The Interface between Phrasal and Functional Constraints

John T. Maxwell III*
Xerox Palo Alto Research Center

Ronald M. Kaplan†
Xerox Palo Alto Research Center

Many modern grammatical formalisms divide the task of linguistic specification into a context-free component of phrasal constraints and a separate component of attribute-value or functional constraints. Conventional methods for recognizing the strings of a language also divide into two parts so that they can exploit the different computational properties of these components. This paper focuses on the interface between these components as a source of computational complexity distinct from the complexity internal to each. We first analyze the common hybrid strategy in which a polynomial context-free parser is modified to interleave functional constraint solving with context-free constituent analysis. This strategy depends on the property of monotonicity in order to prune unnecessary computation. We describe a number of other properties that can be exploited for computational advantage, and we analyze some alternative interface strategies based on them. We present the results of preliminary experiments that generally support our intuitive analyses. A surprising outcome is that under certain circumstances an algorithm that does no pruning in the interface may perform significantly better than one that does.

1. Introduction

A wide range of modern grammatical formalisms divide the task of linguistic specification either explicitly or implicitly into a context-free component of phrasal constraints and a separate component of attribute-value or functional constraints. Lexical-Functional Grammar (Kaplan and Bresnan 1982), for example, is very explicit in assigning both a phrase structure tree and an attribute-value functional structure to every sentence of a language. Generalized Phrase Structure Grammar (Gazdar, Klein, Pullum, and Sag 1985) assigns a phrase structure tree whose categories are attribute-value structures. For Functional Unification Grammar (Kay 1979) and other unification formalisms that evolved from it (such as HPSG [Pollard and Sag 1987]), the phrase structure is more implicit, showing up as the record of the control strategy that recursively reinstantiates the collection of attribute-value constraints from the grammar. For Definite Clause Grammars (Pereira and Warren 1980) the phrase structure is implicit in the unification of the concealed string-position variables and the recursive reinstantiation of the additional logic variables that carry functional information.

The computational problem of recognizing whether a given string belongs to the language of a grammar also divides into two parts, since it must be determined that the string satisfies both the phrasal and functional constraints. These two types of constraints have different computational properties. It is well known that context-free phrase structure constraints can be solved in time polynomial in the length of the input sentence, whereas all known algorithms for solving Boolean combinations of

* 3333 Coyote Hill Rd, Palo Alto, CA 94304. E-mail: maxwell.parc@xerox.com
† 3333 Coyote Hill Rd, Palo Alto, CA 94304. E-mail: kaplan.parc@xerox.com

equality or unification constraints in the worst-case run in time exponential in size of the constraint system.

There have been a number of approaches for implementing such hybrid constraint systems. In one approach the context-free constraints are converted to the form of more general functional constraints so that a general-purpose constraint satisfaction method can uniformly solve all constraints. While this has the advantage of simplicity and elegance, it usually gains no advantage from the special properties of the context-free subsystem. The original implementation for Definite Clause Grammars followed this strategy by translating the grammar into equivalent Prolog clauses and using the general Prolog interpreter to solve them.

On the other hand, functional constraints of a sufficiently restricted kind can be translated into context-free phrasal constraints and solved with special purpose mechanisms. This is true, for example, of all GPSG feature constraints. In the extreme, a GPSG could be completely converted to an equivalent context-free one and processed with only phrasal mechanisms, but the fast polynomial bound may then be overwhelmed by an enormous grammar-size constant, making this approach computationally infeasible for any realistic grammar (Barton, Berwick, and Ristad 1987).

More common approaches involve hybrid implementations that attempt to take advantage of the special computational properties of phrasal constraints while also handling the general expressiveness of arbitrary feature constraints. Although this sounds good in principle, it turns out to be hard to accomplish in practice. An obvious first approach, for example, is to solve the context-free constraints first using familiar polynomial algorithms (Earley 1970; Kaplan 1973; Younger 1967), and then to enumerate the resulting phrase structure trees. Their corresponding functional constraints are solved by converting to disjunctive normal form (DNF) and using also well-known general purpose constraint algorithms (Nelson and Oppen 1980; Knight 1989).

This configuration involves a simple composition of well-understood techniques but has proven to be a computational disaster. The phrasal mechanisms compute in polynomial time a compact representation of all possible trees, each of which presents a potentially exponential problem for the constraint solver to solve. If the phrasal component is not properly restricted, there can be an infinite number of such trees and the whole system is undecidable (Kaplan and Bresnan 1982). But even with an appropriate restriction on valid phrase structures, such as LFG's prohibition against nonbranching dominance chains, the number of such trees can be exponential in the length of the sentence. Thus, even though a context-free parser can very quickly determine that those trees exist, if the grammar is exponentially ambiguous then the net effect is to produce an exponential number of potentially exponential functional constraint problems.

This is an important observation. There have been several successful efforts in recent years to develop solution algorithms for Boolean combinations of functional constraints that are polynomial for certain special, perhaps typical, cases (Kasper 1987; Maxwell and Kaplan 1989; Dörre and Eisele 1990; Nakano 1991). But even if the functional constraints could always be solved in polynomial time (for instance, if there were no disjunctions), the simple composition of phrasal constraints and functional constraints would still in the worst case be exponential in sentence length. This exponential does not come from either of the components independently; rather, it lies in the interface between them.

Of course, simple composition is not the only strategy for solving hybrid constraint systems. A typical approach involves interleaving phrasal and functional processing. The functional constraints associated with each constituent are incrementally solved as the constituent is being constructed, and the constituent is discarded if those constraints prove to be unsatisfiable. Although this interface strategy avoids the blatant excesses of simple composition, we show below that in the worst case it is also exponential in sentence length. However, it is too early to conclude that there is no subexponential interface strategy, since the computational properties of this interface have not yet been extensively investigated. This paper maps out a space of interface possibilities, describes alternative strategies that can provide exponential improvements in certain common situations, and suggests a number of areas for further exploration.

2. Interleaved Pruning

We begin by examining in more detail the common hybrid strategy in which a polynomial context-free parser is modified to interleave functional constraint solving with context-free constituent analysis. All known polynomial parsers make essentially equivalent use of a well-formed substring table (Sheil 1976), so we can illustrate the computational properties of interleaved strategies in general by focusing on the familiar operations of active-chart parsing (Kaplan 1973; Kay 1980; Thompson 1983). There are, of course, other popular parsers, such as the generalized LR(k) parser (Tomita 1986); however, in the worst case these are known not to be polynomial (Johnson 1989) unless a chartlike mechanism is added (Schabes 1991), and so they raise no new interface issues. Here and in the remainder of this paper we assume the restriction against nonbranching dominance chains to guarantee termination of the parsing computation.

2.1 The Active Chart Parser

Recall that the chart in an active-chart parser contains edges that record how various portions of the input string match the categorial sequences specified by different rules. An inactive edge spans a substring that satisfies all the categorial requirements of a rule and thus represents the fact that a constituent has been completely identified. An active edge spans a substring that matches only part of a rule and represents a constituent whose daughters have only been partially identified. An active edge may span an empty substring at a particular string position and indicate that no rule categories have yet been matched; such an edge represents the unconfirmed hypothesis that a constituent of the rule's type starts at that string position.

