Unification-based Parsers that Automatically Take Advantage of Context Freeness*

John T. Maxwell III† and Ronald M. Kaplan‡
Palo Alto Research Center§

Abstract

Although parsing with context-free phrase structure rules can be done in worst-case cubic time, it is well-known that adding feature constraints can make the resulting system exponential in the worst case, if not undecidable. In this paper we observe that the standard algorithms for combining feature constraints with context-free phrase structure rules can be exponential even when the combination is context-free equivalent. We then introduce two new algorithms based on *disjunctive lazy copy links* that automatically take advantage of simple context-freeness in such grammars and parse in cubic time. The algorithms described do not require that the grammar be pre-analyzed or compiled; instead, the computational properties are a by-product of how the algorithms work. Consequently, the algorithms improve performance even for grammars that are not altogether context-free equivalent, since they will automatically take advantage of situations where the amount of information flowing upward from a constituent to any of its consumers is bounded.

1 Introduction

Many grammatical formalisms use hierarchical feature structures to describe the syntactic structure of natural language utterances. For instance, Lexical-Functional Grammar (Kaplan and Bresnan 1982), Functional Unification Grammar (Kay 1979), and HPSG (Pollard and Sag 1987) all use hierarchical feature structures as a major component of grammatical descriptions. Definite Clause Grammars (Pereira and Warren 1980) use what can be regarded as an equivalent representation as well. Feature structures have the advantage of being easy to understand and easy to implement, given unification programming languages such as Prolog. However, they have the disadvantage of making the resulting grammatical formalisms difficult to parse efficiently, both in theory and in practice.

*originally presented at the LFG96 conference in Grenoble as "An Efficient Parser for LFG"

†maxwell.parc@xerox.com
‡kaplan.parc@xerox.com
§3333 Coyote Hill Rd, Palo Alto, CA 94304

On the theory side, grammatical formalisms that use arbitrary hierarchical feature structures can be undecidable in the worst case (Kaplan and Bresnan 1982). Even with suitable restrictions, such as the off-line parsability constraint of LFG, the formalisms can be NP complete (Barton, Berwick, and Ristad 1987). Although in practice the sorts of phenomena that make a formalism take exponential time are rare, it is not uncommon for untuned unification parsers to take minutes to parse a relatively simple sentence.

There have been a number of different approaches over the years to making unification parsers run faster. One approach has been to focus on making unification itself faster. In addition to the general work on efficient unification (Knight 1989), there has been work within the computational linguistics community on undoable unification (Karttunen 1986), lazy copying (Godden 1990), and combinations thereof (Tomabechi 1991).

Another approach has been to focus on the problem of disjunction and to propose techniques that handle special cases of disjunction efficiently. For instance, disjuncts that are inconsistent with a non-disjunctive part of a formula can be eliminated early (Kasper 1987). Also, certain types of disjunctions can be efficiently processed by being embedded within the feature structure (Karttunen 1984; Bear 1988; Maxwell and Kaplan 1989; Dörre and Eisele 1990).

There has also been some work on the interaction between feature constraints and the context-free component of grammatical formalisms. It has long been known that some feature constraints can be compiled into a refined set of categories in the context-free component of the grammar (Gazdar, Klein, Pullum, and Sag 1985). Nagata (1992) showed that the granularity of phrase structure rules has an important effect on parsing efficiency. In particular, grammars with medium-grained categories performed better than grammars with coarse-grained or fine-grained categories. Also, Maxwell and Kaplan (1993) showed that the strategies used for handling the interaction between the phrasal and functional components can make a surprising performance difference.

This paper takes a different tack on the performance problem. Our approach is a continuation of our earlier investigations (Maxwell and Kaplan 1989, 1993), but here we focus on the fine details of the interaction between phrasal and functional constraints. This work is based on the observation that the standard methods for parsing with a unification grammar can be exponential even when the grammar is context-free equivalent. We say that a unification grammar is *context-free equivalent* when the unification grammar can be easily transformed into a finite set of ordinary context-free phrase structure rules that accepts exactly the same language. We know that a sentence of length n can be parsed with a set of context-free phrase structure rules in at most $O(n^3)$ time. However, parsing with a unification grammar that is context-free equivalent can sometimes take $O(2^n)$ time. Since we know that the grammar is context-free equivalent, this exponential result cannot be due to the inherent complexity of the linguistic phenomena that the grammar accounts for. Instead, it must be caused by some deficiency in the strategy used by standard parsing methods. In this paper we investigate the nature of this deficiency and propose two new algorithms that automatically take advantage of context-freeness in context-

free equivalent grammars. We then show that the new algorithms also improve performance for situations in irreducibly non-context-free grammars where the amount of information flowing upward from a constituent to any of its consumers is bounded.

We are not the first to observe that parsing with context-free equivalent grammars should take at most cubic time regardless of the expressive power of the formalism. Although Tree Adjoining Grammars take $O(n^6)$ time in the worst case to parse a sentence, it has been shown that TAGs that are of a certain form can be parsed in $O(n^3)$ time, conforming to the linguistic complexity of the phenomena they account for (Schabes and Waters 1993; Rogers 1994). At a superficial level our work differs from theirs because of the difference in nature between feature structure unification and tree adjunction. A more important difference is that our algorithms take advantage of context-freeness automatically, wherever it occurs. This means that even if a grammar is not completely context-free equivalent, the algorithms will still improve performance if there are situations where the amount of information flowing upward from a constituent to any of its consumers is bounded. The TAG algorithms described in (Schabes and Waters 1993; Rogers 1994) only work for grammars that are completely context-free equivalent. Other advantages of our algorithms are that they do not require a different mechanism for grammars that are not context-free equivalent, and they do not depend on a pre-analysis or special compilation of the grammar.

order	10	20	30	40	50
$O(n)$	.01 sec.	.02 sec.	.03 sec.	.04 sec.	.05 sec.
$O(n^2)$	.1 sec.	.4 sec.	.9 sec.	1.6 sec.	2.5 sec.
$O(n^3)$	1 sec.	8 sec.	27 sec.	64 sec.	125 sec.
$O(n^6)$	17 min.	18 hours	8 days	47 days	180 days
2^n	1 sec.	17 min.	12 days	35 years	357 centuries

Figure 1: How bad is exponential?

Before continuing, it is worth reviewing how bad an exponential can be. Figure 1 shows how various functions of n behave for sentences up to length 50. The starting assumption is that the parser can parse one word in a millisecond. Notice first that at $n = 10$ there is virtually no difference between exponential and cubic. It is only for longer sentences that the difference begins to show up. At $n = 50$, the difference is astronomical. The second observation is that $O(n^6)$, although polynomial, is not practical if the parser is n^6 on every input, not just in the worst case. The final thing to notice is the difference between $O(n^3)$ and $O(n)$ at $n = 50$. This explains the growing interest in (presumably linear) finite-state approximations for syntactic analysis (assuming that the resulting state machine is of tractable size).

In the following sections we first describe a standard approach to parsing with unification grammars and show why it is exponential for grammars that

are context-free equivalent. Then we review how lazy copy links work, since understanding them is crucial to understanding the algorithms that we later introduce. Next, we present *lazy DNF unification*, a fairly simple algorithm for automatically taking advantage of context-freeness. Then we introduce *lazy contexted unification*, a more sophisticated algorithm that packs alternatives even more than the lazy DNF algorithm, which can reduce the effective grammar constant. Finally, we give some performance results for industrial-strength grammars for English, French, and German.

2 The Problem

There are well-known methods for parsing with a context-free phrase structure grammar in time proportional to the cube of the length of a sentence. However, when a grammar has feature structure constraints in addition to its context-free skeleton, then a typical parser will often take an exponential amount of time to analyze a sentence. This is true even if the unification grammar is context-free equivalent, i.e. a new strictly context-free grammar can be produced that accepts the same sentences as the original but without using feature structures. In order to see why this is so, we need to first understand how a parser can parse with a context-free grammar in cubic time using a chart, and why adding unification to a convential chart parser makes the parser exponential even for grammars that are context-free equivalent.

2.1 The Chart

A chart is simply a data structure for caching the constituents that have already been constructed by a parser. The main advantage of a chart is that it allows the parser to reuse previously recognized constituents as it tries to parse the sentence in different ways (Sheil 1976). If the grammar is context-free, then the parser need not record how constituents get constructed, only that they are constructable. For instance, the parser must determine whether there is an NP that goes from the fifth word to the tenth word, but it need not record how many PPs the NP has in it. Because of this, there are only $O(Cn^2)$ different constituents that might be constructed for a sentence of length n, where C is the number of different categories that the grammar allows (The n^2 comes from the cross-product of all possible word positions). Conceptually, the chart is just a three-dimensional array of (Category, Left Position, Right Position) that indicates whether or not there is a constituent of type Category that starts at the Left Position and ends at the Right Position. A sentence has a parse if there is an S category that begins at the beginning of the sentence and ends at the end of the sentence.

