Practical Parsing of Generalized Phrase Structure Grammars

Anthony J. Fisher

Department of Computer Science
The University of York
Heslington, York YO1 5DD, U.K.

An efficient algorithm is described for parsing a dialect of generalized phrase structure grammar (GPSG). A practical parsing system, based on the algorithm, is presented. The dialect of GPSG which the parsing system accepts is smaller, but considerably "purer" (closer to the original definition of GPSG) and mathematically "cleaner" than that which is accepted by other practical parsing systems. In particular, the parsing system correctly implements feature co-occurrence restrictions, subject only to the restriction that the FCR set can be expressed in clausal form as a set of Horn clauses.

1 Introduction

The generalized phrase structure grammar (GPSG) (Gazdar et al. 1985) is one of the most recent, and currently one of the most popular, formalisms used by linguists to describe the syntax of natural (human) languages. A GPSG is basically a context-free grammar (CFG), whose non-terminals are complex symbols called **categories**. A category is a set of **features**. The CFG is augmented by:

- a set of conventions or constraints that govern the automatic "propagation" of features between categories on different nodes of the parse tree; and
- a set of propositions (Boolean formulas whose literals denote the presence of a feature in a category) that are required to hold for the category on each node of the parse tree. These propositions are known as **feature co-occurrence restrictions** (FCRs).

A GPSG also contains **feature specification defaults** (FSDs). An FSD behaves like an FCR in all respects save one: if an FSD, when taken in conjunction with the set of FCRs, cannot by the addition of features be made to hold on a category c on a given node, the FSD is simply ignored. Although this sounds straightforward, the precise effect of FSDs is most unclear. The original definition attempts to explain the effect of FSDs mainly by giving examples, although there are a few mathematical definitions that, however, appear to confuse the definition rather than to clarify it. A clear, formal definition of the effect of FSDs is urgently required. Because the present definition is so obscure, it was reluctantly decided to exclude FSDs from consideration in this paper.

Because a GPSG is so closely related to a CFG, it was thought that the well-known efficient parsing techniques for CFGs could be applied, with modifications, to GPSGs, and that GPSGs would therefore be computationally tractable. Recently, however, Ristad (1985) has shown that this is not the case, and that the unrestricted GPSG parsing problem is NP-complete (on the total problem size, viz. grammar plus input sentence). Even before Ristad's result was known, workers in this field had found the practical problems caused by the interaction of FCRs, FSDs, and the propagation conventions difficult to surmount. Briscoe's comments (1986) are typical: "Finally, the concept of privileged feature, its interaction with feature specification defaults and the bi-directionality of the head feature convention are all so complex that it is debatable how much use they would be in a practical system (even if we did manage to implement them)" (p. 1); "The interaction of feature co-occurrence restrictions, feature specification defaults and feature propagation proved very hard to implement/understand" (p. 2).

This paper does not address the issues of ID/LP parsing and metarules. Barton (1985) has shown that the ID/LP parsing problem is NP-complete (on the total problem size). He argued that a previous result of Shieber (1983), which purported to give a G^2 parsing method for ID/LP grammars, was incorrect. Barton claimed that Shieber's algorithm is exponential in the worst case. Barton's result alone might be considered a

0362-613X/89/010139–148$03.00

strong hint that ID/LP parsing is inappropriate, both as a practical computer-based parsing method and as a possible model for the way in which people parse sentences. It is of course probable that, for many grammars, a sufficiently efficient implementation of ID/LP could be obtained, either by means of a preprocessor or by using Shieber's algorithm, but this has not been attempted in the present implementation. The omission of ID/LP does not appear seriously to limit the usefulness of the parsing system, at least when applied to grammars of English.

The question of metarules deserves a less cursory discussion. It is known (Gazdar et al. 1985:65–67) that the addition of metarules to a GPSG does not alter the CF properties of the grammar. The **finite closure rule**, which (stated informally) prohibits a metarule from reprocessing its own output, implies that the addition of a single metarule to a GPSG can generate only a finite number of phrase structure (PS) rules for each PS rule in the grammar. However, it is easy to write a metarule that will match every PS rule in the grammar and generate, say, two output rules. A second metarule can then be written which does the same. It is clear, therefore, that in the worst case the size of the induced set of PS rules grows exponentially with the number of metarules. For reasons put forward by Thompson (1982), most, if not all, GPSG parsing systems handle metarules by employing a precompilation phase to generate the induced set of PS rules. If this method is employed, the parsing time will, in the worst case, grow exponentially with the number of metarules. In practice, however, exponential behaviour can safely be presumed to be rare, since a typical metarule "triggers" on only a small number of rules and generates only a small number of rules. Preliminary investigations with a "real" grammar of English suggest that the precompilation phase generates an output grammar (containing only PS rules) that is about twice as big as the input grammar (which contains metarules and PS rules).

The metarules of GPSG are related to the hyperrules of a van Wijngaarden grammar (VWG) (van Wijngaarden et al. 1976), as has often been observed (see, e.g., Gazdar et al. 1985:65). There exist various direct methods of parsing VWGs, which do not rely on the prior expansion of hyperrules to generate a (usually large) set of PS rules. Wegner's method (1980) and Fisher's method (1985) are, unfortunately, exponential in the worst case; however, the exponential behaviour of both algorithms stems from the fact that the finite closure property of GPSGs does not apply to VWGs, and the set of induced PS rules might be infinite. It is possible that these algorithms could be modified to handle the very restricted metarules of GPSG in polynomial time.

Besides the omission of FSDs, ID/LP, and metarules, there is one other respect in which the type of grammar under consideration differs from GPSG as originally described. For formal purposes, the original set of feature-propagation conventions seemed rather baroque and unduly specialised. It was felt that it would be preferable to substitute for the "head feature convention", the "foot feature principle" and the "control agreement principle" of the original definition a more general mechanism. Consequently, we assume throughout most of this paper that features may be specified simply as "percolating" (from a node to its mother) or "trickling" (from a node to its daughter). The propagation convention (percolating or trickling or neither or both) can be specified individually for each feature. It is stressed that this simplification is introduced in order to simplify the description of the formalism and of the algorithm. In section 4, the restrictions are relaxed, and it is shown how conventions that are closely related to the HFC, FFP, and CAP of "standard" GPSG (or **GPSG 85**, as it is known) can be accommodated.

