Using Restriction to Extend Parsing Algorithms for Complex-Feature-Based Formalisms

Stuart M. Shieber

Artificial Intelligence Center
SRI International
and
Center for the Study of Language and Information
Stanford University

Abstract

Grammar formalisms based on the encoding of grammatical information in complex-valued feature systems enjoy some currency both in linguistics and natural-language-processing research. Such formalisms can be thought of by analogy to context-free grammars as generalizing the notion of nonterminal symbol from a finite domain of atomic elements to a possibly infinite domain of directed graph structures of a certain sort. Unfortunately, in moving to an infinite nonterminal domain, standard methods of parsing may no longer be applicable to the formalism. Typically, the problem manifests itself as gross inefficiency or even nontermination of the algorithms. In this paper, we discuss a solution to the problem of extending parsing algorithms to formalisms with possibly infinite nonterminal domains, a solution based on a general technique we call restriction. As a particular example of such an extension, we present a complete, correct, terminating extension of Earley's algorithm that uses restriction to perform top-down filtering. Our implementation of this algorithm demonstrates the drastic elimination of chart edges that can be achieved by this technique. Finally, we describe further uses for the technique—including parsing other grammar formalisms, including definite-clause grammars; extending other parsing algorithms, including LR methods and syntactic preference modeling algorithms; and efficient indexing.

This research has been made possible in part by a gift from the Systems Development Foundation, and was also supported by the Defense Advanced Research Projects Agency under Contract N00039-84-K-0078 with the Naval Electronics Systems Command. The views and conclusions contained in this document should not be interpreted as representative of the official policies, either expressed or implied, of the Defense Research Projects Agency or the United States government.

The author is indebted to Fernando Pereira and Ray Perrault for their comments on earlier drafts of this paper.

1 Introduction

Grammar formalisms based on the encoding of grammatical information in complex-valued feature systems enjoy some currency both in linguistics and natural-language-processing research. Such formalisms can be thought of by analogy to context-free grammars as generalizing the notion of nonterminal symbol from a finite domain of atomic elements to a possibly infinite domain of directed graph structures of a certain sort. Many of the surface-based grammatical formalisms explicitly defined or presupposed in linguistics can be characterized in this way—e.g., lexical-functional grammar (LFG) [5], generalized phrase structure grammar (GPSG) [4], even categorial systems such as Montague grammar [8] and Ades/Steedman grammar [1]—as can several of the grammar formalisms being used in natural-language processing research—e.g., definite clause grammar (DCG) [9], and PATR-II [13].

Unfortunately, in moving to an infinite nonterminal domain, standard methods of parsing may no longer be applicable to the formalism. For instance, the application of techniques for preprocessing of grammars in order to gain efficiency may fail to terminate, as in left-corner and LR algorithms. Algorithms performing top-down prediction (e.g. top-down backtrack parsing, Earley's algorithm) may not terminate at parse time. Implementing backtracking regimens—useful for instance for generating parses in some particular order, say, in order of syntactic preference—is in general difficult when LR-style and top-down backtrack techniques are eliminated.

In this paper, we discuss a solution to the problem of extending parsing algorithms to formalisms with possibly infinite nonterminal domains, a solution based on an operation we call restriction. In Section 2, we summarize traditional proposals for solutions and problems inherent in them and propose an alternative approach to a solution using restriction. In Section 3, we present some technical background including a brief description of the PATR-II formalism—which is used as the formalism interpreted by the parsing algorithms—and a formal definition of restriction for

PATR-II's nonterminal domain. In Section 4, we develop a correct, complete and terminating extension of Earley's algorithm for the PATR-II formalism using the restriction notion. Readers uninterested in the technical details of the extensions may want to skip these latter two sections, referring instead to Section 4.1 for an informal overview of the algorithms. Finally, in Section 5, we discuss applications of the particular algorithm and the restriction technique in general.

2 Traditional Solutions and an Alternative Approach

Problems with efficiently parsing formalisms based on potentially infinite nonterminal domains have manifested themselves in many different ways. Traditional solutions have involved limiting in some way the class of grammars that can be parsed.