One way to fill in the chart is to start with all the one word constituents, then build two word constituents out of pairs of adjoining one word constituents, then build three word constituents out of pairs of adjoining one and two word constituents, and so on, with each level building on the results of the previous

levels. This is called the CKY algorithm (Younger 1967). The reason that the algorithm is $O(n^3)$ rather than $O(n^2)$ is that each of the constituents can be built in multiple ways. In the worst case, a constituent that is $O(n)$ in size can be built in $O(n)$ different ways by considering all of the intermediate positions at which the constituent can be split into two daughters.[1] To build $O(n^2)$ constituents each in $O(n)$ ways requires $O(n^3)$ time.

The chart parser described so far is acting just as an acceptor. It determines whether or not a given sentence belongs to the language of the grammar, but it does not produce the sentence's valid parse trees. However, there is a simple extension to provide this additional information. Whenever a "mother" constituent is constructed out of a sequence of "daughter" sub-constituents, the construction is recorded as a local subtree on the mother constituent. A chart annotated with such subtrees is called a "parse forest". When the parser is done, a particular parse tree can be read out by starting at the S constituent that spans the whole sentence, and arbitrarily picking one subtree. Then for each daughter constituent in the subtree, one of its subtrees is arbitrarily picked. This process is carried on until the tree is fully specified. In general, there can be exponentially many such fully-specified trees, but the parse forest for them can be produced in cubic time because they are stored in a compact representation.

Thus the parsing process is divided into two phases: a chart construction phase, and a parse-tree read out phase. The chart construction phase for a context-free grammar runs in worst-case cubic time. The read out phase takes linear time for each of the possible parse trees. In order to be able to say that we can parse in cubic time, parse trees cannot be filtered during the read out phase. Instead, every possible choice at every possible point in a top-down enumeration of the chart must lead to a valid tree.

If there are only a few (e.g. a polynomial number of) valid parse trees, then the whole process runs in polynomial time. On the other hand, for an exponentially ambiguous sentence the time to read out all of its parse trees will, of course, be exponential. However, it is not always necessary to read out all the parse trees to obtain a useful result. For instance, if we only want to pick out the parse tree with the best score according to some metric, we may be able to pick out that tree by making local choices within the chart instead of applying the metric to each tree. This is what Stochastic Context-Free Grammars do using the Inside-Outside algorithm (Lari and Young, 1990).

We can construct in cubic time a parse-forest chart that represents an exponential number of parse trees because constituents are grouped into equivalence classes according to what part of a sentence they span and what their top categories are. Each triple of (Category, Left Position, Right Position) represents a (possibly exponential) number of different parse trees that are indistinguishable from the point of view of possible consumers. The equivalence classes abstract away from the irrelevant details of how the constituents are constructed, and subsequent parsing decisions are based just on these equivalence classes and

[1] This assumes that the grammar is in Chomsky Normal Form, so that all of its rules are at most binary branching.

not their individual members. These notions are crucial to how our unification parsers will automatically take advantage of context-freeness. To foreshadow some of the terminology that we use below, we call the equivalence classes "external solutions", the subtrees that make up an equivalence class "internal solutions", and we say that the equivalence classes are "opaque" since their internal details do not influence the construction of larger constituents.

Another important observation about context-free parse-forest charts is that they bound the amount of work done for each subtree. This means that since there are only $O(n^3)$ subtrees in the worst case, the whole chart takes only $O(n^3)$ time to produce in the worst case. Our new parsing algorithms build off of this chart parser architecture. We can show that our algorithms parse in $O(n^3)$ time in the worst case when a grammar is context-free equivalent by proving that they do a bounded amount of work per subtree. This is an important insight that allows us to focus on processing at individual subtrees instead of having to pay attention to the properties of the entire parse-forest chart.

2.2 Feature Structures

Most feature structure grammar formalisms add the feature structure constraints to a backbone of context-free phrase structure rules. Lexical-Functional Grammar (Kaplan and Bresnan 1982) and PATR (Karttunen 1986), for example, are very explicit in assigning both a phrase-structure tree and an attribute-value functional structure to every sentence of a language. For Functional Unification Grammar (Kay 1979) and other unification formalisms (such as HPSG (Pollard and Sag 1987)), the phrase structure is more implicit, showing up as the record of the control strategy that recursively re-instantiates the collection of attribute-value constraints from the grammar.

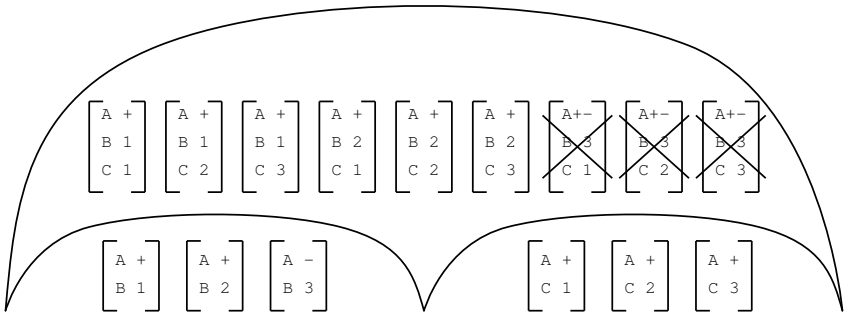

Figure 2: DNF approach to handling feature structures

One way to parse with such a feature structure grammar is first to build a context-free phrase structure chart using standard context-free algorithms, and then to make a second pass over the chart data structure, building feature

structures bottom-up (see Maxwell and Kaplan 1993 for some advantages and disadvantages of this approach). First, feature structures are instantiated from lexical items according to their feature constraints. If the lexical items are ambiguous in any way, this will produce multiple feature structures. Then, feature structures for a mother constituent are constructed by taking the cross-product of the feature structures that belong to the daughter constituents and unifying (or conjoining) each combination. If unification fails or the conjunction is inconsistent for any combination, that combination is thrown away. The result is a set of feature structures in disjunctive normal form (DNF) that are consistent to this point (see Figure 2). If there is more than one way of constructing the mother constituent out of daughter constituents (e.g. there is more than one subtree), then the sets of feature structures produced from all of the analyses are unioned together. This process continues bottom-up until feature structures for all of the constituents have been constructed. This is exponential in the worst case because of the cross-product that occurs at each level. For instance, if each lexical item is ambiguous with two feature structures, then every valid parse tree can be associated with $O(2^n)$ different feature structures.

During the cross-product step discussed above, it may happen that the same feature structure is produced more than once. When this happens, it is sufficient to put only one of the instances in the mother's set of feature structures. This is because further computations only depend on what the feature structure is, not on how it was derived.

If the grammar only specifies finite-valued features, then a parser can be made to run in cubic time. This is because there is only a finite number of possible feature structures. In particular there are only a finite number of different feature structures at each level (if we eliminate duplicate occurrences), and therefore the number of cross-product unifications at each level is bounded. Since each unification step is bounded, the total amount of work per subtree is also bounded and the parser retains the worst case cubic bound of the context-free chart construction phase.

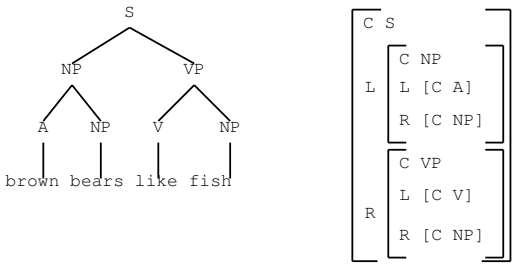

Figure 3: A context-free equivalent grammar

We have seen that finite-valued features thus provide a richer notation for filtering the trees that the context-free rules would otherwise permit, but they

do not change the complexity of the parsing process. On the other hand, we can also exhibit hierarchically-valued features that do not filter the set of trees but do increase the complexity of parsing. Let us construct a simple unification grammar with hierarchical feature structures where the feature structures simply encode the structure of the parse-trees assigned by a given binary-branching context-free grammar. We need only three features, a C feature that indicates the category of each constituent, and L and R features that point to the feature structures of the left daughter and right daughter respectively. Below is a fragment of such a grammar for English. The rules are expressed in traditional LFG notation (Kaplan and Bresnan, 1982); the lexical items are represented as rules expanding the preterminal categories rather than as separate lexical entries.