The chart is initialized by adding inactive edges corresponding to the lexical items and at least one empty active edge before the first word. The active edge represents the hypothesis that an instance of the root category starts at the beginning of the input string. The computation proceeds according to the following fundamental rules: First, whenever an active edge is added to the chart, then a new edge is created for each of the inactive edges to its right whose category can be used to extend the rule-match one step further. The new edge records the extended match and spans the combined substrings of the active and inactive edges. Also, for each category that can extend the active edge, a new empty edge is created to hypothesize the existence of a constituent of that type beginning to the right of the active edge. Second, whenever an inactive edge is added to the chart, a new edge is similarly created for each active edge to its left whose rule-match can be extended by the category of the inactive edge. Newly created edges are added to the chart and spawn further computations only if they are not equivalent to edges that were added in previous steps. Thus, in Figure 1, only one new edge n is created for the four different ways of combining the active edges a_x with the inactive edges i_y.

The polynomial behavior of this algorithm for a context-free grammar depends crucially on the fact that equivalent edges are proscribed and that the number of

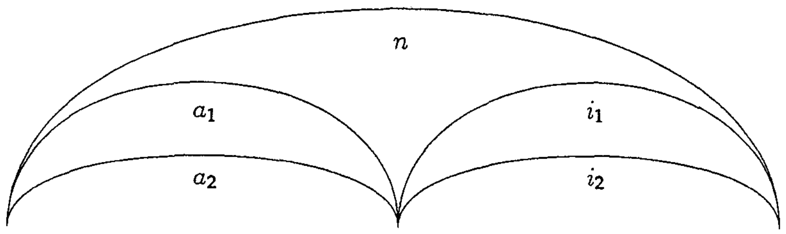

Figure 1
Context-free edge creation.

distinct edges is polynomial in sentence length. In the context-free case, two edges are equivalent if they span the same substring and impose exactly the same requirements for further matching of the same rule. The polynomial bound on the number of distinct edges comes from the fact that equivalence does not depend on the internal substructure of previously matched daughter constituents (Sheil 1976). The chart data structures are carefully organized to make equivalent edges easy to detect.

Conceptually, the chart is only used for determining whether or not a string belongs to the language of a context-free grammar, and by itself does not give any trees for that string. A *parse-forest* variation of the chart can be created by annotating each edge with all of the combinations of active and inactive edges that it could come from (these annotations are ignored for the purpose of equivalence). This representation can be used to read out quickly each of the trees that is allowed by the grammar. Note that a parse-forest representation still only requires space polynomial in sentence length since there are only a polynomial number of ways for each of the edges to be constructed out of edges with the same termination points.

2.2 Augmenting the Active Chart Parser with Functional Constraints

The main benefit of the chart algorithm is that subtrees are not recomputed when they are incorporated as daughters in alternative trees. It is possible to retain this benefit while also allowing functional constraints to be processed as constituents are being analyzed. Edges are augmented so that they also record the functional constraints associated with a constituent. The constraints associated with lexical items are stored in the initial inactive edges that correspond to them. Whenever a new edge is created from an active and an inactive, its constraints are formed by conjoining together the constraints of those edges with the constraints specified on the rule category that matches the inactive edge. Having collected the constraints for each edge in this way, we know that the input string is grammatical if it is spanned by a root-category edge whose constraints are satisfiable. Note that for this to be the case, the notion of equivalence must also be augmented to take account of the constraints: two edges are equivalent now if, in addition to satisfying the conditions specified above, they have the same constraints (or perhaps only logically equivalent ones).

These augmentations impose a potentially serious computational burden, as illustrated in Figure 2. Here, ϕ_x and ψ_y represent the constraints associated with a_x and i_y, respectively. Although we are still carrying out the steps of the polynomial context-free algorithm, the behavior is no longer polynomial. The constraints of an edge include those from the particular rule-categories that match against its daughter edges, with different daughter matches resulting in different constraints. The net effect is that there can be a different set of constraints for every way in which a particular category can

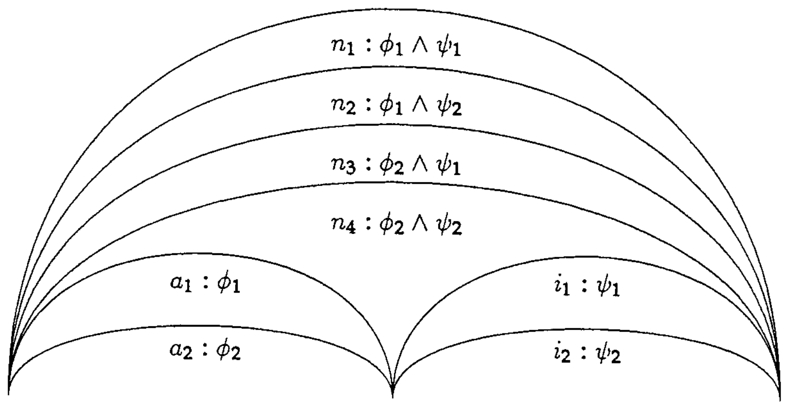
$n_1 : \phi_1 \wedge \psi_1$
$n_2 : \phi_1 \wedge \psi_2$
$n_3 : \phi_2 \wedge \psi_1$
$n_4 : \phi_2 \wedge \psi_2$
$a_1 : \phi_1$
$a_2 : \phi_2$
$i_1 : \psi_1$
$i_2 : \psi_2$

Figure 2
Augmented edge creation.

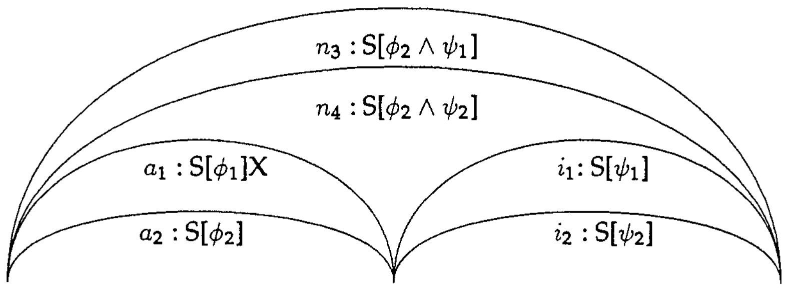
$n_3 : \mathrm{S}[\phi_2 \wedge \psi_1]$
$n_4 : \mathrm{S}[\phi_2 \wedge \psi_2]$
$a_1 : \mathrm{S}[\phi_1]\mathrm{X}$
$a_2 : \mathrm{S}[\phi_2]$
$i_1 : \mathrm{S}[\psi_1]$
$i_2 : \mathrm{S}[\psi_2]$

Figure 3
The advantage of pruning.

be realized over a given substring. If the phrase structure grammar is exponentially ambiguous, there will be exponentially many ways of building at least one constituent, and there will be exponentially many edges in the chart (distinguished by their constraints). Thus we retain the time benefit of avoiding subtree recomputation, but the algorithm becomes exponential in the worst case.