Having removed from consideration ID/LP parsing, metarules, and FSDs, and having simplified (temporarily) the feature propagation conventions, we are left with a much smaller and more manageable formalism. The main thesis of this paper is that, given these restrictions, the simple requirement that the FCRs be expressible as a set of Horn clauses is sufficient to ensure parsability in time order $pK^2G^2n^3$, where p is a measure of the degree of ambiguity of the grammar, K is the size of the alphabet of features, G is a measure of the size of the grammar, and n is the length of the sentence. The exact meanings of p and G are made formal later. It should be pointed out (lest false hopes be raised) that p is not in general independent of n, and for some grammars p is in fact a worse-than-polynomial function of n, making the algorithm worse-than-polynomial on n in the worst case. In practice, however, the algorithm can be implemented quite efficiently in such a way that $p \sim n$ for many linguistically plausible grammars.

Previous implementations. There have been several previous attempts to parse GPSGs. Several workers have made wholesale changes to the definition of GPSG in order to make the formalism easier to parse. The dialect that the present algorithm accepts is considered to be "purer" and nearer to the original than that accepted by most other algorithms. A brief summary of GPSG implementations is given by Gazdar (1983). The "official" definition of GPSG has changed since the list was published, and some of the implementations listed are no longer available. A more recent implementation is that of Harrison and Maxwell (1986).

2 Definitions

Preliminary definitions. The following definitions, of standard terms of formal language theory, are given in, for example, Salomaa (1973).

An *alphabet* is a finite non-empty set. The elements of an alphabet are called *letters*. A *word* over an alphabet V is a finite string consisting of zero or more letters of V, whereby the same letter may occur

several times. The set of all words over an alphabet V is denoted by $W(V)$. For any V, $W(V)$ is infinite.

We denote by $\Pi(S)$ the *powerset* of S, which is the set of all subsets of a set S.

Definitions. A **generalised phrase structure grammar** (GPSG) is an ordered 7-tuple $G = (V_F, V_T, X_0, R, F, F_P, F_T)$, where:

V_F is a finite set of *features*;

V_T is a finite set of *terminals*, $V_F \cap V_T = \emptyset$;

X_0 is the *starting category*, a finite subset of V_F;

R is the set of *rules*, a finite set of ordered pairs $P \rightarrow Q$, such that P is a subset of V_F (i.e. $P \in \Pi(V_F)$), and Q is a word over the alphabet $V = \Pi(V_F) \cup V_T$;

F is the *FCR set*, a function from $\Pi(V_F)$ to {true, false};

F_P is the set of *percolating features*, a subset of V_F; and

F_T is the set of *trickling features*, a subset of V_F.

For $P, Q \in W(V)$, we say that P *derives* Q, written $P \Rightarrow Q$, iff $\exists$ an integer n and $\alpha, \beta \in W(V)$, P', Q'_i $(i = 1, \ldots, n)$, P'', Q''_i $(i = 1, \ldots, n) \in V$ such that the following all hold:

1. $\alpha P' \beta = P$ and $\alpha Q'_1 Q'_2 \ldots Q'_n \beta = Q$
2. $P'' \rightarrow Q''_1 Q''_2 \ldots Q''_n \in R$
3. [Extension:]
   (i) $P'' \subseteq P'$
   (ii) if $Q'_i \in V_T$: $Q''_i = Q'_i$ $(i = 1, 2, \ldots, n)$
   if $Q'_i \in \Pi(V_F)$: $Q''_i \subseteq Q'_i$ $(i = 1, 2, \ldots, n)$
4. [FCR constraints:]
   $F(P')$
5. [Propagation constraints:]
   (i) $Q'_i \in V_T \vee (Q'_i \cap F_P) \subseteq P'$ $(i = 1, 2, \ldots, n)$
   (ii) $Q'_i \in V_T \vee (P' \cap F_T) \subseteq Q'_i$ $(i = 1, 2, \ldots, n)$.

We denote by $\Rightarrow^*$ the reflexive and transitive closure of $\Rightarrow$.

The *language generated by* G, written $L(G)$, is defined by

$$L(G) = \{P \mid P \in W(V_T), X_0 \Rightarrow^* P\}.$$

End of definitions.

Informally, the definitions of GPSG, $\Rightarrow$, $\Rightarrow^*$ and $L(G)$ are the standard definitions of a context-free grammar, modified by defining the non-terminal of the standard CFG definition as a set of features. Furthermore, the standard definition of $\Rightarrow$ has been extended to take into account feature matching by the free addition of features to categories specified in rules (extension), the FCR constraints, and the propagation constraints. The original definition of GPSGs postulated a set of FCRs whose conjunction is required to hold; this conjunction has been collapsed into a single Boolean function F in our definition. F is required to hold on each non-terminal node of the parse tree by virtue of condition 4 above. (It is unnecessary to specify that $F(Q'_i)$ must hold; this is implied by the use of the reflexive and transitive closure of $\Rightarrow$ in the definition of $L(G)$.) Finally, condition 5(i) requires that, if P' is the mother of a non-terminal node Q'_i (considering P' and Q'_i as nodes in a parse tree), then for each feature f which has been defined as percolating from daughter to mother (i.e. is a member of F_P), if f is present on the daughter node, then it must be present also on the mother. Condition 5(ii) is the corresponding statement for trickling features.