S	$\longrightarrow$	NP	VP
		$(\uparrow \mathrm{L}) = \downarrow$	$(\uparrow \mathrm{R}) = \downarrow$
		$(\uparrow \mathrm{C}) = \mathrm{S}$	
NP	$\longrightarrow$	A	NP
		$(\uparrow \mathrm{L}) = \downarrow$	$(\uparrow \mathrm{R}) = \downarrow$
		$(\uparrow \mathrm{C}) = \mathrm{NP}$	
NP	$\longrightarrow$	N	NP
		$(\uparrow \mathrm{L}) = \downarrow$	$(\uparrow \mathrm{R}) = \downarrow$
		$(\uparrow \mathrm{C}) = \mathrm{NP}$	
VP	$\longrightarrow$	V	NP
		$(\uparrow \mathrm{L}) = \downarrow$	$(\uparrow \mathrm{R}) = \downarrow$
		$(\uparrow \mathrm{C}) = \mathrm{VP}$	
N	$\longrightarrow$	light: $(\uparrow \mathrm{C}) = \mathrm{N}$	
A	$\longrightarrow$	light: $(\uparrow \mathrm{C}) = \mathrm{A}$	
N	$\longrightarrow$	brown: $(\uparrow \mathrm{C}) = \mathrm{N}$	
A	$\longrightarrow$	brown: $(\uparrow \mathrm{C}) = \mathrm{A}$	
NP	$\longrightarrow$	bears: $(\uparrow \mathrm{C}) = \mathrm{NP}$	
V	$\longrightarrow$	like: $(\uparrow \mathrm{C}) = \mathrm{V}$	
N	$\longrightarrow$	white: $(\uparrow \mathrm{C}) = \mathrm{N}$	
A	$\longrightarrow$	white: $(\uparrow \mathrm{C}) = \mathrm{A}$	
NP	$\longrightarrow$	fish: $(\uparrow \mathrm{C}) = \mathrm{NP}$	

Figure 3 shows a sample tree and its feature structure encoding as produced by this grammar. This unification grammar is clearly context-free equivalent, since it is a transparent encoding of a context-free phrase structure grammar. However, the typical unification parsing algorithms can take exponential time to analyze a sentence with respect to this grammar. There will be one feature structure produced for each parse tree so, if there are an exponential number of parse trees, there will be an exponential number of feature structures at the top level. Figure 4 illustrates this possibility (in this figure, we use a letter in square

brackets to represent a copy of the daughter feature structure that is labeled with the same letter in order to reduce the clutter).

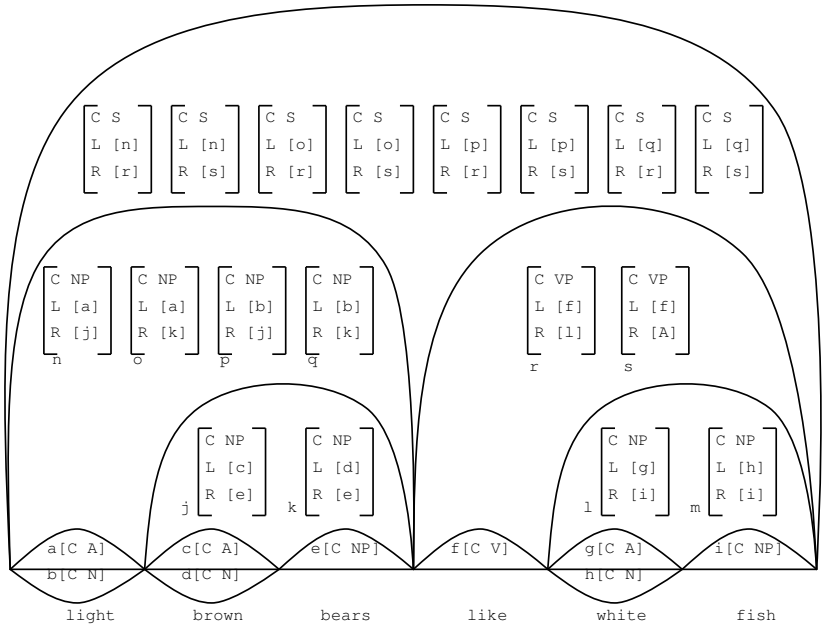

Figure 4: Unnecessary multiplication

For this particular grammar the feature constraints do not filter the parse trees, and so we could have postponed constructing the feature structures until the phase of reading out the solutions. We cannot do this in general, though, because most grammars do use the feature structures to filter the parse trees and we are not allowed to filter trees during the read out phase.

One way of dealing with grammars that have hierarchical feature structures is to truncate the feature structures below a certain depth, effectively treating the hierarchical situations as a finite-valued case. However, this might not produce correct results. If two feature structures have a feature in common, then when the feature structures are unified, the values of that feature are unified too. If those values have features in common, the process continues recursively. In the worst case, the two feature structures will be unified together from top to bottom. We call this a "zipper unification", since the process is analogous to how a zipper works. Because of zipper unifications, it is not possible to truncate feature structures while parsing, since no matter at what depth a feature structure is embedded, it may become relevant to some future unification.

If the parser knows for a given grammar that the feature constraints do not filter the trees, then it could conceivably finish the construction phase in cubic time. However, there are very few unification grammars that do absolutely no

feature-structure filtering. Usually, there is at least a little bit of filtering to determine things like subject-verb agreement, but rarely are things filtered by means zipper unification. One possibility might be to define the formalism so that zipper unification cannot be expressed or is limited to a certain depth. But a better possibility is to design algorithms that run in cubic time except for the few places where zipper unification plays an important grammatical role.

3 Lazy Copy Links

In order to understand our algorithms, we need to first understand how lazy copy links work. Lazy copy links are one way of reducing the amount of copying required by a unification chart parser (Godden 90). Feature structures from the chart must be copied when they are unified because the same feature structure may be used in two disjoint analyses. If the feature structure is not copied when unified, then the result of unification in one analysis might show up in the other analysis (this is called "cross-talk"). However, copying feature structures is very expensive since we may have to copy an amount that is proportional to the whole tree. Thus, a lot of research has been directed towards reducing the amount of copying necessary to achieve correct results. (Karttunen 86) proposed one approach to this problem, which is to unify daughter feature structures together undoably, to copy the result only if the unification succeeds, and then to undo the unification. This is valuable if most unifications fail.

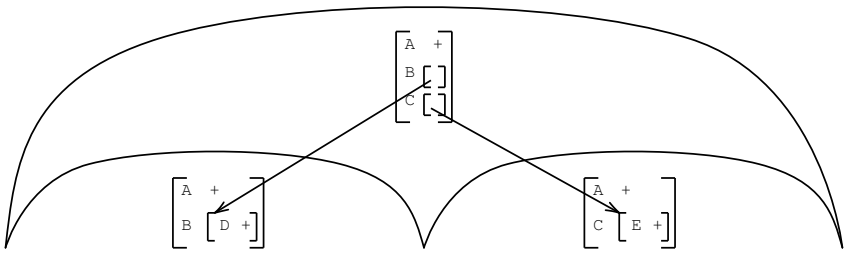

Figure 5: Lazy copying

(Godden 90) proposed another approach, namely, to copy the feature structures lazily in an incremental fashion. At first, just the top levels of each feature structure are copied. At the fringe of what has been copied, lazy copy links point to the material that has not yet been copied. Then the copied feature structures are unified together. If no lazy copy links are encountered during the unification, then the features in the two copied feature structures are disjoint and the unification can safely stop. This is because there is no possibility of the unification failing by processing the lower levels of the feature structures. The resulting feature structure will probably have a mixture of lazy links, some pointing to parts of the feature structure of one daughter and some pointing to parts of the

other's (see Figure 5). On the other hand, if lazy links are encountered during a unification, it may be that the material that has not yet been copied is incompatible with the corresponding material in the other daughter. The lazy link must be expanded to explore this possibility. This is done by copying up one level of features in the feature structure pointed to by the lazy link, introducing a new set of lazy links pointing to the next level of material. The next level of common features are unified together, with the possibility that more lazy links will need to be expanded.

The advantage of lazy copying is that it defers copying of some material when combining feature structures from two different daughter constituents. The disadvantage is that it does not deal with the possibility of alternative subtrees. In particular, it does not reduce the number of feature structures that are constructed in the cross-product computation, as can be seen by considering what lazy copying would do for Figure 4. Replacing the bracketed letters with lazy copy links would reduce the amount of copying, but it would not reduce the number of top-level feature structures.

4 Lazy DNF Unification

Our first new algorithm, Lazy DNF Unification, builds on the lazy copy idea but extends it to reduce the amount of unnecessary cross-product multiplication. The key idea of lazy DNF unification is to merge together feature structures that have the same expanded material but could have different material in the parts that have not yet been expanded (i.e. which are pointed to by lazy links). Instead of having at most one lazy link per feature at the fringe, we can have a set of *disjunctive* lazy copy links, one for each feature structure that was merged in. Whenever the unifier encounters disjunctive lazy copy links, it incrementally expands them and replaces the existing feature structure with a set of expanded feature structures. The expanded feature structures may have lazy copy links further down, allowing the process to continue if necessary.

Each merged feature structure represents a possible equivalence class of feature structures that could have the same satisfiability from the point of a consuming feature structure, depending on whether the consumer cares about the material under the lazy copy links. The major difference between these equivalence classes and those used in the chart is that these equivalence classes may be dynamically split into smaller classes when more detailed information is needed by the unifier.

Each constituent has two types of feature structures: "internal" feature structures which are the result of unifying together daughter feature structures, and "external" feature structures which are the result of merging together internal feature structures that are the same down to a certain point. The internal feature structures use lazy copy links in the standard way to avoid copying up material from below. The external feature structures use disjunctive lazy copy links to represent alternative extensions of the feature structure. The internal and external feature structures form an AND-OR tree of feature structures: the internal

feature structures point to the external daughter feature structures that they unify, and the external feature structures point to the alternative internal feature structures that they are an equivalence class for. This AND-OR structure mirrors the AND-OR structure of a parse forest, where constituents can have alternative subtree analyses and subtrees are the conjunction of constituents.