2.3 The Advantage of Pruning

This strategy has proved to be very appealing, however, because it does offer computational advantages over the simple composition approach. Under this regime every edge, not just the spanning roots, has its own constraints, and we can therefore determine the satisfiability of every edge as it is being constructed. If the constraint system is monotonic and the constraints for a particular edge are determined to be unsatisfiable, then that edge is discarded. The effect of this is to prune from the search space all edges that might otherwise have been constructed from unsatisfiable ones. This is illustrated in Figure 3, where S[ϕ] denotes the solution of ϕ, and X indicates that a solution is unsatisfiable. Since ϕ_1 is unsatisfiable, n_1 and n_2 never get built. Pruning n_1 and n_2 does not eliminate any valid solutions, since we know that their constraints would also have been unsatisfiable. Thus, by incrementally gathering and solving functional constraints, we can potentially eliminate from later consideration a number of trees exponential in sentence length. In some cases it may only take a polynomial amount of work to determine all solutions even though the phrasal constraints are exponentially ambiguous.

A familiar variation on the pruning strategy is to use the solutions associated with daughter constituents when computing a solution for a mother's constraints. This can have a significant effect, since it avoids recomputing the solutions to the daughters' constraints in the process of solving those of the mother. However, there is a technical issue that needs to be addressed. Since a daughter edge may be used by more than one mother, its solution cannot be changed destructively without the risk of introducing cross-talk between independent mothers. One way to avoid this is to copy the daughter solutions before merging them together, but this can be expensive. In recent years, there has been a great deal of attention devoted to this problem, and a number of different techniques have been advanced to reduce the amount of copying (Karttunen 1986; Wroblewski 1987; Godden 1990; Tomabechi 1991).

2.4 Still Exponential

Although pruning can eliminate an exponential number of trees, this strategy is still exponential in sentence length in the worst case when the grammar is exponentially ambiguous with few constituents that are actually pruned. There are two cases where few constituents are actually pruned. One is true ambiguity, as occurs with unrestricted prepositional phrase attachment. The grammar for PPs in English is well known to be exponentially ambiguous (Church and Patil 1982). If there are no functional or semantic restrictions on how the PPs attach, then none of the possibilities will be pruned and the interleaved pruning strategy, just like simple composition, will produce an exponential number of constituents spanning a string of prepositional phrases.

The other case where few constituents are actually pruned is when most candidate solutions are eliminated high in the tree, for example, because they are incomplete rather than inconsistent. In LFG (Kaplan and Bresnan 1982) functional constraints are incomplete when a predicate requires grammatical functions that are not realized in the string. (The requirement that predicate argument frames be completely filled is encoded in different but equivalent ways in other formalisms.) This can occur when, say, a verb requires a SUBJ and an OBJ, but the tree only provides a SUBJ. Since edges constructed from an incomplete edge may themselves be complete, incomplete edges cannot be discarded from the chart.

In sum, although the interleaved bottom-up strategy does permit some edges to be discarded and prunes the exponentially many trees that might be built on top of them, it does not in general eliminate the exponential explosion at the phrasal-functional interface. In fact, some researchers have observed that an augmented chart, even with interleaved pruning, may actually be worse than general constraint satisfaction algorithms because of the exponential space required to cache intermediate results (Varile, Damas, and van Noord, personal communications).

3. Exploitable Properties

Monotonicity is one of several constraint system properties that can be exploited to produce different interface strategies. Other properties include independence, conciseness, order invariance, and constraint system overlap. In the remainder of this section we discuss these properties and outline some techniques for exploiting them. In the following sections we give examples of interface algorithms that incorporate some of these techniques. Finally, we compare the performance of these algorithms on a sample grammar and some sample sentences.

3.1 Monotonicity

A system of constraints is *monotonic* if no deduction is ever retracted when new constraints are conjoined. This means that if ψ is unsatisfiable, then $\psi \wedge \phi$ is also unsatisfiable for arbitrary ϕ, so that ϕ can be completely ignored. This property is exploited, for instance, in unification algorithms that terminate as soon as an inconsistency is detected. In order for this to be a useful heuristic, it must be easy to determine that ψ is unsatisfiable and hard to solve $\psi \wedge \phi$. In the interleaved pruning strategy, determining that a constituent's constraints are unsatisfiable can be expensive, but this cost is often offset by the exponential number of edges that may be eliminated when a constituent is discarded. In general, the usefulness of the interleaved pruning strategy is determined by the fraction of edges that are pruned.

3.2 Independence

Two systems of constraints are *independent* if no new constraints can be deduced when the systems are conjoined. In particular, two disjunctions $\bigvee_i \phi_i$ and $\bigvee_j \psi_j$ are independent if there are no i, j and atomic formula χ such that $\phi_i \wedge \psi_j \rightarrow \chi$ and $\neg(\phi_i \rightarrow \chi)$ and $\neg(\psi_j \rightarrow \chi)$. If two systems of constraints are independent, then it can be shown that their conjunction is satisfiable if and only if they are both satisfiable in isolation. This is because there is no way of deriving false from the conjunction of any subconstraints if false was not already implied by one of those subconstraints by itself. Independence is most advantageous when the systems contain disjunctions, since there is no need to multiply into disjunctive normal form in order to determine the satisfiability of the conjunction. This can save an amount of work exponential in the number of disjunctions, modulo the cost of determining or producing independence.

One example of an algorithm that exploits independence is the context-free chart parser. Since sister constituents are independent of each other, their satisfiability can be determined separately. This is what makes a context-free chart parser polynomial instead of exponential. There are also several disjunctive unification algorithms that exploit independence, such as constraint unification (Hasida 1986; Nakano 1991), contexted unification (Maxwell and Kaplan 1989), and unification based on disjunctive feature logic (Dörre and Eisele 1990).

We say that a system of constraints is in *free-choice form* if it is a conjunction of independent disjunctions and all of the disjuncts are satisfiable. This means that we can freely choose one disjunct from each disjunction, and the result of conjoining these disjuncts together is guaranteed to be satisfiable. If recursively all of the disjuncts are also in free-choice form, then we have a *nested free-choice form*. The parse-forest representation for the chart discussed earlier is an example of a nested free-choice form. The advantage of such a form is that an exponential number of solutions (trees) can be represented in polynomial space. In general, any system of constraints in free-choice form can produce a number of solutions exponential in the size of the system. Each solution only requires a polynomial number of disjunctive choices to produce.

3.3 Conciseness

We say that a constraint system (or solution) is *concise* if its size is a polynomial function of the input that it was derived from. Most systems of constraints that have been converted to DNF are not concise, since in general converting a system of constraints to DNF produces a system that is exponential in the size of the original. Free-choice systems may or may not be concise. However, the constraint systems that tend to arise in solving grammatical descriptions are often concise when kept in free-choice form.

It is an important but often overlooked property of parse-forest representations of context-free charts that they are concise. All of the solutions of even an exponentially ambiguous context-free grammar can be represented in a structure whose size is cubic in the size of the input string and quadratic in the size of the grammar. So far, there has been little attention to the problem of developing algorithms for hybrid systems that exploit this property of the chart.

