennes, France, 1984.