4.1 Example

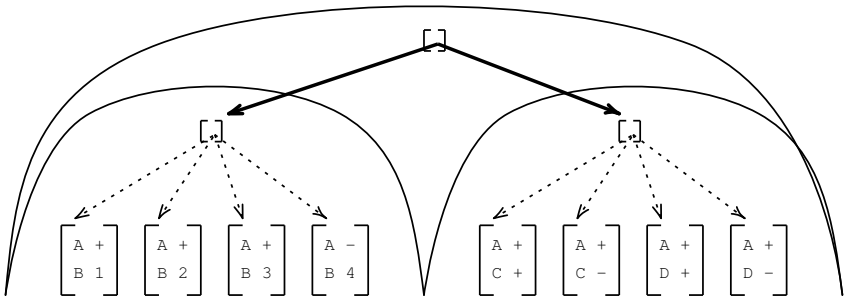

Figure 6: Unification begun, possible interaction detected

Figures 6-11 illustrate how lazy DNF unification works. We start in Figure 6 with two daughter constituents and a mother constituent. Each daughter constituent has a number of alternative internal feature structures at its bottom. These feature structures could come from lexical and/or phrasal ambiguities. The left daughter has four feature structures, each of which has an A attribute and a B attribute with different values. The right daughter has four feature structures with different combinations of A, C, and D attributes and their values. The A, C, and D attributes can have + or − as their values, the B attribute can have 1, 2, 3, or 4 for its value.

Each daughter constituent is assigned an empty external feature structure with disjunctive lazy copy links to the internal feature structures (the disjunctive lazy copy links are shown as dotted arrows). This represents the situation where the external feature structures have not been expanded at all, and so all of the internal feature structures of each daughter are in the same equivalence class.

We want to unify the external feature structures of the daughter constituents in order to produce an internal feature structure for the mother. When we try to unify the two external feature structures, the result is an empty feature structure with lazy copy links to the external feature structures. The lazy copy links are shown in bold in Figure 6 to indicate that there is a possible interaction between them and that they will have to be expanded to determine whether or not the unification is successful (there is the possibility of an interaction whenever a lazy copy link is conjoined with anything else, including another lazy copy link). This would normally trigger a simple copy, but since these lazy copy links point to

disjunctive lazy copy links, the external feature structures will first have to be expanded into a set of alternatives.

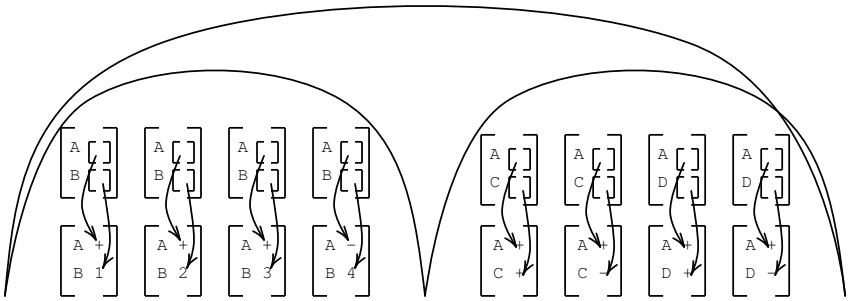

Figure 7: Lazy disjunction expanded

In Figure 7 we have expanded the daughter constituents' external feature structures. This produces four partial copies for each daughter, with lazy copy links under each attribute pointing down to the original values. Note that except for what the lazy copy links point to, many of the partial copies are identical.

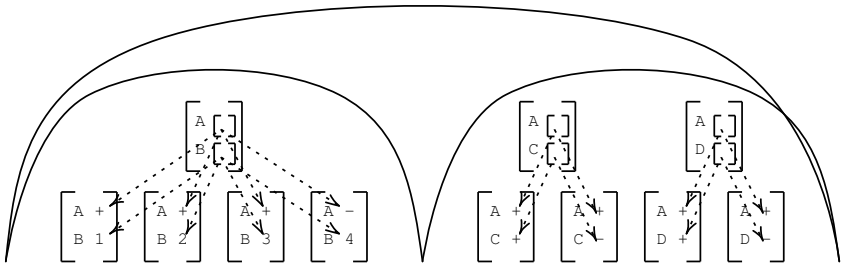

Figure 8: Equivalent feature structures merged together

In Figure 8 we have merged the identical partial copies together, producing one external feature structure on the left and two on the right (because the attributes are different). The lazy copy links have also been merged, producing disjunctive lazy copy links (again, shown as dotted arrows).

In Figure 9 we have restarted taking the cross-product of the daughter feature structures by unifying the external feature structures L1 and R1 to produce M1. This produces lazy copy links for the A, B, and C attributes. Because there are two lazy copy links under the A attribute, this attribute will have to be expanded to detect possible conflicts. But the lazy copy links point to disjunctive values, so the disjunctive values will have to be expanded first.

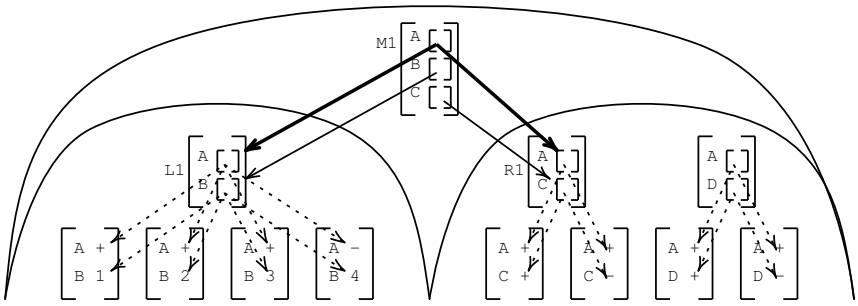

Figure 9: Unification restarted, new interaction detected

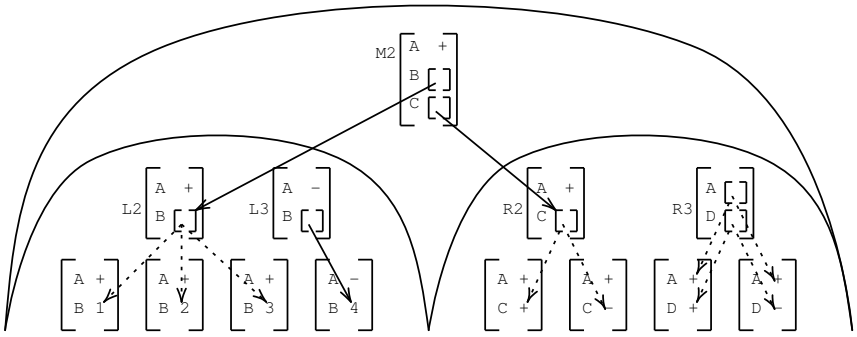

Figure 10: First unification completed

In Figure 10 we have expanded the A attributes of L1 and R1 to produce L2, L3, and R2. Expanding L1 produced four new partial copies on the left, but we merged the four partial copies into two equivalence classes: those with A=+ (represented by L2), and those with A=- (represented by L3). Expanding R1 produced two partial copies which then got merged into one equivalence class (represented by R2). At this point, we have finished the first unification in the cross-product of the daughter feature structures.

Figure 11 shows the result of taking the cross-product of all the daughter feature structures. The third and fourth unifications (the ones with L3) failed because their A attributes had inconsistent values. This left us with two valid internal feature structures for the mother. We constructed an empty feature structure (M5) with disjunctive lazy copy links to these as the external feature structure of the mother constituent. Note that if all of the lazy copy links in Figure 11 were expanded we would have 12 features structures at the top level instead of 2.

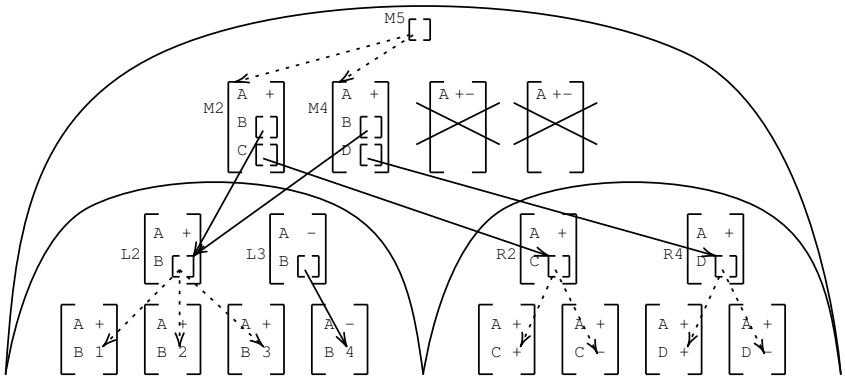

Figure 11: Cross-product completed

4.2 Dealing with a changing cache in the chart

As we noted earlier, one of the differences between the equivalence classes of a standard chart parser and the equivalence classes of a lazy DNF unifier is that the latter equivalence classes can change dynamically as more information is needed for unifications higher up[2]. This can pose a problem in a chart environment where the equivalence classes of a particular constituent may be used by several different mother constituents. What happens to the first consumer of a set of equivalence classes when the second consumer wants the set expanded? This becomes even more complicated if subsequently the first consumer also needs to change the set of equivalence classes because of another unification that is higher up in the chart.