A constraint system may be made concise by *factoring* the constraints. A disjunction can be factored if there is a common part to all of its disjunctions. That is, the disjunction $(A \wedge \phi_1) \vee (A \wedge \phi_2) \vee ... (A \wedge \phi_n)$ can be reduced to $A \wedge (\phi_1 \vee \phi_2 \vee ... \phi_n)$. Another advantage of factoring is that under certain circumstances it can improve the effectiveness of the pruning and partitioning techniques mentioned above. For instance, suppose that two disjunctions are conjoined, one with factor A and the other with factor B, and that $A \wedge B \rightarrow$ FALSE. Then if A and B are factored out and processed before the residual disjunctions, then the disjunctions don't have to be multiplied out. In a similar manner, if A and B are independent of the residual disjunctions, and the residual disjunctions are also independent of each other, then factoring A and B out first would allow the problem to be partitioned into three independent sub-problems, and again the disjunctions would not have to be multiplied out. Thus under some circumstances, factoring can save an exponential amount of work. In Section 5 we discuss an interface algorithm based on factoring.

3.4 Order Invariance

Phrasal constraint systems and functional constraint systems commonly used for linguistic description have the property that they can be processed in any order without changing the final result. Although the order in which the constraints are processed doesn't change the result in any way, it can have a dramatic impact on how quickly solutions can be found or non-solutions discarded. Unfortunately, we do not know in advance which order will find solutions or discard non-solutions in the shortest amount of time, and so we depend on heuristics that choose an order that is thought more likely to evaluate solutions quickly. The question of processing order can be broken down into three parts: the order in which functional constraints are processed, the order in which phrasal constraints are processed, and the order in which functional and phrasal constraints are processed relative to one another.

There has been a lot of effort directed toward finding the best order for processing functional constraints. Kasper observed that separating constraints into disjunctive and nondisjunctive parts and processing the nondisjunctive constraints first can improve performance when the nondisjunctive constraints are unsatisfiable (Kasper 1987). It has also been observed that the order in which features are unified can have an effect, and that it is better to unify morpho-syntactic features before structural features. Both of these approaches reorder the constraints so that pruning is more effective, taking advantage of the monotonicity of functional constraints.

Research in context-free parsing has led to methods that can process phrasal constraints in any order and still maintain a polynomial time bound (e.g., Sheil 1976). However, in an interleaved strategy the order in which phrasal constraints are evaluated can make a substantial performance difference. This is because it determines the order in which the functional constraints are processed. The particular interleaved strategy discussed above effectively builds constituents and thus solves functional constraints in a bottom-up order. An alternative strategy might build constituents top-down and prune daughters whenever the collection of top-down functional constraints are unsatisfiable. It is also possible to process constituents in a head-driven order (Kay 1989) or to utilize an opportunistic islands-of-certainty heuristic (Stock, Falcone, and Insinnamo 1988).

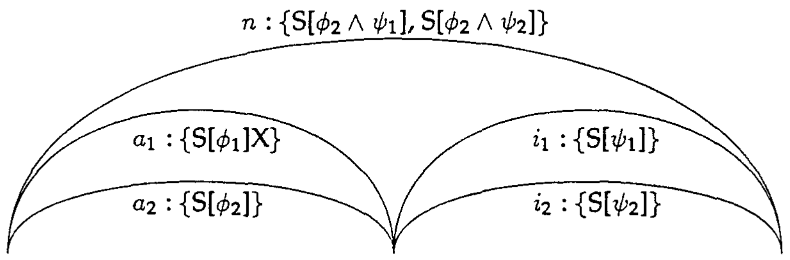

Figure 4
Noninterleaved pruning.

The relative processing order of phrasal and functional constraints is not as well-studied. There has been relatively uncritical acceptance of the basic interleaved arrangement. Another possibility might be to process all of the functional constraints before the phrasal constraints. An example of this kind of strategy is a semantic-driven algorithm, where subjects and objects are chosen from the string for their semantic properties, and then phrasal constraints are checked to determine whether the connection makes sense syntactically. In Section 4 we describe still another algorithm in which all of the phrasal constraints are processed before any of the functional constraints and discuss the advantages of this order.

3.5 Constraint System Overlap

As we mentioned in the introduction, the division between phrasal and functional constraints is somewhat fluid. All phrasal constraints can be converted into functional constraints, and some functional constraints can be converted into phrasal constraints. Turning all of the phrasal constraints into functional constraints obscures their special computational properties. On the other hand, turning all of the functional constraints into phrasal constraints is impractical even when possible because of the huge grammar that usually results. So it seems that the ideal is somewhere in between, but where? In Section 7, we observe that moving the boundary between phrasal and functional constraints can have a striking computational advantage in some cases.

4. Noninterleaved Pruning

We now consider a pruning strategy that does not interleave the processing of phrasal and functional constraints. Instead, all of the phrasal constraints are processed first, and then all of the functional constraints are collected and processed. This takes advantage of the fact that our constraint systems are order-invariant. In the first step, an unmodified context-free chart parser processes the phrasal constraints and produces a parse-forest representation of all the legal trees. In the second step, the parse-forest is traversed in a recursive descent starting from the root-spanning edge. At each edge in the parse forest the solutions of the daughter edges are first determined recursively and then combined to produce solutions for the mother edge. For each way that the edge can be constructed, the daughter solutions of that way are conjoined and solved. If a daughter edge has no solutions, then there is no need to extract the solutions of any remaining sisters. The resulting set of solutions is cached on the mother in case the mother is also part of another tree. This process is illustrated in Figure 4. Note

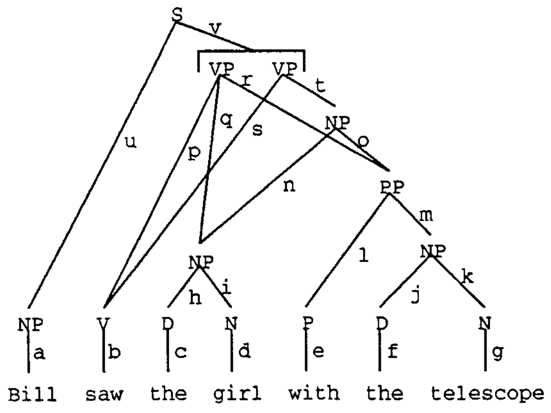

Figure 5
Parse forest.

that this strategy differs from simple composition in that the functional component operates on edges in the chart rather than individually enumerated trees.

The first step of this strategy is polynomial in sentence length since we can use a context-free algorithm that does not accumulate constraints for each constituent. The second step may be exponential since it does accumulate constraints for each edge and the constraints can encode all possible sub-trees for that edge. However, this method filters the functional computation using the global well-formedness of the phrase structure constraints. The performance can be significantly better than an interleaved approach if an exponentially ambiguous sub-tree fits into no complete parse tree. The disadvantage of this approach is that edges that might have been eliminated by the functional constraints have to be processed by the chart parser. However, this can at most add a polynomial amount of work, since the chart parser is in the worst case polynomial. Of course, this approach still incurs the overhead of copying, since it caches solutions on each edge.