Our definition is given in terms of features that can be either present or absent from a category, whereas in the original definition a feature has a *value*. This distinction is merely a mathematical device to simplify the definition and the discussion of the algorithm which will follow. Our definition can be related to the original definition by reading "feature" as "feature-value pair". For example, a "real" GPSG might contain a feature PAST which can take a value which is either + or −. We interpret that as two separate features, say PAST+ and PAST−. The standard definition would require that a category may not contain both PAST+ and PAST−. This can be expressed by conjoining the FCR $\neg$(PAST+ $\wedge$ PAST−) to the FCR set. Such an FCR is called a **group FCR**.

3 A Parsing Algorithm for GPSGs

The algorithm belongs to the class of algorithms that obtain a grammar G', variously called a **skeleton grammar** or an **underlying grammar**, from a given grammar G and then parse according to G'. In these algorithms the skeleton grammar G' is chosen such that $L(G) \subseteq L(G')$, so, if the parse according to G' fails, the sentence can be rejected immediately. If the parse succeeds, it is necessary to check some additional constraints, typically by examining the parse tree, to ensure that the sentence is indeed acceptable to the more restrictive, given, grammar G. The extra checking process typically annotates the parse tree with extra information but does not change its shape. At the end of the checking process, either the sentence is rejected as not conforming to G, or the sentence is accepted, in which case the annotated parse tree is the parse tree of the sentence according to G. Wegner's (1980) algorithm for VWGs belongs to this class.

In the present algorithm, the skeleton grammar G' is a GPSG that is obtained from a given GPSG G by neglecting some of the FCRs and the percolating feature propagation constraints. The skeleton grammar can be parsed by a simple modification of Earley's (1970) algorithm. The algorithm comprises a precompilation phase, in which the skeleton grammar G' is obtained from the given grammar G, followed by three parse-time phases that are executed one after the other.

Precompilation. Given a GPSG

$$G = (V_F, V_T, X_0, R, F, F_P, F_T),$$

first define the *side-effect-free FCR set* F' algorithmically in the following manner.

1. Convert F to *clausal form*, in other words a conjunction of disjunctions of terms, each of which is either an unnegated literal feature or a negated literal feature. (This can be done uniquely, apart from questions of ordering of terms; see, e.g., Loveland (1978:32ff).)
2. Remove from the clause set all clauses (i.e., disjunctions) that contain one or more unnegated literals, leaving behind only those clauses that contain only negated literals.

The resulting set of clauses represents the side-effect-free FCR set F'. Since F' was obtained from F by *removing* clauses, the resulting function F' cannot be more restrictive than F; in other words, $F \supset F'$.

The reason for removing clauses that contain unnegated literals is that the evaluation of FCRs can, in general, cause the instantiation of new features on a node. This can be viewed as a "side effect" of the evaluation, whose primary function is to filter out inadmissible parses. Side effects are difficult to handle, because they interact with each other and with other aspects of the grammar, in particular with the propagation constraints. For example, a new feature added as a result of a "non-side-effect-free" FCR clause might cause some other clause, which was satisfied before the new feature was added, to become false. This is not possible, however, if no clause contains an unnegated literal: each clause can be satisfied only by the *absence* of one or more stated features, and if the required features are not absent, the clause yields false — there is no mechanism in GPSG for *removing* features from a category.

Now define the *skeleton grammar* G' of G by

$$G' = (V_F, V_T, X_0, R, F', \emptyset, F_T).$$

Clearly $L(G) \subseteq L(G')$, since whenever conditions 4 and 5 of section 2 hold for a derivation according to G, they will also hold for a corresponding derivation according to G'. (Remember that $F \supset F'$.) Informally, G' is more permissive than G. For the same reason, each parse according to G has a corresponding parse according to G', differing only in the distribution of features among categories on nodes. In other words, each parse tree according to G is the same "shape" as some parse tree (of the same sentence) according to G'.

Phase 1. We now parse G' by applying a modified form of Earley's algorithm (Earley 1970; see also Pulman 1985, Ritchie and Thompson 1984). (The reader is assumed to be familiar with Earley's algorithm, in particular with the rôle played by the predictor in adding new states to a state set.) The algorithm is extended so that it creates a parse tree as the parse progresses. A method for doing this is described briefly by Earley (1970) and in more detail by Earley (1968).

There is no need to handle the percolating feature propagation constraint at this stage, because in G' F_P is empty. There is no need to consider what happens when the evaluation of an FCR causes a new feature to be added to a category, since the FCR set F' is side-effect-free. We can therefore treat G' as a CFG whose non-terminals are categories, provided that we allow for extension (condition 3 of section 2) and the trickling feature constraint (condition 5(ii)). This is done in the following way.

A category appears on a node by virtue of the appearance of a category c_1 on the right-hand side of a rule R_1 and the appearance of a "matching" category c_2 on the left-hand side of a rule R_2. The extension condition permits the free addition of features to c_1 and to c_2 to generate the category c which appears on the node. By the extension condition, $c_1 \subseteq c$ and $c_2 \subseteq c$, which implies that $c \supseteq (c_1 \cup c_2)$. Now let us neglect for the moment the trickling feature constraint. Since we are ignoring the non-side-effect-free FCRs and the propagation constraints, *any* superset of $c_1 \cup c_2$ which satisfies $F'(c)$ will suffice; consequently, we take the smallest superset, namely $c_1 \cup c_2$, which is the least upper bound of c_1 and c_2 under the ordering relation of extension (see Gazdar et al. 1985:39).

Now we consider the trickling feature constraint. The effect of this constraint is to instantiate extra features on certain categories: those features that belong to a mother category and which are also members of F_T must be instantiated on each daughter category. The category that is instantiated on the node of the parse tree is the smallest superset of $c_1 \cup c_2$ which contains all of its mother's trickling features, which is $c_1 \cup c_2 \cup (c_0 \cap F_T)$, where c_0 is the category of the mother node.

To determine the category to place on a node of the parse tree, therefore, the algorithm needs to know:

- the a priori category c_1 on the right-hand side of a rule;
- the a priori category c_2 on the left-hand side of the rule which "matches" c_1;
- the fully-evaluated a posteriori category on the mother of the node to which a category is currently being assigned.