One possibility is to allow there to be multiple sets of equivalence classes. At first there is just one set, but then as different consumers request different expansions of the equivalence classes, the sets begin to diverge. We must be careful to avoid the situation where we end up with one set of equivalence classes per consumer since in the worst case this would make the process $O(n^4)$ instead of $O(n^3)$. So this requires that we regularly check whether two sets are the same and replace one set with the other.

Another possibility is to only allow one set of equivalence classes per constituent, and to update all of the consumers so that they use the latest set whenever some equivalence classes are expanded. This can be accomplished in the following manner: whenever an equivalence class needs to be expanded, the equivalence class is modified in place to include one of the alternatives. The other alternatives are then added to the set of equivalence classes, and are cross-unified with the equivalence classes of the other daughter (if any) of all existing consumers. The cross-unification is necessary since the existing equivalence class

[2]This is independent of whether or not the chart itself is dynamic, e.g. that new subtrees can be added to a constituent even after the constituent has been consumed.

has been narrowed.

4.3 Reading out solutions

In order to be able to enumerate solutions when we are done, we always keep track of how feature structures were constructed. Thus, external feature structures will have a list of the internal feature structures that they represent, and internal feature structures have a list of the external feature structures that were used to produce them[3]. This allows us to enumerate solutions by starting at the top with the top constituent's external feature structures, and for each external feature structure non-deterministically choosing a feature structure. This feature structure must have been a product of a unification, and so we follow its links to the daughter external feature structures, and so on. The whole structure is basically an AND-OR tree similar to a parse forest in a chart, where the external feature structures correspond to constituents with different analyses and the internal feature structures correspond to particular subtrees with a conjunctive list of daughters.

4.4 Computational complexity

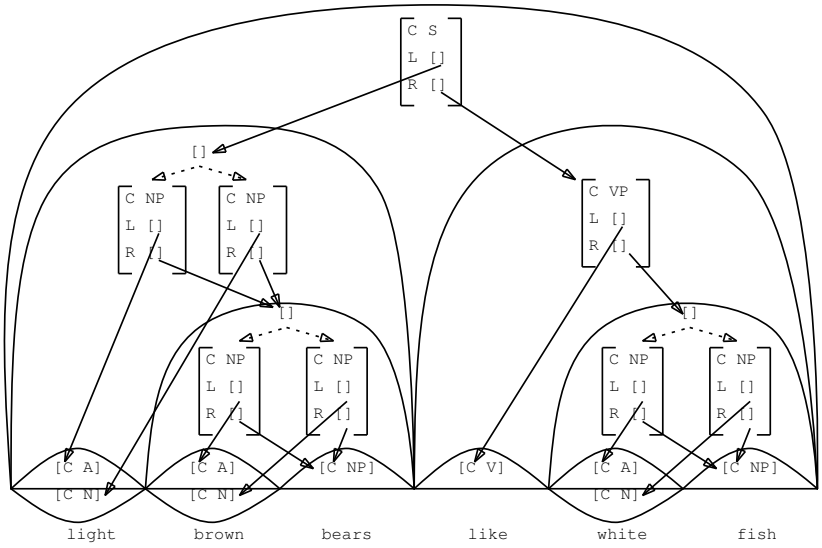

Figure 12: Figure 4 revisited

[3]The reason that we cannot use the lazy links directly for reading out solutions is that the disjunctive lazy links have to be correlated. For instance in Figure 9, the disjunctive lazy links of L1's A attribute must be correlated with those of L1's B attribute.

The computational complexity of lazy DNF unification will be proportional to the size and number of internal and external feature structures produced for each subtree in the chart forest. If a grammar is context-free equivalent, then the size of the external feature structures will be bounded by a grammar constant. This is because the number of interactions possible at each constituent will be limited, and so the amount of material that gets copied up into an external feature structure will be similarly limited.

If the size of the external feature structures is bounded by a grammar constant, then the number of external feature structures that a constituent can have is bounded, too. This is because equivalent external feature structures are always merged together, and there can only be a bounded number of equivalence classes. If the number of external feature structures that each constituent can have is bounded, then the number of internal feature structures that each subtree can have is also bounded. This is because the daughter constituents each have a bounded number of external feature structures, and the product of two bounded numbers is bounded. Since the size and number of internal feature structures per subtree is bounded, the parser parses in worst case $O(n^3)$ time. Even if a grammar is not context-free equivalent, such a parser will improve performance with for the parts that are. Note that this cubic time result does not depend on any pre-analysis or compilation of the grammar. Instead, the result is an emergent property of the way that lazy DNF unification works.

As a final example, consider how the grammar in Figure 4 is handled by lazy DNF unification in Figure 12. Note that the result is isomorphic to a parse-forest chart constructed using the context-free grammar that the constraints were derived from applied to the same input.

5 Lazy Contexted Unification

In the previous section we showed that lazy DNF unification automatically takes advantage of context-freeness when parsing with unification grammars. Although the lazy DNF unification algorithm produces a performance curve with the desired shape, it still has some potential performance problems. Although bounded, the number of external feature structures may be large. This may pose a problem whenever a new external feature structure is produced, since it must be merged with any existing external feature structure that has the same structure. In this section we present another algorithm for automatically taking advantage of context-freeness that has the property that there is always at most one external feature structure per constituent. This algorithm is called lazy contexted unification.

Lazy contexted unification is like lazy DNF unification except that it replaces standard DNF unification with contexted unification (Maxwell and Kaplan 1989). Contexted unification allows a number of alternative feature structures to be merged into one contexted feature structure, plus a set of solutions to the propositional variables used for the contexts (described below). This means that there is always exactly one representative feature structure for each

daughter constituent, and so the cross-product unification of daughter feature structures found in lazy DNF unification can be eliminated. The solutions to the contexted variables are constructed in a second pass. There is a cross-product at this stage, but the atomic operations are simpler and there is more information about which solutions are invalid, allowing more pruning of the cross-product. Thus each sentence is processed in three passes:

1. produce a context-free chart based on the phrase structure rules
2. build a contexted feature structure for each constituent in a bottom-up fashion (i.e. all the different analyses of a constituent are incorporated into its feature structure before the feature structure is consumed by higher constituents.)
3. construct the set of contexted variable solutions for each constituent in a bottom-up fashion.

Before we describe the algorithm in more detail, we first review how contexted unification works.

5.1 Contexted Unification

Contexted[4] unification is a method for merging alternative feature structures together by annotating the alternatives with propositional variables. Contexted unification is based on ideas from Assumption-based Truth Maintenance Systems (de Kleer 1986). The basic idea can be formalized with the following rules:

1. $\phi_1 \vee \phi_2$ is satisfiable if and only if $(p \rightarrow \phi_1) \wedge (\neg p \rightarrow \phi_2)$ is satisfiable, where p is a new propositional variable
2. if $\phi_1 \wedge \phi_2 \rightarrow \phi_3$ is a rule of deduction, then $(P \rightarrow \phi_1) \wedge (Q \rightarrow \phi_2) \rightarrow (P \wedge Q \rightarrow \phi_3)$ is a contexted version of that rule, where P and Q are boolean combinations of propositional variables
3. $(P \rightarrow \phi) \wedge (Q \rightarrow \phi)$ can be replaced by $(P \vee Q) \rightarrow \phi$
4. if $P \rightarrow$ FALSE, then assert $\neg P$ (P is called a nogood in ATMS terminology).

Contexted unification is performed in three stages. First, the disjunctions are converted to conjunctions using rule 1 above and instantiated as a feature structure with the propositional variables stored with the values. Then feature structures are unified and nogoods are produced. Finally, the nogoods are collected and solved in order to find out which combinations of propositional variables are valid[5].

[4]The "context" in contexted unification is not the same "context" that occurs in context-free, although they are related ideas. To avoid confusion, we will only use the latter notion of context in the phrases "context-free" or "non-context-free".

[5]See (Maxwell and Kaplan 1989) for more details.

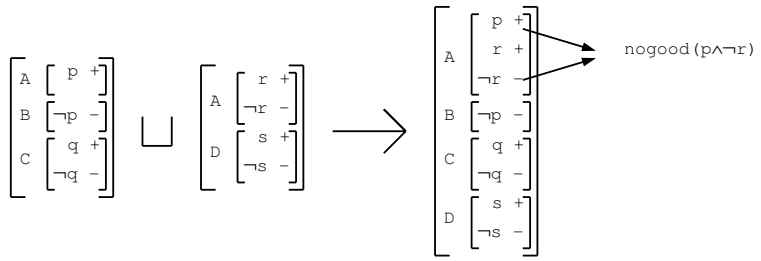

Figure 13: contexted unification

Figure 13 gives a simple example that shows how contexted unification works. We assume that the first feature structure is the result of instantiating the constraints $([A+] \vee [B-]) \wedge ([C+] \vee [C-])$ and the second is the result of $([A+] \vee [A-]) \wedge ([D+] \vee [D-])$. The propositional variables p, q, r, and s have been introduced to represent the four disjunctions. When we unify the two feature structures on the left, only the A attributes overlap. A cannot be $+$ and $-$ at the same time, so we obtain the nogood $p \wedge \neg r$.