2.1 Limiting the formalism

The limitations can be applied to the formalism by, for instance, adding a context-free "backbone." If we require that a context-free subgrammar be implicit in every grammar, the subgrammar can be used for parsing and the rest of the grammar used as a filter during or after parsing. This solution has been recommended for functional unification grammars (FUG) by Martin Kay [6]; its legacy can be seen in the context-free skeleton of LFG, and the Hewlett-Packard GPSG system [3], and in the *cat* feature requirement in PATR-II that is described below.

However, several problems inhere in this solution of mandating a context-free backbone. First, the move from context-free to complex-feature-based formalisms was motivated by the desire to structure the notion of nonterminal. Many analyses take advantage of this by eliminating mention of major category information from particular rules[1] or by structuring the major category itself (say into binary N and V features plus a bar-level feature as in $\bar{X}$-based theories). Forcing the primacy and atomicity of major category defeats part of the purpose of structured category systems.

Second, and perhaps more critically, because only certain of the information in a rule is used to guide the parse, say major category information, only such information can be used to filter spurious hypotheses by top-down filtering. Note that this problem occurs even if filtering by the rule information is used to eliminate at the earliest possible time constituents and partial constituents proposed during parsing (as is the case in the PATR-II implementation and the Earley algorithm given below; cf. the Xerox LFG system). Thus, if information about subcategorization is left out of the category information in the context-free skeleton, it cannot be used to eliminate prediction edges. For example, if we find a verb that subcategorizes for a noun phrase, but the grammar rules allow postverbal NPs, PPs, $\bar{S}$s, VPs, and so forth, the parser will have no way to eliminate the building of edges corresponding to these categories. Only when such edges attempt to join with the V will the inconsistency be found. Similarly, if information about filler-gap dependencies is kept extrinsic to the category information, as in a slash category in GPSG or an LFG annotation concerning a matching constituent for a $\Uparrow$ specification, there will be no way to keep from hypothesizing gaps at any given vertex. This "gap-proliferation" problem has plagued many attempts at building parsers for grammar formalisms in this style.

In fact, by making these stringent requirements on what information is used to guide parsing, we have to a certain extent thrown the baby out with the bathwater. These formalisms were intended to free us from the tyranny of atomic nonterminal symbols, but for good performance, we are forced toward analyses putting more and more information in an atomic category feature. An example of this phenomenon can be seen in the author's paper on LR syntactic preference parsing [14]. Because the LALR table building algorithm does not in general terminate for complex-feature-based grammar formalisms, the grammar used in that paper was a simple context-free grammar with subcategorization and gap information placed in the atomic nonterminal symbol.

2.2 Limiting grammars and parsers

On the other hand, the grammar formalism can be left unchanged, but particular grammars developed that happen not to succumb to the problems inherent in the general parsing problem for the formalism. The solution mentioned above of placing more information in the category symbol falls into this class. Unpublished work by Kent Wittenburg and by Robin Cooper has attempted to solve the gap proliferation problem using special grammars.

In building a general tool for grammar testing and debugging, however, we would like to commit as little as possible to a particular grammar or style of grammar.[2] Furthermore, the grammar designer should not be held down in building an analysis by limitations of the algorithms. Thus a solution requiring careful crafting of grammars is inadequate.

Finally, specialized parsing algorithms can be designed that make use of information about the particular grammar being parsed to eliminate spurious edges or hypotheses. Rather than using a general parsing algorithm on a

[1] See, for instance, the coordination and copular "be" analyses from GPSG [4], the nested VP analysis used in some PATR-II grammars [15], or almost all categorial analyses, in which general rules of combination play the role of specific phrase-structure rules.

[2] See [12] for further discussion of this matter.

limited formalism, Ford, Bresnan, and Kaplan [2] chose a specialized algorithm working on grammars in the full LFG formalism to model syntactic preferences. Current work at Hewlett-Packard on parsing recent variants of GPSG seems to take this line as well.