5. Factored Extraction

We now examine an interface algorithm that is very different from both interleaved and noninterleaved pruning. Instead of focusing on pruning, this strategy focuses on factoring. We call this strategy a *factored extraction* strategy because it extracts a concise set of functional constraints from a chart and then passes the constraints to a constraint solver. Unlike the pruning strategies, constraints are not solved on an edge-by-edge basis: only the constraints for the spanning root edge are solved. Thus this is a noninterleaved strategy.

As with the noninterleaved pruning strategy, the first step is to build a chart based on the context-free grammar alone. This can be done in polynomial time using the active chart parser, and has the advantage of filtering constituents that are not part of some spanning tree for the sentence.

The second step is to extract the system of constraints associated with the spanning root edge. Consider the parse forest for the sentence *Bill saw the girl with the telescope* given in Figure 5. All of the constituents that are not part of a spanning tree have already been eliminated (for instance, the S that spans *Bill saw the girl*). The letters *a* through *v* represent lexical and grammatical constraints. For instance, *a* stands for the lexical constraints for *Bill* as an NP, and *u* stands for the grammatical constraint

$(f_S \text{ SUBJ}) = f_{NP(Bill)}$, indicating that the NP that dominates *Bill* is the subject of S. Structural ambiguity is represented by a bracket over the ambiguous constituents. In this case, there is only one structural ambiguity, the one between the VPs that span the string *saw the girl with the telescope*. They represent two different ways of attaching the PP; the first attaches it to *saw*, and the second attaches it to *girl*.

We extract the system of constraints for this sentence by starting from the S at the top and conjoining the result of recursively extracting constraints from its daughters. For constituents that are ambiguous, we *disjoin* the result of extracting the constraints of the ambiguous constituents. In addition, we cache the constraints of each node that we encounter, so that even if a node can be incorporated in more than one parse, we need only extract its constraints once. Note that since we are not caching solved constraints, there can be no cross-talk between constituents and copying is therefore not required. The result of this process is a re-entrant structure that is polynomial in the length of the string. If the re-entrant structure were expanded, it would produce the following:

$$a \wedge u \wedge [(b \wedge p \wedge c \wedge h \wedge d \wedge i \wedge q \wedge e \wedge l \wedge f \wedge j \wedge g \wedge k \wedge m \wedge r) \vee (b \wedge s \wedge c \wedge h \wedge d \wedge i \wedge n \wedge e \wedge l \wedge f \wedge j \wedge g \wedge k \wedge m \wedge o \wedge t)] \wedge v$$

However, instead of expanding the constraints, we make them smaller by factoring common elements out of the disjunctions. For instance, the b constraint is common to both disjuncts, and hence can be factored into the conjunctive part. Also, since the p and s constraints identically encode the relationship between the verb and the VP, they can also be factored. In general, we factor disjunctions on a node by node basis and cache the results on each node, to avoid repeating the factoring computation. Although a straight-forward implementation for factoring two sets of constraints would be quadratic in the number of edges, a linear factoring algorithm is possible if the constraints are sorted by string position and height in the tree (as they are in the example above). Factoring produces the following system of constraints:

$$a \wedge u \wedge b \wedge c \wedge h \wedge d \wedge i \wedge e \wedge l \wedge f \wedge j \wedge g \wedge k \wedge m \wedge p \wedge [(q \wedge r) \vee (n \wedge o \wedge t)] \wedge v$$

We can make factoring even more effective by doing some simple constraint analysis. In LFG, for example, the head of a constituent is usually annotated with the constraint $\uparrow=\downarrow$. This equality means that the head can be substituted for the mother without affecting satisfiability. This substitution tends to increase the number of common constraints, and thus increases the potential for factoring. In this example, q and t become the same since the NPs have the same head and n becomes tautologically true since its only function is to designate the head. This means that the disjunction can be reduced to just $r \vee o$:

$$a \wedge u \wedge b \wedge c \wedge h \wedge d \wedge i \wedge e \wedge l \wedge f \wedge j \wedge g \wedge k \wedge m \wedge p \wedge q \wedge (r \vee o) \wedge v$$

Thus the resulting system of constraints is completely conjunctive except for the question of where the PP attaches. This is the ideal functional characterization for this sentence. This approach produces an effect similar to Bear and Hobbs (1988), only without requiring special mechanisms. It also avoids the objections that Wittenburg and Barnett (1988) raise to a canonical representation for PP attachment, such as always attaching low. The only point at which special linguistic knowledge is utilized is the last step, where constraint analysis depends on the fact that heads can be substituted for mothers in LFG. Similar head-dependent analyses may also be possible for other grammatical theories, but factoring can make the constraint system substantially smaller even without this refinement.

Factoring is advantageous whenever a node participates in all of the sub-trees of another node. For example, this occurs frequently in adjunct attachment, as we have seen. It also occurs when a lexical item has the same category in all the parses of a sentence, which permits all the constraints associated with that lexical item to be factored out to the top level. Another advantage of the extraction algorithm comes from the fact that it does not solve the constraints on a per-edge basis, so that copying is not an issue for the phrasal-functional interface (although it still may be an issue internal to some functional constraint solvers).

The major disadvantage of factored extraction is that no pruning is done in the interface. This is left for the functional constraint solver, which may or may not know how to prune constraints based on their dependencies in the chart. Without pruning, the solver may do an exponential amount of futile work. In the next two sections we describe ways to get both pruning and factoring in the same algorithm.

6. Factored Pruning

It is relatively easy to add factoring to the noninterleaved pruning strategy. Remember that in that strategy the result of processing an edge is a disjunction of solutions, one for each alternative sequence of daughter edges. We can factor these solutions before any of them is used by higher edges (note that this is easier to do in a noninterleaved strategy than in an interleaved one). That is, if there are any common sub-parts, then the result will be a conjunction of these sub-parts with a residue of disjunctions. This is very similar to the factoring in factored extraction, except that we are no longer able to take advantage of the phrasally motivated groupings of constraints to rapidly identify large common sub-parts. Instead we must factor at the level of individual constraints, since the solving process tends to destroy these groupings.

The advantage of factored pruning over factored extraction is that we can prune, although at the cost of having to copy solutions. In the next section we will describe a complementary strategy that has the effect of adding pruning to factored extraction without losing its noncopying character.

7. Selective Feature Movement

So far we have examined how the properties of monotonicity, independence, conciseness, and order invariance can be exploited in the phrasal-functional interface. To conclude our discussion of interface strategies, we now consider how constraint system overlap can be exploited. As we have noted, many functional constraints can in principle be converted to phrasal constraints. Although converting all such functional constraints is a bad idea, it can be quite advantageous to convert some of them; namely, those constraints that would enable the context-free parser to prune the space of constituents.