In order to find the solutions, we compute all possible combinations of propositional variables that are consistent with the nogoods that we have found. In this case, there are 12 solutions:

$p \wedge q \wedge r \wedge s$	$\neg p \wedge q \wedge r \wedge s$	$\neg p \wedge \neg q \wedge r \wedge s$
$p \wedge q \wedge r \wedge \neg s$	$\neg p \wedge q \wedge r \wedge \neg s$	$\neg p \wedge \neg q \wedge r \wedge \neg s$
$p \wedge \neg q \wedge r \wedge s$	$\neg p \wedge q \wedge \neg r \wedge s$	$\neg p \wedge \neg q \wedge \neg r \wedge s$
$p \wedge \neg q \wedge r \wedge \neg s$	$\neg p \wedge q \wedge \neg r \wedge \neg s$	$\neg p \wedge \neg q \wedge \neg r \wedge \neg s$

Note that all of the solutions that involve $p \wedge \neg r$ have been eliminated.

The main advantage of contexted unification is that it postpones taking the cross-product of alternatives until we have found all of the nogoods. This is helpful for three reasons: 1) taking the cross-product of propositional variables is less expensive than taking the cross-product of feature structures, 2) we do not take cross-products until we have complete information (It is often the case that late in the unification process we discover that the unification is completely invalid, in which case there is no need for cross-products), 3) the nogood database may decompose into a set of independent problems that can be solved independently, avoiding a global cross-product of solutions.

5.2 Lazy Copying Using Contexted Unification

Let us go back to the example we used to illustrate how lazy DNF unification worked, and use it to show how lazy contexted unification works. In Figure 14 we have two daughter constituents that each have an empty external feature structure with disjunctive lazy copy links pointing to the internal feature structures. This is like Figure 6, only now the disjunctive lazy copy links have been

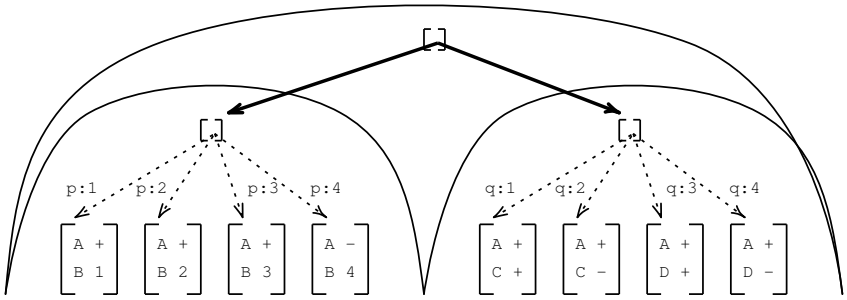

Figure 14: Possible interaction detected

annotated with contexts p:1, p:2, p:3, p:4 and q:1, q:2, q:3, q:4[6]. We start to produce an internal feature structure for the mother constituent by unifying copies of the daughters' external feature structures. The bold lazy copy links indicate that there is a possible interaction between the external feature structures and that they will need to be expanded.

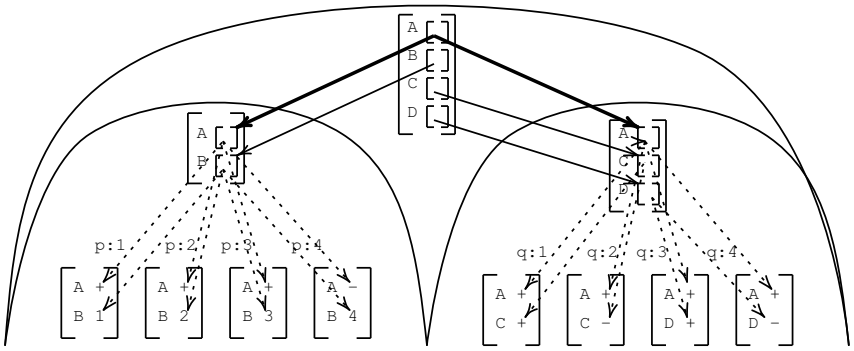

Figure 15: New interaction detected

In Figure 15[7] the daughters' external feature structures have been expanded in place. We copied up the first level of each of the alternative feature structures into the corresponding daughter feature structure and pushed the contexted lazy copy links down one level. Notice that in the right daughter we have a single feature structure that has both a C feature and a D feature. This is correct, because the C feature only has values in the q:1 and q:2 contexts, and

[6]We use this notation to allow the context variables to have more than two values. It is logically equivalent to define p:1 = $r \wedge s$, p:2 = $r \wedge \neg s$, p:3 = $\neg r \wedge t$, and p:4 = $\neg r \wedge \neg t$, where s only takes a value when r is TRUE, and t only takes a value when $\neg r$ is TRUE.

[7]The contexts annotate the disjunctive lazy copy links, not the feature structures. To avoid clutter each context in the figure labels two lazy links.

the D feature only has values in the q:3 and q:4 contexts. Having expanded the attributes in place in the mother's internal solution, we continued the unification and notice a possible interaction under the A attribute, shown in bold.

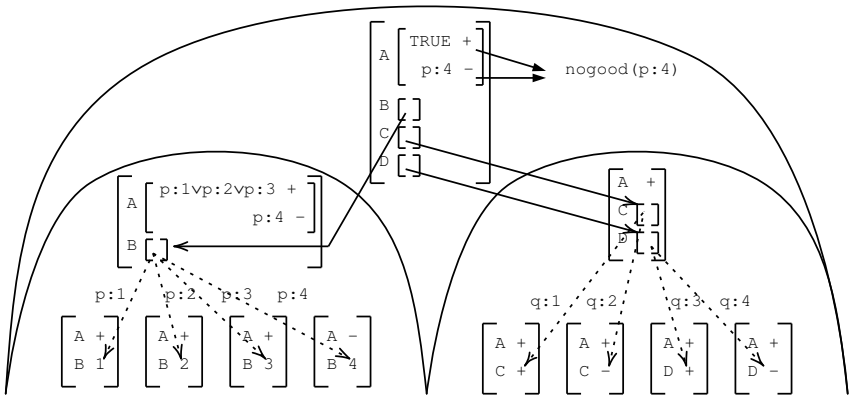

Figure 16: Nogood asserted

In Figure 16 we expand the A attribute of both external daughter feature structures and determine that A=+ when p:1 or p:2 or p:3 is TRUE, and A=- when p:4 is TRUE for the left daughter, and that A=+ in every context for the right daughter. Copying this information up into the mother's internal feature structure, we find that A=+ in the TRUE context[8] and that A=- when p:4 is TRUE. Since A cannot be plus and minus at the same time, we assert p:4 as a nogood.

5.3 Solving the Nogood Database

After we have done all of the unifications, we still need to find ways of assigning values to propositional variables that avoid the nogoods in the nogood databases. The case above is fairly simple: we throw out p:4, and we are left with (p:1 ∨ p:2 ∨ p:3) ∧ (q:1 ∨ q:2 ∨ q:3 ∨ q:4). Another case that is handled fairly simply is when there are no nogoods at all. In this case, we can freely choose any value for each propositional variable and we are guaranteed to have a solution.

In more complicated cases, we may have nogoods involving conjunctions of propositional variables scattered all over the chart. This can happen even in context-free equivalent unification grammars because of things like subject-verb agreement and unsatisfied subcategorization frames. In these more complicated cases, we cannot use simple techniques like those described above to determine what are the valid solutions. Instead, we need to enumerate possible assignments of values to propostional variables and test each set of assignments against

[8] A=+ is in the TRUE context because of the rule that $(P \rightarrow \phi) \wedge (Q \rightarrow \phi)$ can be replaced by $(P \vee Q) \rightarrow \phi$. In this case, TRUE disjoined with anything is TRUE.

the nogoods to determine which assignments are good. One way to do this is to gather all of the nogoods together and treat the nogoods and variables as a giant boolean satisfaction problem. Unfortunately, this can produce $O(n^3)$ nogoods over $O(n^3)$ propositional variables, even if the grammar is context-free equivalent. Since the boolean satisfaction problem is NP complete, it could take $O(2^{n^3})$ time to solve this problem (unless P=NP).

What we really want to do is to solve the nogoods locally, within the internal feature structures that they occur. If the grammar is context-free equivalent, then the number of nogoods that each internal feature structure can have will be bounded. This means that the boolean satisfaction problem will be bounded. We may end up with $O(n^3)$ boolean satisfaction problems, but if each is bounded, the overall process will still be cubic.

Just solving the nogoods locally is not enough, however. This is because the contexts of the facts that are copied up from the external feature structures of the daughters can be complicated boolean expressions that involve propositional variables used by descendents of the daughter constituents. In the worst case, some of these expressions can have a size proportional to the chart up to that point.

The key insight that lets us solve the local nogoods in a bounded amount of time is that although the contexts of the copied facts can be arbitrarily complicated, there are only a bounded number of contexts for each internal feature structure if the grammar is context-free equivalent (because only a bounded number of facts are copied). We take advantage of this insight by introducing a new propositional variable for each context expression when a context are copied from an external feature structure into an internal feature structure. We call these variables *opaque* variables since their internal structure is not available at the higher level[9].