All of this information is available to the predictor in Earley's algorithm. This follows from the fact that Earley's algorithm is "top down", which means that the full category on a node is known before any of that node's daughters are considered.

We now consider how Earley's algorithm can be modified to parse the skeleton grammar in the manner outlined above. A **state** in Earley's standard algorithm can be written as $X \rightarrow \alpha \cdot \beta$, which signifies that the algorithm is considering the rule $X \rightarrow \alpha\beta$, and has successfully matched the α with some portion of the sentence being parsed. The **predictor** is applied to states $X \rightarrow \alpha \cdot Y\beta$, which have a non-terminal to the right of the dot. The predictor adds new states $Y \rightarrow \cdot\ \gamma$ for each rule $Y \rightarrow \gamma$ with matching non-terminal Y.

In the new algorithm, a state is written $(c_0)\ c \rightarrow \alpha \cdot \beta$.

As in the standard algorithm, this signifies that the algorithm is considering the rule $c \rightarrow \alpha\beta$, and has successfully matched the α. The extra category c_0 contains the features that are passed from mother to daughter and which will ultimately appear on a node of the parse tree.

The predictor in the new algorithm is applied to states $(c_0)\ c \rightarrow \alpha \cdot c_1\beta$ with a non-terminal category to the right of the dot. The predictor adds new states

$$(c_1 \cup c_2 \cup (c_0 \cap F_T))\ c_2 \rightarrow \cdot\ \gamma$$

for each rule $c_2 \rightarrow \gamma$ such that $F'(c_1 \cup c_2 \cup (c_0 \cap F_T))$ holds. (At first sight it appears that this entails a search through all of the rules of the grammar, but a means of avoiding a full search is presented in section 4.)

The rôles of the scanner and of the completer in Earley's standard algorithm are unchanged in the new algorithm. The initial state that is entered to start the parse is $(\emptyset)\ \emptyset \rightarrow \cdot\ X_0$. The associated "dummy" rule $\emptyset \rightarrow X_0$ is not considered part of the grammar, and is exempt from being matched by the predictor.

It would be quite possible to use a "bottom up" algorithm in place of Earley's algorithm, in which the rôles of trickling and percolating features would be reversed. It is not possible to handle *both* percolating *and* trickling features in phase 1, since a provisional decision at some point deep down in the parse tree to instantiate a feature on a certain category would in general cause changes to the membership of categories in remote parts of the tree.

Phase 2. The "parse tree" that is generated by Earley's algorithm is in general not a tree at all; it is a directed graph. Besides non-terminal nodes and terminal nodes, the graph will in general contain branching nodes that point to alternative daughters of a non-terminal node. It is by this means that multiple parses, arising from an ambiguous sentence, are represented. If the degree of ambiguity of the sentence with respect to the skeleton grammar is infinite, the finite graph must represent infinitely many distinct parse trees; in this case the graph is cyclic. We assume that the degree of ambiguity is finite, in which case the graph is a **directed acyclic graph** (DAG). A DAG differs from a tree in that whereas each node in a tree (except the root) has precisely one parent, a node in a DAG may have more than one parent. In other words, a DAG represents common sub-trees only once; a single sub-tree may be descended from several parents. DAGs are often used in the construction of compilers for computer programming languages.

Let p be the degree of ambiguity of the skeleton grammar, i.e., the number of distinct parse trees represented by the DAG. We expand the DAG, generating p distinct parse trees. This can easily be done by means of conventional tree processing techniques, provided that p is finite, in other words if the graph is acyclic.

Phase 3. Each distinct parse tree is examined in turn. For each tree, sufficient features are added to the categories on each node of the tree to cause the tree to reflect a parse according to the original GPSG G. This entails the evaluation of F and of the propagation constraints on each category, and the construction of a category on each node which satisfies all of the constraints. Once again, the smallest possible category is constructed. That is, if a category c satisfies all of the constraints, and so does a larger category $c \cup c'$, we choose c. It is debatable whether this is the correct behaviour; some might argue that separate parse trees ought to be constructed in which all possible legal extensions are shown. However, the resulting set of parse trees would then in general be very large, and it is difficult to believe that this behaviour is desirable. Our smallest category is similar to the most general unifier of a set of expressions in mathematical logic; as in logic, particular, less general instances can be derived from the most general case, but it is the most general (least fully specified) case that is of most interest.

We assume that the FCR set F is expressed in clausal form and that each clause (i.e., each disjunction) is a Horn clause (a clause with either zero or one unnegated literal; see, e.g., Loveland (1978:99)). Number the clauses $F_1, \ldots, F_M$.

We denote by M(N) the mother of the node N, if it exists (i.e. unless N is the root of its tree). We denote by C(N) the category on the node N.

Let the distinct parse trees produced by phase 2 be $T_1, \ldots, T_p$. The algorithm *unify*, defined below, is applied to each T_i in turn, for $i = 1, \ldots, p$.

unify (T): Let the non-terminal nodes in T be $N_1, \ldots, N_N$.

```
1.          set again := false;
2.          for j = 1, ..., N do
2.1.          if N_j is not the root of T and
                (C(M(N_j)) ∩ F_T) ⊈ C(N_j) then
2.1.1.            C(N_j) := C(N_j) ∪ (C(M(N_j)) ∩ F_T);
2.1.2.            set again := true;
2.2.          if N_j is not the root of T and
                (C(N_j) ∩ F_P) ⊈ C(M(N_j)) then
2.2.1.            C(M(N_j)) := C(M(N_j)) ∪ (C(N_j) ∩ F_P);
2.2.2.            set again := true;
3.          for j = 1, ..., N do
3.1.          for k = 1, ..., M do
3.1.1.          let f+ be the set of unnegated literals in
                  F_k;
3.1.2.          let f_ be the set of negated literals in F_k;
3.1.3.          if f_ ⊈ C(N_j) then
3.1.3.1.          if f+ = ∅ then fail;
3.1.3.2.          if f+ ⊈ C(N_j) then
3.1.3.2.1.          C(N_j) := C(N_j) ∪ f+;
3.1.3.2.2.          set again := true;
4.          if again then go to step 1;
5.          output T.
```

Proof of the algorithm. First notice that the flag *again* is set (in steps 2.1.2, 2.2.2, and 3.1.3.2.2) whenever a feature is added to a category that is not already in that category, and at no other time. Since there are only finitely many features, steps 1 to 4 are repeated only finitely many times, so the procedure terminates.