Consider a grammar with the following two rules (using LFG notation [Kaplan and Bresnan 1982]):

$$\mathrm{S} \quad \longrightarrow \quad \left(\begin{array}{c} \mathrm{S}' \\ \downarrow \in (\uparrow \text{ADJUNCT}) \\ (\downarrow \text{COMPL}) = + \end{array} \right) \quad \begin{array}{c} \mathrm{NP} \\ (\uparrow \text{SUBJ}) = \downarrow \\ \end{array} \quad \begin{array}{c} \mathrm{VP} \\ \uparrow = \downarrow \\ \end{array}$$

Rule

$$S' \longrightarrow \left\{ \begin{array}{c} \text{COMP} \\ (\uparrow \text{COMPL}) = + \\ \text{e} \\ (\uparrow \text{COMPL}) = - \end{array} \right\} \begin{array}{c} \text{S} \\ \uparrow=\downarrow \end{array}$$

The first rule says that an S consists of an NP and a VP optionally preceded by an S′. The functional constraints assert that the functional structure corresponding to the NP is the SUBJ of the one corresponding to the S, the VP's f-structure is the head, and the f-structure of the S′ is an adjunct whose COMPL feature is +. According to the second rule, an S′ consists of an S optionally preceded by a COMP (the e stands for the empty string). If the COMP is present, then the COMPL feature will be +; otherwise it will be −. These rules allow for sentences such as *Because John kissed Sue, Mary was jealous,* but exclude sentences such as **John kissed Sue, Mary was jealous.*

The difficulty with these rules is that they license the context-free parser to postulate an initial S′ for a sentence such as *Bill drank a few beers.* This S′ will eventually be eliminated when its functional constraints are processed, because of the contradictory constraints on the value of the COMPL feature. An interleaved strategy would avoid building any edges on top of this spurious constituent (for example, an S with an initial adjunct). However, a noninterleaved strategy may build an exponential number of unnecessary trees on top of this S′, especially if such a string is the prefix of a longer sentence. If we convert the COMPL functional requirements into equivalent phrasal ones, the context-free parser will not postulate an initial S′ for sentences like these. This can be done by splitting the S′ rule into distinct categories $S'_{\text{COMPL}+}$ and $S'_{\text{COMPL}-}$ as follows:

Rule

$$S \longrightarrow \left(\begin{array}{c} S'_{\text{COMPL}+} \\ \downarrow\in (\uparrow \text{ADJUNCT}) \\ (\downarrow \text{COMPL}) = + \end{array} \right) \begin{array}{c} \text{NP} \\ (\uparrow \text{SUBJ}) = \downarrow \end{array} \begin{array}{c} \text{VP} \\ \uparrow=\downarrow \end{array}$$

Rule

$$S'_{\text{COMPL}+} \longrightarrow \begin{array}{c} \text{COMP} \\ (\uparrow \text{COMPL}) = + \end{array} \begin{array}{c} \text{S} \\ \uparrow=\downarrow \end{array}$$

Rule

$$S'_{\text{COMPL}-} \longrightarrow \begin{array}{c} \text{e} \\ (\uparrow \text{COMPL}) = - \end{array} \begin{array}{c} \text{S} \\ \uparrow=\downarrow \end{array}$$

With these rules the context-free parser would fail to find an $S'_{\text{COMPL}+}$ in the sentence *Bill drank a few beers.* Thus the S with an initial adjunct and many otherwise possible trees would never be built. In general, this approach notices local inconsistencies in the grammar and changes the categories and rules to avoid encountering them.

Moving features into the constituent space has the effect of increasing the number of categories and rules in the grammar. In the worst case, the size of the chart grows linearly with the number of categories, and computation time grows quadratically in the size of the grammar (Younger 1967; Earley 1970). Just considering the cost of phrasal processing, we have increased the grammar size and therefore have presumably made the worst case performance worse. However, if features are carefully selected so as to increase the amount of pruning done by the chart, the net effect may be that even though the grammar allows more types of constituents, the chart may end up with fewer instances.

It is interesting to compare this technique to the restriction proposal in Shieber (1985). Both approaches select functional features to be moved forward in processing order in the hope that some processing will be pruned. Shieber's approach changes the processing order of functional constraints so that some of them are processed top-down instead of bottom-up. Our approach takes a different tack, actually converting some of the functional constraints into phrasal constraints. Thus Shieber's does its pruning using functional mechanisms whereas our approach prunes via standard phrasal operations.

8. Some Performance Measures

In the foregoing sections we outlined a few specific interface strategies, each of which incorporates a different combination of techniques for exploiting particular constraint system properties. We argued that each of these techniques can make a substantial performance difference under certain circumstances. In this section we report the results of some preliminary computational comparisons that we conducted to determine whether these techniques can make a practical difference in parsing times. Our results are only suggestive because the comparisons were based on a single grammar and a small sample of sentences. Nevertheless, the patterns we observed are interesting in part because they reinforce our intuitions but also because they lead to a deeper understanding of the underlying computational issues.

We conducted our comparisons by first fixing a base grammar and 20 test sentences and then varying along three different dimensions. The LFG grammar was developed by Jeff Goldberg and Annie Zaenen for independent purposes and came to our attention because of its poor performance using previously implemented algorithms. The test sentences were derived from a compiler textbook and are given in the appendix. One dimension that we explored was selective feature movement. We produced a descriptively equivalent variation of the base grammar by choosing certain functional constraints to move into the phrasal domain. A second dimension was the choice of strategy. We compared the interleaved pruning, noninterleaved pruning, factored pruning, and factored extraction strategies discussed above. As a final dimension we compared two different unification algorithms.

8.1 Grammar Variants

The Goldberg-Zaenen base grammar was designed to have broad coverage over a set of complex syntactic constructions involving predicate-argument relations. It does not handle noun–noun compounds, and so these are hyphenated in the test sentences. The grammar was written primarily to capture linguistic generalizations, and little attention was paid to performance issues. We measured performance on the 20 test sentences using this grammar in its original form. We also measured performance on a variant of this grammar produced by converting certain function requirements into phrasal constraints. We determined which constraints to move by running the interleaved pruning strategy on the base grammar and identifying which constraints caused constituents to be locally unsatisfiable. We then modified the grammar and lexicon by hand so that those constraints were reflected in the categories of the constituents. Examination of the results prompted us to split five categories:

- VP was split into VP_{INF+} and VP_{INF-}, where $(\uparrow \text{INF}) = +$ is true of VP_{INF+}, and $(\uparrow \text{INF}) \neq +$ is true of VP_{INF-}.

Table 1
Strategies and techniques.

Strategy	Interleaving	Per-edge solving	Pruning	Factoring
Simple composition	—	—	—	—
Interleaved pruning	yes	yes	yes	—
Non-interleaved pruning	—	yes	yes	—
Factored pruning	—	yes	yes	yes
Factored extraction	—	—	—	yes

- V was split into V_{AUX}, V_{OBL}, V_{TRANS}, and V_{OTHER}, where V_{AUX} is an auxiliary verb, V_{OBL} is a verb with an oblique argument, V_{TRANS} is a transitive verb, and V_{OTHER} is anything else.
- N was split into N_{OBL+} and N_{OBL-}, where N_{OBL+} takes an oblique argument and N_{OBL-} does not.
- COMP was split into $COMP_{COMPL+}$ and $COMP_{COMPL-}$, where $COMP_{COMPL+}$ has $(\uparrow \text{COMPL}) = +$ and $COMP_{COMPL-}$ has $(\uparrow \text{COMPL}) = -$.
- PP was split into PP_{PRED} and PP_{PCASE}, where PP_{PRED} has a predicate and PP_{PCASE} has a PCASE (is used as an oblique argument).