Making the contexts opaque does not make them independent. In general, there may be complicated dependencies between the opaque variables that will have to be taken account of when solving the nogood database associated with an internal feature structure. To do this, we first collect all of the opaque variables that were imported from each daughter constituent. We then give each daughter the opaque variables that were imported from it, and request that the opaque variables be solved for the assignments of TRUE and FALSE to the variables that satisfy the daughter's nogoods. The collection of opaque variables and their solutions are cached on the daughter constituent in case other consumers need the same information. If both daughters have solutions, we filter these solutions by the local nogoods. If there are still solutions left, we take the cross product of the solutions and filter the cross product solutions by the local nogoods. If the constituent has multiple internal solutions (one for each subtree used to build the constituent), then each internal feature structure is assigned a context and the context is added to each of its solutions.

[9]In practice we make contexts opaque by wrapping them in an opaque wrapper when they are imported by a consumer. We do the wrapping after a context is imported instead of before so that we can associate information with the wrapper that is unique to the consumer without worrying about cross-talk between consumers.

Finally, we need to recast these internal solutions as solutions to the opaque variables exported by the external feature structure of the mother. We do this by assigning truth values to the internal solutions one at a time, and then evaluating each opaque variable in an exported set to see what truth value it evaluates to (there may be several exported sets of variables because different consumers may import different sets of facts). The set of values represents a solution to the exported set of variables. In general, each solution to an exported set will be represented by several internal solutions, since there will be $O(n)$ internal solutions but the number of external variables (and hence solutions) must be bounded if the grammar is context-free equivalent. This is analogous to there being multiple subtrees for each constituent in a context-free parse forest, where the "constituent" in our case is some set of feature-value combinations.

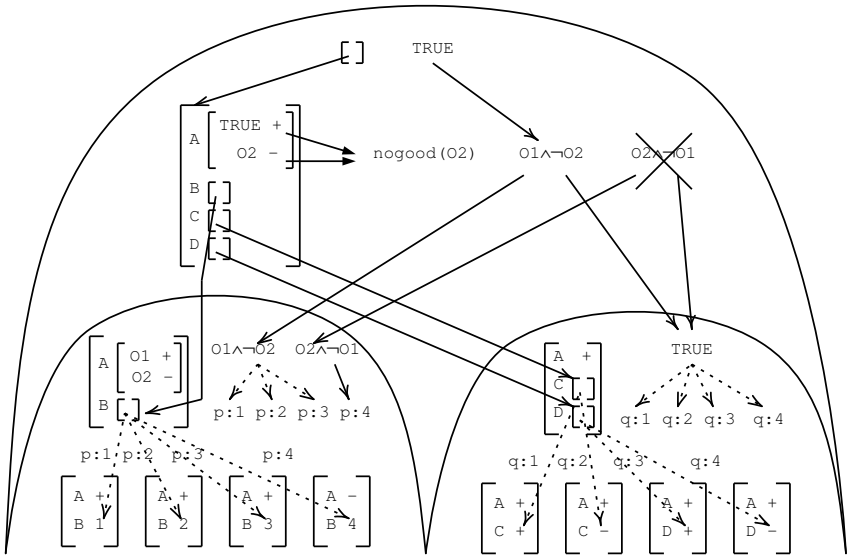

Figure 17: Solutions constructed

We show how this works with Figure 17. The left daughter has two opaque variables, $O1$ and $O2$, and the right daughter has none (because no contexts are exported). This produces two external solutions for the left daughter: $O1 \wedge \neg O2$ and $O2 \wedge \neg O1$[10]. The $O1 \wedge \neg O2$ solution represents three internal solutions: p:1, p:2, and p:3, corresponding to the cases where A=+. The $O2 \wedge \neg O1$ solution represents one internal solution: p:4, corresponding to the case where A=-. The right daughter has one external solution in the TRUE context which represents four internal solutions. The solutions to the mother constituent are constructed by taking the cross-product of the external daughter solutions and filtering by

[10] $O1 \wedge O2$ and $\neg O1 \wedge \neg O2$ are ruled out by the implicit disjointness of p:1, p:2, p:3, and p:4.

the local nogood $O2$. This produces one internal solution, $O1 \wedge \neg O2$, which is assigned to the external solution TRUE (since we are assuming that no contexts were exported from the mother's external feature structure). The internal solutions keep track of the daughter solutions that they came from so that the solutions can be enumerated easily.

5.4 Reading out solutions

Solutions are read out of a lazy contexted unification chart by first reading out a solution to the propositional variables, and then using the propositional variables to read out a constituent structure and feature structure. We begin reading out a solution to the propositional variables by picking an external solution for the top constituent (the root constituent that spans the sentence). Each external solution may have multiple internal solutions. We freely choose one. Internal solutions usually point to two external solutions, one for each daughter constituent. For each external solution, we freely choose an internal solution. We continue this process recursively until we reach internal solutions that do not point to external solutions (these correspond to the leaves of the parse tree).

Once we have a solution to the propositional variables, we can use this solution to extract a constituent structure and a feature structure. We start from the top constituent. If there are multiple subtrees, we pick the one whose internal feature structure's context evaluates to TRUE in this solution. Also, we pick the parts of the external and internal feature structures whose contexts evaluate to TRUE. We recursively apply the same process to each of the subtree's daughters. When we are done, we will have a complete constituent structure and a consistent feature structure.

5.5 Computational complexity

The lazy contexted unification algorithm is worst-case polynomial except for the solution construction phase. In the worst case, the number of solutions to a set of opaque variables is exponential in the number of opaque variables. However, if the number of opaque variables is bounded by the grammar, this exponential becomes a grammar constant. So the lazy contexted unification algorithm is $O(Gn^3)$ unless the number of opaque variables per subtree can grow as a function of n, the length of the sentence. This can happen in three different ways:

1. The number of attributes at a particular level grows as a function of n.
2. The number of values that an attribute can have grows as a function of n.
3. The depth of attributes grows as a function of n.

In general, the number of attributes and values that can appear is determined in advance by the grammar. The depth of attributes can only grow as a function

of n if there are zipper unifications. One conclusion from this is that if there are no zipper unifications and long-distance dependencies are handled by slash categories and the number of values in the slash categories is bounded either in theory or in practice, then the parser will still parse in cubic time with that grammar.

6 Experimental Results

The algorithm for lazy contexted unification described above is used in the parser and generator of the Xerox Linguistic Environment (XLE[11]). XLE, in turn, has been used as part of the ParGram project[12] to develop parallel LFG grammars of English, French, and German (among other languages). At one point, these grammar were all used to parse parallel versions of a software manual for the HomeCentre, a Xerox product. The sentences (or sentence fragments) range from 1 word to 50 words long, with the average being about 9 words. If we plot the number of subtrees processed per sentence against the number of seconds that it takes to parse a sentence[13], then we can see whether lazy contexted unification is automatically taking advantage of context-freeness. The plots for English(Figure 18), French(Figure 19), and German(Figure 20) can be found at the end of the paper.

If we do a regression test on the statistics for the English HomeCentre corpus, we find that 75 per cent of the variance in time is explained by the number of subtrees. Most of the outliers are on sentences that have an interaction between a long distance dependency and coordination. This sort of interaction increases the amount of copying required, which increases the effective grammar constant for these sentences. For the French corpus, 79 per cent of the variance in time is explained by the number of subtrees. Finally, 51 per cent of the variance in time is explained by the number of subtrees for the German corpus. This is probably because German allows much more scrambling than English or French, and so more information must be copied up.

The English grammar can also be used to parse the Wall Street Journal corpus from the Penn treebank. The statistics for section 02 of this corpus are shown in Figure 21. About 2 per cent of the sentences fail to parse within 60 seconds and are excluded from the statistics. For some of these sentences, the grammar constant becomes so large that the sentences are effectively unparsable. However, it may be that by writing the grammar more carefully XLE will be able to parse the sentences in a reasonable amount of time. If we exclude these sentences, we find that 78 per cent of the variance in time is explained by the number of subtrees.

[11] http://www.parc.xerox.com/istl/groups/nltt/xle
[12] http://www.parc.xerox.com/istl/groups/nltt/pargram
[13] using a 360MHz processor on a Sun Ultra 60

7 Dealing with Incompleteness

A number of grammatical theories have a mechanism for specifying that certain analyses are incomplete. For instance, Functional Unification Grammar has an *ANY* value that unifies with anything and which must not be present in a valid (i.e. complete) solution. As another example, LFG requires that each argument of a predicate have a predicate itself, plus it allows the grammar writer to specify that a feature must be assigned any value or some particular value in order for the feature structure to be complete. These mechanisms are useful for requiring that a predicate be complete (i.e. saying why *John threw* is not a valid sentence without a thrown object), and for other things (like requiring that all sentences be tensed).