Next observe that on successful termination, *again* is false, so steps 2.1 and 2.2 must have been obeyed for each node with the conditions in 2.1 and 2.2 false each time. Consequently, on successful termination, the propagation constraints hold for each node.

Finally, on successful termination, the FCR set F also holds for each node, for the following reasons. Step 3.1.3 checks the negated literals in the clause F_k against the category $C(N_j)$. If the condition in 3.1.3 is false, there is at least one negated literal in F_k which is indeed absent from $C(N_j)$, so the clause F_k is satisfied. If, on the other hand, the condition in 3.1.3 is true, none of the negated literals can possibly be satisfied, since features may not be *removed* from a category, only added. Since F_k is a Horn clause, there is at most one unnegated literal in F_k, so f_+ is either empty or has one member. If f_+ is empty, the clause can not be satisfied, so the algorithm fails. If f_+ is not empty, the feature is added to the category if it is not already there, and the clause is thereby satisfied. The addition of the new feature might invalidate previously satisfied clauses or propagation constraints, so the flag *again* is set which causes the propagation constraints and FCR clauses to be checked afresh. As noted, the process will eventually terminate with all propagation constraints and all FCR clauses satisfied, or else the algorithm will fail, in which case the sentence does not belong to the language generated by the original grammar G.

End of proof.

Now consider what would happen if one of the clauses were not a Horn clause. The algorithm would not know which of the several features from f_+ to add in step 3.1.3.2.1 in order to satisfy the clause. The only solution would seem to be to generate copies of the parse tree, and to follow through each choice of feature from f_+ on a different copy of the parse tree, finally presenting the user of the parsing system with all of the parse trees. This would cause a combinatorial explosion, since the splitting and copying would have to be done at each level of the parse tree at which the particular feature in question is instantiated.

The linguistic consequences of the Horn clause restriction are not clear, but experience with the parsing system suggests that they are not severe. The Horn clause restriction prohibits the grammar writer from writing FCRs such as

$$[\text{PRD } +] \wedge [\text{VFORM}] \supset [\text{VFORM PAS}] \vee [\text{VFORM PRP}]$$

(Gazdar et al. 1985:111), in which a disjunction of non-negated literals appears on the right of $\supset$. It is in such FCRs that the Horn clause restriction appears in its true colours, as a mechanism for curbing a combinatorial explosion or, to put it another way, a mechanism for prohibiting a source of non-determinism. If the consequences of forbidding such FCRs later appear too severe, the possibility will be investigated of moving the non-determinism from the FCRs into the rules of the grammar, by replacing an FCR like the one above by a new FCR

$$[\text{PRD } +] \wedge [\text{VFORM}] \supset [\text{F}]$$

where F is a new feature, and adding appropriate rules to the grammar. The details of this have yet to be worked out; it is presented as a possible solution to a problem that has not yet arisen.

Time and space bounds. The following parameters are relevant to a consideration of time and space bounds for the algorithm:

- p, the degree of ambiguity of the skeleton grammar;
- K, the cardinality of V_F;
- G, the number of rules in the grammar;
- n, the length of the sentence being parsed.

Earley's algorithm, as is well known, operates in time order G^2n^3. Earley's proof of the time complexity of his algorithm (Earley 1970) is in no way affected by the elaboration of the predictor to handle feature matching. In particular, the number of states in a state set does not increase with K. Although K features may in principle be combined to construct 2^K different categories, the algorithm generates new categories by extension only when they are required. In fact, if a new feature specification is added to a rule that is previously unspecified for that feature, the state sets will either remain the same size or become *smaller*, since adding a feature restricts the range of rules that the rule in question will "match". Speaking informally, it is underspecified rules that cause the problems; the more fully specified are the rules, the closer is the GPSG to a CFG, and the fewer are the states that are needed.

The factor K does, however, enter into the time bound for phase 1 in the following manner. Although the number of "primitive steps" (Earley's terminology) that are executed by the modified algorithm is independent of K, the time taken to complete certain primitive steps, in particularly the addition of a state to a state set and the feature matching operation in the predictor, is proportional to K. The overall time bound is therefore KG^2n^3.

The expansion of the DAG to yield p distinct parse trees can be done by conventional tree processing techniques in time proportional to p, the number of nodes in each tree, and the size of a node (which affects the time taken to copy a node). This gives a bound of order $pKGn^2$ for phase 2.

The algorithm *unify* contains three nested loops (steps 3 and 3.1, and the *again* loop). An upper bound on the number of nodes is a constant times Gn^2, and an upper bound on the number of times round the *again* loop is the number of features, K. To simplify the analysis, we take $M \sim G$. (Formally, we *define* G to be the sum of the number of rules in the grammar and the number of clauses in the FCR set.) Moreover, the set operation $\subseteq$ in step 3.1.3 can realistically be expected to

take time proportional to K, although the operations involving f_+ can be done in constant time, since f_+ has either zero or one member. A time bound for *unify* is therefore $K^2G^2n^2$. Finally, *unify* is obeyed p times, which gives a time bound for phase 3 of order $pK^2G^2n^2$.

It is unfortunate that, as noted earlier, p is not in general independent of n. To see why this is so, consider the following grammar.

Non-terminal alphabet: {S}
Terminal alphabet: {a}
Rules: S → S S
S → a

It has been shown (Church and Patil 1982) that the number of distinct trees generated, for a sentence of length n, grows factorially with n. This means that the algorithm as a whole will take factorial time to parse a sentence of length n according to this grammar. This is a matter of concern, because constructions similar to this example are commonly used to handle coordination. It is even possible in principle for p to be infinite, in which case the algorithm will not terminate (although the advertised time bound of $pK^2G^2n^3$ will still hold!). In practice, however, no grammar has been encountered which *unavoidably* has infinite p. (Self-referential rules of the form X → X have occasionally appeared, but these were always traced to an error in the grammar.)