Again, we feel that the separation of burden is inappropriate in such an attack, especially in a grammar-development context. Coupling the grammar design and parser design problems in this way leads to the linguistic and technological problems becoming inherently mixed, magnifying the difficulty of writing an adequate grammar/parser system.

2.3 An Alternative: Using Restriction

Instead, we would like a parsing algorithm that placed no restraints on the grammars it could handle as long as they could be expressed within the intended formalism. Still, the algorithm should take advantage of that part of the arbitrarily large amount of information in the complex-feature structures that is *significant* for guiding parsing with the particular grammar. One of the aforementioned solutions is to require the grammar writer to put all such significant information in a special atomic symbol—i.e., mandate a context-free backbone. Another is to use *all* of the feature structure information—but this method, as we shall see, inevitably leads to nonterminating algorithms.

A compromise is to parameterize the parsing algorithm by a small amount of grammar-dependent information that tells the algorithm *which* of the information in the feature structures is significant for guiding the parse. That is, the parameter determines how to split up the infinite nonterminal domain into a finite set of equivalence classes that can be used for parsing. By doing so, we have an optimal compromise: Whatever part of the feature structure is significant we distinguish in the equivalence classes by setting the parameter appropriately, so the information is used in parsing. But because there are only a finite number of equivalence classes, parsing algorithms guided in this way will terminate.

The technique we use to form equivalence classes is *restriction*, which involves taking a quotient of the domain with respect to a *restrictor*. The restrictor thus serves as the sole repository of grammar-dependent information in the algorithm. By tuning the restrictor, the set of equivalence classes engendered can be changed, making the algorithm more or less efficient at guiding the parse. But independent of the restrictor, the algorithm will be correct, since it is still doing parsing over a finite domain of "nonterminals," namely, the elements of the restricted domain.

This idea can be applied to solve many of the problems engendered by infinite nonterminal domains, allowing preprocessing of grammars as required by LR and LC algorithms, allowing top-down filtering or prediction as in Earley and top-down backtrack parsing, guaranteeing termination, etc.

3 Technical Preliminaries

Before discussing the use of restriction in parsing algorithms, we present some technical details, including a brief introduction to the PATR-II grammar formalism, which will serve as the grammatical formalism that the presented algorithms will interpret. PATR-II is a simple grammar formalism that can serve as the least common denominator of many of the complex-feature-based and unification-based formalisms prevalent in linguistics and computational linguistics. As such it provides a good testbed for describing algorithms for complex-feature-based formalisms.

3.1 The PATR-II nonterminal domain

The PATR-II nonterminal domain is a lattice of directed, acyclic, graph structures (dags).[3] Dags can be thought of as similar to the reentrant f-structures of LFG or functional structures of FUG, and we will use the bracketed notation associated with these formalisms for them. For example, the following is a dag (D_0) in this notation, with reentrancy indicated with coindexing boxes:

$$\left[\begin{array}{ll} a: & \left[b:\ c\right] \\ d: & \left[\begin{array}{ll} e: & \boxed{1}\left[f:\ \left[g:\ h\right]\right] \\ i: & \left[j:\ \boxed{1}\right] \\ k: & l \end{array}\right] \end{array}\right]$$

Dags come in two varieties, *complex* (like the one above) and *atomic* (like the dags h and c in the example). Complex dags can be viewed as partial functions from labels to dag values, and the notation $D(l)$ will therefore denote the value associated with the label l in the dag D. In the same spirit, we can refer to the domain of a dag ($dom(D)$). A dag with an empty domain is often called an *empty* dag or *variable*. A *path* in a dag is a sequence of label names (notated, e.g., $(d\ e\ f)$), which can be used to pick out a particular subpart of the dag by repeated application (in this case, the dag $[g : h]$). We will extend the notation $D(p)$ in the obvious way to include the subdag of D picked out by a path p. We will also occasionally use the square brackets as the dag constructor function, so that $[f : D]$ where D is an expression denoting a dag will denote the dag whose f feature has value D.