One problem with detecting incompleteness is that we may not know whether a feature structure is incomplete until we have finished processing the entire sentence. For instance, the sentence *John threw* is perfectly valid if it is embedded in a relative clause: *The ball that [John threw] is lost.* This means that incomplete solutions cannot be pruned early. It also means that we must examine an entire feature structure in order to determine whether or not it is incomplete. On the surface, this would seem to imply that we would have to look at every solution one at a time to see which ones are incomplete. But this results in an exponential system, since there can be an exponential number of solutions, even when the grammar is context-free equivalent.

However, there is a fairly simple way of detecting incompletenesses locally. First of all, it is obvious that if a feature structure has something like the *ANY* value in it, and the feature that contains it has never been copied up, then the feature structure that contains the *ANY* valued feature is incomplete. This observation allows us to locally declare the feature structure incomplete, and add a corresponding nogood to the database.

If an *ANY* value has been copied up by all of the consumers of the constituent that it is part of, then this *ANY* value will be either satisfied or not satisfied at some other level, and there is no need to do any further work at this one's level.

A more complicated case is when an *ANY* value has been copied up by some but not all of the consumers of its constituent. In this case, the *ANY*s that have been copied up will be dealt with at a different level, but not the ones that haven't. To solve this, all that we need to do is assert the context of the *ANY* to be nogood at the level of the consumers that have not copied up the *ANY*.

In order to be able to tell whether something has been copied, we require that whenever a lazy copy link is expanded, a forward copy link is added to the copied feature structure. That way we can determine whether a feature structure has been copied at all. Also, we can compare the forward links with the list of consumers to find out whether all of the consumers have copied the *ANY* value, and if not, which ones should have a nogood asserted on them. This adds to the constant overhead of doing unification, but is justified since it helps us to avoid an exponential cost.

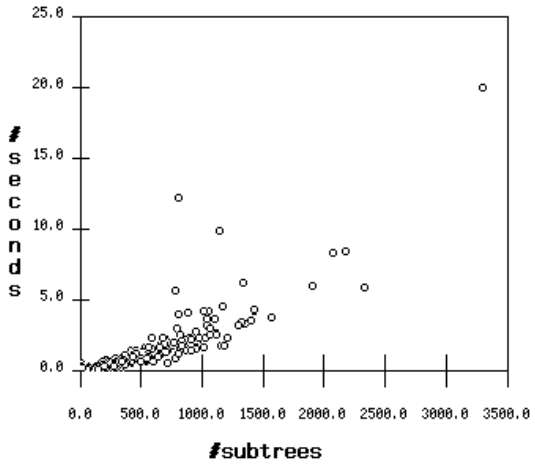

Figure 18: English HomeCentre corpus

8 Conclusion

In this paper we have shown that the standard algorithms for parsing unification-based grammars can run in exponential time even when the grammars are context-free equivalent. Using this as our starting point, we have introduced two different algorithms for automatically taking advantage of simple context-freeness in unification-based grammars. The essence of both of these algorithms is to implement lazy copying of a data structure that packs alternative analyses together. Finally, we have shown that these techniques are useful for industrial-strength grammars even when the grammars may not be completely context-free equivalent. This allows us to write grammars using an expressive formalism without having to sacrifice performance except in the case where non-context-free phenomena are being analysed.

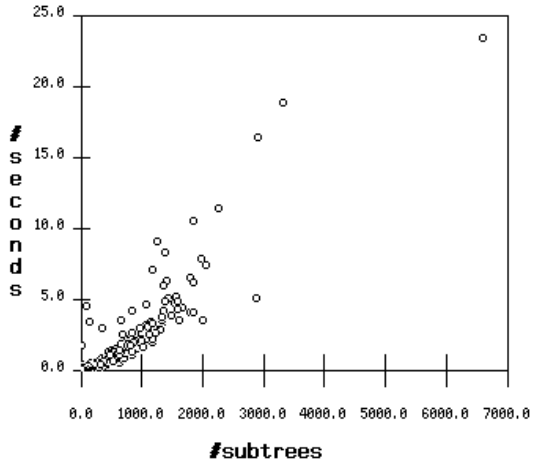

Figure 19: French HomeCentre corpus

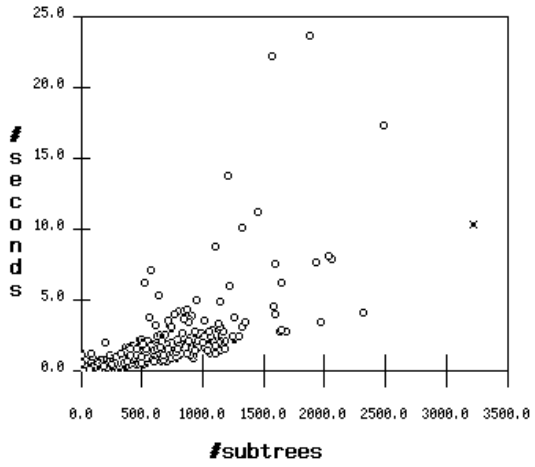

Figure 20: German HomeCentre corpus

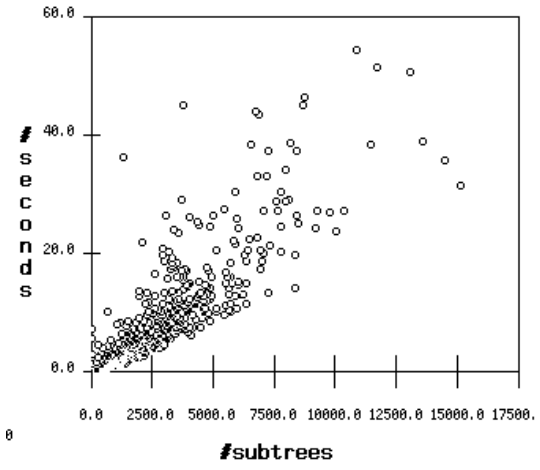

Figure 21: section 02 of the WSJ

References

[1] Barton, G. Edward; Berwick, Robert C.; and Ristad, Eric Sven. (1987). *Computational Complexity and Natural Language*. MIT Press, Cambridge, Mass.

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

[3] Blackburn, Patrick, and Spaan, Edith. (1993). "Decidability and Undecidability in stand-alone Feature Logics." In *Proceedings of the Sixth Conference of the EACL*, Utrecht, The Netherlands, pages 30–36.

[4] de Kleer, Johan. (1986). "An Assumption-based TMS". *Artificial Intelligence*, 28:127–162.

[5] Dörre, Jochen, and Eisele, Andreas. (1990). "Feature logic with disjunctive unification." In *Proceedings, 13th International Conference on Computational Linguistics (COLING 90)*, Helsinki, Finland, pages 100–105.

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

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

[8] Godden, K. (1990). "Lazy unification." In *Proceedings of the 28th Annual Meeting of the ACL*, pages 180–187.

[9] Johnson, Mark. (1988). *Attribute-Value Logic and the Theory of Grammar*, volume 16 of *CSLI Lecture Notes*. CSLI, Stanford.

[10] Kaplan, Ronald M., and Bresnan, Joan. (1982). "Lexical-Functional Grammar: A formal system for grammatical representation." In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–281. MIT Press, Cambridge, Mass.

[11] Karttunen, Lauri. (1984). "Features and Values". In *Proceedings of COLING 1984*, Stanford, Calif.

[12] Karttunen, Lauri. (1986). "D-PATR: A development environment for unification-based grammars." In *Proceedings, 11th International Conference on Computational Linguistics (COLING 86)*, Bonn, Germany, pages 74–80.

[13] Kay, Martin. (1979). "Functional Grammar." In C. Chiarello et al., editors, *Proceedings of the 5th Annual Meeting of the Berkeley Linguistic Society* Berkeley, California, pages 142–158.

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

[15] Lari, K. and Young, S. J. (1990). "The Estimation of Stochastic Context-Free Grammars Using the Inside-Outside Algorithm" *Computer Speech and Language*, 4:35-56.

[16] Maxwell, John T. III, and Kaplan, Ronald M. (1989). "An overview of disjunctive constraint satisfaction." In *Proceedings of the International Workshop on Parsing Technologies*, pages 18–27. (Also published as "A method for disjunctive constraint satisfaction" in M. Tomita, editor, *Current Issues in Parsing Technology*, Kluwer Academic Publishers, 1991).

[17] Maxwell, John T. III, and Kaplan, Ronald M. (1993). "The interface between phrasal and functional constraints." *Computational Linguistics*, 19(3).

[18] 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.

[19] Pollard, Carl, and Sag, Ivan. (1987). *Information-Based Syntax and Semantics*, volume 13 of *CSLI Lecture Notes*. CSLI, Stanford.

[20] Rogers, James. (1994). "Capturing CFLs with Tree Adjoining Grammars". In *Proceedings of the 32nd Annual Meeting of the ACL*, Las Cruces, New Mexico, pages 155–162.

[21] Schabes, Yves, and Waters, Richard C., (1993). "Lexicalized Context-Free Grammars". In *Proceedings of the 31st Annual Meeting of the ACL*, Columbus, Ohio, pages 121–129.

[22] Sheil, Beau. (1976). "Observations on context-free parsing." In *Statistical Methods in Linguistics*, 1976:71-109.

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

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