Considerable effort has been expended in an attempt to improve the theoretical worst-case performance of the algorithm when p is a finite valued but rapidly increasing function of n. It might be possible to combine phases 2 and 3, employing "lazy evaluation" (a technique often used in functional programming) to expand the DAG only when necessary. If this were done, much of the DAG might remain unexpanded, with consequent savings in time and space. The problem with this approach is that some features are required to percolate right to the root of their tree, and a given branching point might have different (and incompatible) features percolating to it from each of its alternative descendants. It turns out to be often necessary to expand the DAG all the way back to the root, in which case little is saved by using lazy evaluation. It is worth pointing out that, in cases (such as the example) in which the algorithm is least efficient, the output is often very large, consisting of many parse trees. In many (but not all) of these cases, the time taken is asymptotically linear in the length of the *output*, i.e. the number of nodes in the set of parse trees displayed. Surely, no algorithm can ever behave sublinearly on the length of its output. Furthermore, as discussed later, in cases in which this problem does not arise, the execution time is dominated by phase 1. We therefore have an algorithm that:

- behaves as well as one of the best general CF parsing algorithms, for all unambiguous grammars and for many ambiguous grammars;
- takes time that is linear in the length of the output for some "problem" grammars; and
- takes a very long time in a small number of really awkward cases.

The time bounds for the three phases are KG^2n^3, $pKGn^2$, and $pK^2G^2n^2$. This gives an overall worst-case time bound of order $pK^2G^2n^3$.

The space bound is of order $pKGn^3$ in the worst case, for the following reasons. Earley's algorithm requires space proportional to KGn^2 to hold the states, n for the state sets (that is, the list-processing overhead), and KGn^3 for the DAG. The grammar itself requires space proportional to KG. The p distinct parse trees require space proportional to KGn^2 each. Phase 3 does not require any working storage. The worst-case space bound is therefore of order $pKGn^3$.

4 Implementation

A practical GPSG parsing system has been constructed, based on the algorithm just described. The system comprises a table generator and a parser. The system was originally written in the programming language BCPL, and ran on a VAX 780 computer under the Unix operating system. The system has recently been reimplemented in C to run on a Sun 3/50 workstation. The Sun version generally runs several times faster than the VAX version. The parse times given below relate to the slower VAX implementation.

The table generator performs the precompilation phase of the algorithm. It generates a tabular representation of the skeleton grammar, which the parser can interpret more efficiently than it could the "raw" rules, and it converts the FCR set into clausal form. The table generator also performs various checks to ensure, as far as possible, that the grammar is well formed. Besides the obvious syntactic checks (to detect such errors as a comma in the wrong place), the table generator checks that the FCR set is not identically false, that there are no obvious blind alleys or non-reachable categories (this is not checked rigorously), and that various other subtle "well-formedness" conditions are satisfied. This error checking has proved very useful in practice, since GPSGs are notoriously difficult to debug.

The input grammar is written in the notation of Gazdar et al. (1985), with a few concessions to the limitations of the typical computer input device. In particular, features have values, and what we have referred to as a feature is, in the notation accepted by the table generator, a feature-value pair, written [*f v*]. Each distinct feature value pair is associated by the table generator with a particular bit in a computer word. A category is represented by a set of bits, i.e., by a word with several bits set, one for each feature-value pair in the set. Category valued features correspond to trees, and a distinct bit is allocated to each terminal node of the tree. For example, the feature PAST, with two

values + and −, would have two bits allocated to it, and for a category valued feature SLASH, the values [N +, V −] and [N−, V +] would be allocated four bits. Note that, in any given grammar, the depth of the tree induced by a category valued feature is finite; furthermore, the range of possible values of a category valued feature is known at table generation time, so it is known at this stage how many bits to allocate to the feature. The representation of feature-value pairs by bit positions in a computer word allows the very efficient logical instructions of the computer ($\cap$, $\cup$, $\neg$), which operate on a whole word of bits at a time, to be used.

As explained earlier, a feature in GPSG 85 may take at most one value at a time, since a GPSG 85 feature is in fact a function. This restriction is expressed by conjoining an FCR, known as a **group FCR**, to the FCR set. For example, if the grammar contains the two-valued feature PAST referred to above, the FCR

$$\neg([\text{PAST} +] \wedge [\text{PAST} -])$$

would be conjoined to the FCR set. In general, the presence of an n-valued feature f, with values $v_1, \ldots, v_n$, entails the addition of the FCRs

$$\neg([f\ v_i] \wedge [f\ v_j]) \text{ for each } i, j = 1, \ldots, n,\ i < j.$$

When converted to clausal form these FCRs become the $n(n-1)/2$ clauses

$$\neg[f\ v_i] \vee \neg[f\ v_j] \text{ for each } i, j = 1, \ldots, n,\ i < j$$

whose inclusion in the set of clauses presented to the parser would make the table very large. Consequently, these group clauses are abbreviated. For each n-valued feature f with $n \geq 2$, a **group mask** is included in the parser table which has one bit set for each feature-value pair whose conjunction is to be prohibited. The parser checks these group masks whenever it consults the FCR clause set. If g is a group mask and c is a mask representing a category, the parser has only to check that $(g \cap c)$ has not more than one bit set.

It has been observed that, in practice, it is likely that the explicit FCR set supplied by the grammar writer will contain mostly non-side-effect-free clauses. However, the (notional) group FCRs *are*, by definition, side-effect-free. Because of this, the algorithm is modified for implementation in the following way. The FCR set F' which is used in the definition of the skeleton grammar is taken to be just the set of notional group FCRs; any "genuine" FCRs, be they side-effect-free or not, are excluded from F'. Furthermore, the table generator ensures that the full FCR set F is satisfied on each node at table generation time. For example, if the grammar contains the FCR

$$[\text{NOM}] \supset [\text{NFORM NORM}],$$

then a category [NOM +] occurring in a phrase structure rule would be rewritten by the table generator as [NOM +, NFORM NORM]. These modifications increase the efficiency of the implementation, and enable certain errors to be detected at table generation time.