All of these splits were into mutually exclusive classes. For instance, in the PP case every use of a preposition in the grammar had either a PCASE or a predicate but not both.

8.2 Strategy Variants

Table 1 summarizes the combination of techniques used in the strategies we have mentioned in this paper. The simple composition strategy is the naive first implementation discussed in the introduction; it is included in the table only as a point of reference. Factored extraction is the only other interface strategy that does not do per-edge solving and caching, and therefore does not require a special copying algorithm. Obviously, the listed strategies do not instantiate all possible combinations of the techniques we have outlined. In all the strategies we use an active chart parser for the phrasal component.

8.3 Unifier Variants

Unification is a standard technique for determining the satisfiability of and building attribute-value models for systems of functional constraints with equality. In recent years there has been a considerable amount of research devoted to the development of unification algorithms that perform well when confronted with disjunctive constraint systems (Hasida 1986; Maxwell and Kaplan 1989; Dörre and Eisele 1990; Nakano 1991). Some of these unifiers take advantage of the same properties of constraint systems that we have discussed in this paper. For example, Kasper's algorithm takes advantage of monotonicity and order invariance to achieve improved performance when pruning is possible. It works by first determining the satisfiability of the conjunctive constraints, and then checking disjuncts one at a time to find those that are inconsistent with the conjunctive part. Finally, the disjuncts that remain are multiplied into DNF. Our contexted unification algorithm (Maxwell and Kaplan 1989) also allows for pruning but

Table 2
Mean scaled computation time.

Grammar	Strategy	Benchmark	Contexted
Base	Interleaved pruning	100	42
	Noninterleaved pruning	71	25
	Factored pruning	—	23
	Factored extraction	>1000	>1000
Modified	Interleaved pruning	38	26
	Noninterleaved pruning	29	19
	Factored pruning	—	13
	Factored extraction	21	7

in addition takes advantage of independence to achieve its performance. It works by objectifying the disjunctions so that the constraints can be put into conjunctive normal form (CNF). This algorithm has the advantage that if disjunctions are independent, they do not have to be multiplied out. These unifiers depend on different properties, so we have included both variants in our comparisons to see whether there are any interactions with the different interface strategies. In the discussion below, we call the unifier that we implemented based on Kasper's technique the "benchmark" unifier.

8.4 Results and Discussion

We implemented each of the four strategies and two unifiers in our computational environment, except that, because of resource limitations, we did not implement factored pruning for the benchmark unifier. We then parsed the 20 test sentences using the two grammars for each of these configurations. We measured the compute time for each parse and averaged these across all the sentences. The results are shown in Table 2. To make comparisons easier, the mean times in this table have been arbitrarily scaled so that the mean for the interleaved pruning strategy with the benchmark unifier is 100.

The most striking aspect of this table is that it contains a wide range of values. We can conclude even from this limited experiment that the properties and techniques we have discussed do in fact have practical significance. The strategy in the fourth line ran much longer than we were willing to measure, while every other combination behaved in a quite reasonable way. Since the fourth line is the only combination that does neither functional nor phrasal pruning, this demonstrates how important pruning is.

Looking at the grammar variants, we see that in all cases performance is substantially better for the modified grammar than for the base grammar. This is in agreement with Nagata 1992's finding that a medium-grain phrase structure grammar performs better than either a coarse-grain or fine-grain grammar. The modified grammar increases the amount of pruning that is done by the chart because we carefully selected features for this effect. The fact that this improves performance for even the pruning strategies is perhaps surprising, since the same number of inconsistencies are being encountered. However, with the modified grammar the inconsistencies are being encountered earlier, and hence prune more. This effect is strongest for the factored extraction algorithm since inconsistencies are never detected by the interface; they are left for the unifier to discover.

Turning to the interface strategies, we see that noninterleaved pruning is always

Table 3
Maximum scaled computation time.

Grammar	Strategy	Benchmark	Contexted
Base	Interleaved pruning	691	314
	Noninterleaved pruning	421	182
	Factored pruning	—	135
	Factored extraction	>20000	>20000
Modified	Interleaved pruning	112	104
	Noninterleaved pruning	101	74
	Factored pruning	—	43
	Factored extraction	126	15

better than interleaved pruning. This is also as expected, because the noninterleaved strategy has the benefit of global phrasal pruning as well as incremental functional pruning. Nagata (1992) reports similar results with early and late unification. Noninterleaved pruning is not as efficient as factored pruning, however. This shows that factoring is an important technique once the benefits of pruning have been obtained. The factored extraction strategy exhibits the most interesting pattern of results, since it shows both the worst and the best performance in the table. It gives the worst performance with the base grammar, as discussed above. It gives the overall best performance for the modified grammar with the contexted unifier. This takes advantage of the best arrangement for pruning (in the chart), and its contexted unifier can best operate on its factored constraints. The next best performance is the combination of factored pruning with the modified grammar and the contexted unifier. Although both strategies take advantage of factoring and pruning, factored pruning does worse because it must pay the cost of copying the solutions that it caches at each edge.

Finally, the type of unifier also made a noticeable difference. The contexted unifier is always faster than the benchmark one when they can be compared. This is to be expected because, as mentioned above, the contexted unifier both prunes and takes advantage of independence. The benchmark unifier only prunes.

Average computing time is one way of evaluating the effects of these different combinations, since it gives a rough performance estimate across a variety of different sentences. However, the degree of variability between sentences is also important for many practical purposes. A strategy with good average performance may be unacceptable if it takes an unpredictably large amount of time on some sentences. Table 3, which shows the computing time of the worst sentence in each cell, gives a sense of the inter-sentence variability. These values use the same scale as Table 2.

This table supports roughly the same conclusions as Table 2. There is a wide range of values, the modified grammar is better than the base, and the contexted unifier is faster than the benchmark one. In many cells, the maximum values are substantially larger than the corresponding means, thus indicating how sensitive these algorithms can be to variations among sentences. There is an encouraging result, however. Just as the lowest mean value appears for factored extraction with the modified grammar and contexted unifier, so does the lowest maximum. Moreover, that cell has the lowest ratio of maximum to mean, almost 2. Thus, not only is this particular combination the fastest, it is also much less sensitive to variations between sentences. However, factored extraction is very sensitive to the amount of pruning done by the phrasal constraints, and thus may not be the best strategy when it is impractical to perform appropriate grammar modifications. In this situation, factored pruning may be the best choice because it is almost as fast as factored extraction but is much less sensitive to grammar variations.

9. Concluding Remarks

As we discussed in the introduction, the interleaved pruning strategy is substantially better than simple composition and so it is no surprise that it is a widely used and little questioned interface strategy. However, it is only one point in a complex and multidimensional space of possibilities, and not necessarily the optimal point at that. We outlined a number of alternative strategies, and presented preliminary measurements to suggest that factored extraction may give better overall results, although it is very sensitive to details of the grammar. Factored pruning also gives good results and is less sensitive to the grammar. The good results of these two strategies show how important it is to take advantage both of monotonicity and independence and of the polynomial nature of the phrasal constraints.