3.2 Subsumption and Unification

There is a natural lattice structure for dags based on *subsumption*—an ordering on dags that roughly corresponds to the compatibility and relative specificity of information

[3] The reader is referred to earlier works [15,10] for more detailed discussions of dag structures.

contained in the dags. Intuitively viewed, a dag D subsumes a dag D' (notated $D \sqsubseteq D'$) if D contains a subset of the information in (i.e., is more general than) D'.

Thus variables subsume all other dags, atomic or complex, because as the trivial case, they contain no information at all. A complex dag D subsumes a complex dag D' if and only if $D(l) \sqsubseteq D'(l)$ for all $l \in dom(D)$ and $D'(P) = D'(q)$ for all paths p and q such that $D(p) = D(q)$. An atomic dag neither subsumes nor is subsumed by any different atomic dag.

For instance, the following subsumption relations hold:

$$[\ ] \sqsubseteq [d : e] \sqsubseteq \begin{bmatrix} a: & [b:c] \\ e: & f \end{bmatrix} \sqsubseteq \begin{bmatrix} a: & \boxed{1}[b:c] \\ d: & \boxed{1} \\ e: & f \end{bmatrix} .$$

Finally, given two dags D' and D'', the *unification* of the dags is the most general dag D such that $D' \sqsubseteq D$ and $D'' \sqsubseteq D$. We notate this $D = D' \sqcup D''$.

The following examples illustrate the notion of unification:

$$\begin{bmatrix} a: & [b:c] \end{bmatrix} \sqcup \begin{bmatrix} d: & e \end{bmatrix} = \begin{bmatrix} a: & [b:c] \\ d: & e \end{bmatrix}$$

$$\begin{bmatrix} a: & [b:c] \end{bmatrix} \sqcup \begin{bmatrix} a: & \boxed{1}[] \\ d: & \boxed{1} \end{bmatrix} = \begin{bmatrix} a: & \boxed{1}[b:c] \\ d: & \boxed{1} \end{bmatrix} .$$

The unification of two dags is not always well-defined. In the cases where no unification exists, the unification is said to *fail*. For example the following pair of dags fail to unify with each other:

$$\begin{bmatrix} a: & \boxed{1}[] \\ d: & \boxed{1} \end{bmatrix} \sqcup \begin{bmatrix} a: & [b:c] \\ d: & [b:d] \end{bmatrix} = fail .$$

3.3 Restriction in the PATR-II nonterminal domain

Now, consider the notion of *restriction* of a dag, using the term almost in its technical sense of restricting the domain of a function. By viewing dags as partial functions from labels to dag values, we can envision a process of restricting the domain of this function to a given set of labels. Extending this process recursively to every level of the dag, we have the concept of restriction used below. Given a finite specification Φ (called a restrictor) of what the allowable domain at each node of a dag is, we can define a functional, $\restriction$, that yields the dag restricted by the given restrictor.

Formally, we define restriction as follows. Given a relation Φ between paths and labels, and a dag D, we define $D\restriction\Phi$ to be the most specific dag $D' \sqsubseteq D$ such that for every path p either $D'(p)$ is undefined, or $D'(p)$ is atomic, or for every $l \in dom(D'(p))$, $p\Phi l$. That is, every path in the restricted dag is either undefined, atomic, or *specifically allowed* by the restrictor.

The restriction process can be viewed as putting dags into equivalence classes, each equivalence class being the largest set of dags that all are restricted to the same dag (which we will call its *canonical member*). It follows from the definition that in general $D\restriction\Phi \sqsubseteq D$. Finally, if we disallow infinite relations as restrictors (i.e., restrictors must not allow values for an infinite number of distinct paths) as we will do for the remainder of the discussion, we are guaranteed to have only a finite number of equivalence classes.

Actually, in the sequel we will use a particularly simple subclass of restrictors that are generable from sets of paths. Given a set of paths s, we can define Φ such that $p\Phi l$ if and only if p is a prefix of some $p' \in s$. Such restrictors can be understood as "throwing away" all values not lying on one of the given paths. This subclass of restrictors is sufficient for most applications. However, the algorithms that we will present apply to the general class as well.