In practice, most GPSGs closely resemble traditional CFGs, with most categories fully specified for the "major" features N, V, and perhaps BAR. Consequently, the group FCR constraints ensure that the skeleton grammar also resembles a traditional CFG, and is certainly not, in practice, massively ambiguous. Indeed, the table generator insists that categories in rules are written as X[Y], where X is a name (a traditional non-terminal), and Y is a category. The non-terminal X is defined (by the grammar writer) to stand for some set of major features. This convention is perhaps controversial, but Gazdar et al. (1985) is full of such rules, and the linguists who use the parsing system have not grumbled yet. The convention does allow the table generator to check the grammar more stringently than would otherwise be the case, and it enables the parser to be made considerably more efficient, by dividing the set of all categories (which must be searched by the predictor) into disjoint subsets. The convention has no theoretical significance; the program would work without it.

The head feature convention. The grammar writer is able to denote certain non-terminals on the right-hand side of a rule as head non-terminals, which correspond to the head symbols of traditional X-bar syntax. This is done by prefixing the name of the non-terminal in the rule by a star. The percolation and trickling of features can be restricted to occur only between a mother and a head daughter. There are thus nine possible propagation behaviours for any feature:

one of
- not trickling
- trickling, but only to head daughters
- trickling to all daughters

together with one of
- not percolating
- percolating, but only from head daughters
- percolating from any daughter

The head feature convention is simulated by defining head features to trickle, but only to heads. This is adequate in most situations, but it falls short of the behaviour postulated by Gazdar et al. (1985:94ff). In particular, the notion of free feature specification sets is not accommodated. This causes problems in, for example, the treatment of conjunctions, in which the conjoined constituents are conventionally all heads. GPSG 85 allows a rule that in our notation would be written

NP: *NP [CONJ and], *NP [CONJ NIL].

In the present implementation, any PER feature (for example) which happens to be present on the mother would trickle to both head daughters, thereby forcing agreement between the daughters. Our solution has been to make the daughters non-heads, which is unattractive, but which has been made to work.

The foot feature principle. The foot feature principle is more of a problem than the head feature convention. It

is clear that foot features ought to percolate, but the situation is more complicated than this. In a rule such as

S: NP, S [SLASH NP]

the SLASH feature (which is a foot feature) must be prevented from percolating from the node that is generated by extension from the right-hand-side S. This is achieved by forbidding any feature that has been declared to be a foot feature (e.g., SLASH and WH) to percolate from a node on which the feature appears by virtue of its appearance on the right-hand side of a phrase structure rule. This is easy to implement.

This is only a partial solution to the problem, however. The rule given above correctly generates

the telephone Carol tested.

It is not possible, however, by this mechanism to prevent

* the telephone Carol tested the telephone,

in which "Carol tested the telephone" is correctly parsed as an S, but in which a SLASH NP specification is "gratuitously" instantiated in order to satisfy the extension conditions imposed by the rule given above. To solve this and other problems, a tree is now defined to be admissible only if each non-terminal node of the tree satisfies the **foot condition**, which is related to the original FFP of GPSG 85. The foot condition is defined as follows.

Define a **lexical node** of a parse tree as a node that immediately dominates a terminal node. (A gap, which is explicitly denoted in the grammar by the word GAP, is a terminal node.) Define an **interior node** as a node that is neither terminal nor lexical. An interior node is said *to meet the foot condition* (FC) iff each foot feature that it contains appears also on at least one daughter *from which it can legally percolate*. A lexical node is said to meet the FC iff each foot feature that it contains appears also on the left-hand side of the lexical rule that gave rise to the lexical node.

This definition implies that the FC cannot cause the instantiation of any features. In this respect, the FC differs from the propagation conventions, which add the necessary features to make the conditions hold. The FC mechanism operates on the tree as it is after FCRs and propagation conditions have been enforced. It does not alter the tree; it merely checks that the foot condition is true on each node. Note that all of this follows from the definition. It is not necessary to put forward a procedural definition of the FC, which would fit ill with the non-procedural definition of GPSG. In contrast to the GPSG 85 FFP, the FC readily permits a straightforward, efficient, and deterministic implementation.

The control agreement principle. A mechanism has been provided for specifying horizontal propagation of features in a way similar to that implied by the control agreement principle of Gazdar et al. (1985). Sister categories in a rule may be designated **control sisters** (by prefixing the name of the non-terminal by a dollar). A set of **control features** is defined by the grammar writer, analogous to the sets of trickling and percolating features. Each non-terminal node N has associated with it an extra node N'. If a node N_0 has daughters $N_1, \ldots, N_n$, then N_0' is called the **stepmother** of each $N_1, \ldots, N_n$. If N_i is a control sister, then any control features in $C(N_i)$ are required to percolate to the stepmother, and any features on the stepmother are required to trickle to each stepdaughter that is a control sister. The effect is that control features present on a control sister are forced to appear on each other control sister (which has the same mother).

One consequence of this modified CAP is that agreement is mutual, or bidirectional, whereas in the CAP of GPSG 85 it is unidirectional. Another consequence is that, in the present implementation, it is impossible by these means to express agreement between (for example) the daughter NP and the NP "under the slash" in

S: NP, S [SLASH NP].

This has not yet proved to be a problem; such agreement can easily be accommodated by defining appropriate propagation constraints for those features (such as PER, PLU and NFORM) that must agree.

Metarules. Despite the misgivings expressed earlier concerning the possible exponential growth in grammar size, a form of metarule mechanism has been incorporated. Metarules are implemented by precompilation by the table generator. In fact, there is a separate metarule preprocessing program, called **metagee**, which runs as a Unix filter, passing the expanded set of rules to the table generator proper. It would be possible to process separated ID/LP rules by means of a similar preprocessor. This has not been done.