The investigations summarized in this paper suggest several directions for future research. One direction would aim at developing a grammar compiler that automatically selects and moves the best set of features. A compiler could hide this transformation from the grammar developer or end user, so that it would be considered merely a performance optimization and not a change of linguistic analysis. Another research direction might focus on a way of adding functional pruning to the factored extraction algorithm so that it would be less sensitive to variations in the grammar.

At a more general level, our explorations have illustrated the richness of the space of phrasal-functional interface possibilities, and the potential value of examining these issues in much greater detail. Of course, further experimental work using other grammars and larger corpora are necessary to confirm the preliminary results we have obtained. We also need more formal analyses of the computation complexity of interface strategies to support the intuitive characterizations that we have presented in this paper. We believe that the context-free nature of phrasal constraints has not yet been fully exploited in the construction of hybrid constraint processing systems and that further research in this area can still lead to significant performance improvements.

References

Barton, G. Edward; Berwick, Robert C.; and Ristad, Eric Sven (1987). *Computational Complexity and Natural Language.* The MIT Press.

Bear, John, and Hobbs, Jerry R. (1988). "Localizing expression of ambiguity." In *Proceedings, Second Conference on Applied Natural Language Processing.* 235–241.

Church, Kenneth W., 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.

Dörre, Jochen, and Eisele, Andreas (1990). "Feature logic with disjunctive unification." In *Proceedings, COLING-90.*

Earley, J. (1970). "An efficient context-free algorithm." *Communications of the ACM,* 13, 94–102.

Gazdar, Gerald; Klein, Ewan; Pullum, Geoffrey; and Sag, Ivan. (1985). *Generalized Phrase Structure Grammar.* Harvard University Press.

Godden, K. (1990). "Lazy unification." In *Proceedings of the 28th Annual Meeting of the ACL.*

Hasida, K. (1986). "Conditioned unification for natural language processing." In *Proceedings of COLING-86,* 85–87.

Johnson, Mark. (1989). The computational complexity of Tomita's algorithm." In *Proceedings, International Workshop on Parsing Technologies.* 203–208.

Kaplan, Ronald M. (1973) "A multi-processing approach to natural language." In *Proceedings, 1973 National Computer Conference.* Montvale, N.J., 435–440.

Kaplan, Ronald M., and Bresnan, Joan (1982). "Lexical-functional grammar: A formal system for grammatical representation." In *The Mental Representation of Grammatical Relations,* edited by Joan Bresnan, 173–281. MIT Press.

Karttunen, Lauri (1986). "D-PATR: A development environment for unification-based grammars." In *Proceedings, COLING-86,* Bonn, Germany.

Kasper, Robert (1987). "A unification method for disjunctive feature descriptions." In *Proceedings, 25th Annual Meeting of the ACL.*

Kay, Martin (1979). "Functional Grammar." In *Proceedings, 5th Annual Meeting of the Berkeley Linguistic Society.*

Kay, Martin (1980). "Algorithm schemata and data structures in syntactic processing." In *Readings in Natural Language Processing,* edited by Barbara J. Grosz, Karen Sparck-Jones, and Bonnie Lynn Webber, 35–70. Morgan Kaufmann.

Kay, Martin (1989). "Head-driven parsing." In *Proceedings, International Workshop on Parsing Technologies.* 52–62.

Knight, Kevin (1989). "Unification: A multidisciplinary survey." *ACM Computing Surveys,* 21(1), 93–124.

Maxwell, John T. III, and Kaplan, Ronald M. (1989). "An overview of disjunctive constraint satisfaction." In *Proceedings, International Workshop on Parsing Technologies.*

Nagata, Masaaki (1992). "An empirical study on rule granularity and unification interleaving toward an efficient unification-based parsing system." In *Proceedings, COLING-92.* 177–183.

Nakano, Mikio (1991). "Constraint projection: An efficient treatment of disjunctive feature descriptions." In *Proceedings, 29th Annual Meeting of the ACL.* 307–314.

Nelson, Greg, and Oppen, Derek C. (1980). "Fast decision procedures based on congruence closure." *Journal of the ACM,* 27(3), 356–364.

Pereira, Fernando C. N., and Warren, David H. D. (1980). "Definite clause grammars for language analysis—a survey of the formalism and a comparison with augmented transition networks." *Artificial Intelligence,* 13(3), 231–278.

Pollard, Carl, and Sag, Ivan (1987). *Information-Based Syntax and Semantics.* CSLI Lecture Notes, Volume 13. Stanford, CA.

Schabes, Yves (1991). "Polynomial time and space shift-reduce parsing of arbitrary context-free grammars." In *Proceedings, 29th Annual Meeting of the ACL.* 106–113.

Sheil, Beau (1976). "Observations on context-free parsing." In *Proceedings, COLING-76.*

Shieber, Stuart (1985). "Using restriction to extend parsing algorithms for complex feature-based formalisms." In *Proceedings, 23rd Annual Meeting of the ACL.*

Stock, Oliviero; Falcone, Rino; and Insinnamo, Patrizia (1988). "Island parsing and bidirectional charts." In *Proceedings, COLING-88.* 636–641.

Thompson, Henry (1983). "MCHART: a flexible, modular chart parsing system." In *Proceedings, AAAI-83.* 408–410.

Tomabechi, Hideto (1991). "Quasi-destructive graph unification." In *Proceedings, Second International Workshop on Parsing Technology.* 164–171.

Tomita, Masaru (1986). *Efficient Parsing for Natural Language.* Kluwer Academic Publishers.

Wittenburg, Kent, and Barnett, Jim (1988). "Canonical representation in NLP systems design." In *Proceedings, Second Conference on Applied Natural Language Processing.* 253–259.

Wroblewski, David A. (1987). "Nondestructive graph unification." In *Proceedings, AAAI-87.*

Younger, D. H. (1967). "Recognition and parsing of context-free languages in time n^3." *Information and Control,* 10, 189–208.

Appendix A: Test Sentences

1. These normally include syntactic analyses.
2. The phases are largely independent of the target-machine.
3. Those phases depend primarily on the source-language.
4. Code-optimization is done by the front-end as well.
5. However there has been success in this direction.
6. Often the phases are collected into a front-end.
7. Generally these portions do not depend on the source-language.
8. The front-end consists of those phases that depend primarily on the source-language.
9. If the back-end is designed carefully it may not be necessary to redesign the back-end.
10. It produces a compiler for the same source-language on a different machine.
11. It has become fairly routine to take the front-end of a compiler.
12. It is not even necessary to redesign much of the back-end.
13. The front-end consists of those phases that depend primarily on the source-language.
14. It is also tempting to compile several different languages into the same intermediate language.
15. The back-end also includes those portions of the compiler that depend on the target-machine.
16. This matter is discussed in Chapter 9.
17. The front-end also includes the error-handling that goes along with these phases.
18. It is tempting to use a common back-end for the different front-ends.
19. Because of subtle differences in the viewpoints of the different languages there has been only limited success in that direction.
20. It has become routine to redo its associated back-end to produce a compiler for the same source-language on a different machine.