Form of the parser table. The output from the table generator, the table which is interpreted by the parser, comprises:

- an encoded list of rules, with a pointer from each occurrence of a non-terminal on the right-hand side of a rule to a list of rules with matching left-hand side non-terminals;
- an encoded lexicon;
- a list of non-terminal names, feature names and feature value names;
- a set of group masks;
- a set of FCR clauses, each comprising two bit masks. One mask (word of bits) represents the category f_+ and the other represents the category f_-.

Performance. The parsing system has been tested with a grammar for a subset of English. The grammar contains 512 rules after metarule expansion, comprising 228 non-lexical rules and 284 lexical rules. There are 107 feature value pairs. There are 18 FCRs, which, when converted to clausal form, yield only 39 clauses. The size of the parser table is about 63,000 bytes, about 93% of which is occupied by the encoded rules. The remaining 7% (4,500 bytes) comprises the tables of bit masks

that represent the FCRs and the propagation masks, a table of non-terminal names, and the lexicon. The table generator takes about two minutes to compile the grammar.

Typical parse times are given in figure 1. As the table shows, for simple short sentences (unambiguous sentences of fewer than 15 words), phase 1 consistently takes more time than phases 2 and 3 together. For sentences of moderate ambiguity, the times for phase 1 and phase 2+3 are comparable. The 15-word sentence for which a time is given in the table is

Length of sentence (words)	Parse time (seconds) Phase 1	Phase 2+3	Total
3	0.8	0.6	1.4
6	2.3	1.1	3.4
9	3.9	2.3	6.2
12	3.6	2.9	6.5
15	4.9	7.9†	12.8

† 4.2s excluding the time taken to format and print the trees

Figure 1.

> which number ought Carol to have dialed on the telephone the happy engineer was testing?

which, the parser correctly reports, is ambiguous (it has two parses). Phase 1 yields a DAG that represents four parses. Phase 2 expands this into four distinct trees, two of which are then ruled out by phase 3. The figures for phase 2+3 include the time taken to format and print the trees, which for the longer sentences is not insignificant, amounting to almost half of the processing time for the 15–word sentence.

Acknowledgments

The parsing system described in this paper was developed for use in a research project that is funded by British Telecom Research Laboratories (R18).

I am grateful to C.J. Cullen and S.J. Harlow for their comments on this paper at various stages of revision.

References

Barton, G. Edward, Jr. 1985 On the complexity of ID/LP parsing. *Computational Linguistics* 11(4): 205–218.

Briscoe, Edward J. 1986 *Grammar Development Environment.* Internal report, Department of Linguistics and Modern English Language, University of Lancaster, U.K.

Church, Kenneth and Patil, Ramesh 1982 Coping with syntactic ambiguity or how to put the block in the box on the table. *Computational Linguistics* 8(3–4): 139–149.

Earley, Jay 1968 *An efficient context-free parsing algorithm.* Ph.D. thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA.

Earley, Jay 1970 An efficient context-free parsing algorithm. *Communications of the Association for Computing Machinery* 13(2): 94–102.

Fisher, Anthony J. 1985 Practical LL(1)-based parsing of van Wijngaarden grammars. *Acta Informatica* 21: 559–584.

Gazdar, Gerald 1983 *Recent computer implementations of phrase structure grammars.* Internal report, Cognitive Studies Programme, University of Sussex, U.K.

Gazdar, Gerald; Klein, Ewan; Pullum, Geoffrey K.; and Sag, Ivan 1985 *Generalized Phrase Structure Grammar.* Basil Blackwell, Oxford, U.K.

Harrison and Maxwell 1986 A New Implementation for GPSG. *Proc. 6th Canadian Conf. on A.I. (CSCSI-86), May 21–23, École Polytechnique de Montréal, Montréal, Québec*: 78–83.

Loveland, Donald W. 1978 *Automated Theorem Proving: a Logical Basis.* North-Holland, Amsterdam, the Netherlands.

Pulman, S.G. 1985 Generalised phrase structure grammar, Earley's algorithm, and the minimisation of recursion. In: Sparck Jones, Karen; and Wilks, Yorick (eds.) 1985 *Automatic Natural Language Parsing.* Ellis Horwood, Chichester, U.K.

Ristad, Eric S. 1985 *GPSG recognition is NP-hard.* Artificial Intelligence Memo no. 837, MIT Artificial Intelligence Laboratory, Cambridge, MA.

Salomaa, Arto 1973 *Formal Languages.* Academic Press, London, England.

Shieber, Stuart M. 1983 Direct parsing of ID/LP grammars. Technical report 291R, SRI International, Menlo Park, CA. Also in: *Linguistics and Philosophy* 7(2): 135–154.

Thompson, Henry S. 1982 *Handling metarules in a parser for GPSG.* Research paper no. 175, Department of Artificial Intelligence, University of Edinburgh, U.K. Also in: Barlow, M.; Flickinger, D.; and Sag, I.A. (eds.) Developments in Generalized Phrase Structure Grammar. *Stanford Working Papers in Grammatical Theory* 2: 26–37. Indiana University Linguistics Club, U.S.A.

Ritchie, Graeme; and Thompson, Henry S. 1984 Natural language processing. In: O'Shea, Tim and Eisenstadt, Marc 1984 *Artificial Intelligence: Tools, Techniques and Applications.* Harper and Row, New York, NY.

Wegner, L.M. 1980 On parsing two-level grammars. *Acta Informatica* 14: 175–193.

Wijngaarden, A. van; Mailloux, B.J.; Peck, J.E.L.; Koster, C.H.A.; Sintzoff, M.; Lindsey, C.H.; Meertens, L.G.L.T.; and Fisker, R.G. (eds.) 1976 *Revised Report on the Algorithmic Language Algol 68.* Springer-Verlag, Berlin, W. Germany.