Using our previous example, consider a restrictor Φ_0 generated from the set of paths $\{\langle a\ b\rangle, \langle d\ e\ f\rangle, \langle d\ i\ j\ f\rangle\}$. That is, $p\Phi_0 l$ for all p in the listed paths and all their prefixes. Then given the previous dag D_0, $D_0\restriction\Phi_0$ is

$$\begin{bmatrix} a: & [b: & c] \\ d: & \begin{bmatrix} e: & \boxed{1}[f: & []] \\ i: & [j: & \boxed{1}] \end{bmatrix} \end{bmatrix} .$$

Restriction has thrown away all the information except the direct values of $\langle a\ b\rangle$, $\langle d\ e\ f\rangle$, and $\langle d\ i\ j\ f\rangle$. (Note however that because the values for paths such as $\langle d\ e\ f\ g\rangle$ were thrown away, $(D_0\restriction\Phi_0)(\langle d\ e\ f\rangle)$ is a variable.)

3.4 PATR-II grammar rules

PATR-II rules describe how to combine a sequence of constituents, $X_1, \ldots, X_n$ to form a constituent X_0, stating mutual constraints on the dags associated with the $n+1$ constituents as unifications of various parts of the dags. For instance, we might have the following rule:

$$\begin{aligned} X_0 \rightarrow X_1 X_2 : \\ &\langle X_0\ cat\rangle = S \\ &\langle X_1\ cat\rangle = NP \\ &\langle X_2\ cat\rangle = VP \\ &\langle X_1\ agreement\rangle = \langle X_2\ agreement\rangle . \end{aligned}$$

By notational convention, we can eliminate unifications for the special feature *cat* (the atomic major category feature) recording this information implicitly by using it in the "name" of the constituent, e.g.,

$S \rightarrow NP\ VP$:
$\langle NP\ agreement \rangle = \langle VP\ agreement \rangle$.

If we *require* that this notational convention always be used (in so doing, guaranteeing that each constituent have an atomic major category associated with it), we have thereby mandated a context-free backbone to the grammar, and can then use standard context-free parsing algorithms to parse sentences relative to grammars in this formalism. Limiting to a context-free-based PATR-II is the solution that previous implementations have incorporated.

Before proceeding to describe parsing such a context-free-based PATR-II, we make one more purely notational change. Rather than associating with each grammar rule a set of unifications, we instead associate a dag that incorporates all of those unifications implicitly, i.e., a rule is associated with a dag D_r such that for all unifications of the form $p = q$ in the rule. $D_r(p) = D_r(q)$. Similarly, unifications of the form $p = a$ where a is atomic would require that $D_r(p) = a$. For the rule mentioned above, such a dag would be

$$\begin{bmatrix} X_0: & [\ cat: & S\ ] \\ X_1: & \begin{bmatrix} cat: & NP \\ agreement: & \boxed{1}[] \end{bmatrix} \\ X_2: & \begin{bmatrix} cat: & VP \\ agreement: & \boxed{1} \end{bmatrix} \end{bmatrix}$$

Thus a rule can be thought of as an ordered pair $\langle P, D \rangle$ where P is a production of the form $X_0 \rightarrow X_1 \cdots X_n$ and D is a dag with top-level features $X_0, \ldots, X_n$ and with atomic values for the *cat* feature of each of the top-level subdags. The two notational conventions—using sets of unifications instead of dags, and putting the *cat* feature information implicitly in the names of the constituents—allow us to write rules in the more compact and familiar format above, rather than this final cumbersome way presupposed by the algorithm.

4 Using Restriction to Extend Earley's Algorithm for PATR-II

We now develop a concrete example of the use of restriction in parsing by extending Earley's algorithm to parse grammars in the PATR-II formalism just presented.

4.1 An overview of the algorithms

Earley's algorithm is a bottom-up parsing algorithm that uses top-down prediction to hypothesize the starting points of possible constituents. Typically, the prediction step determines which *categories* of constituent can start at a given point in a sentence. But when most of the information is not in an atomic category symbol, such prediction is relatively useless and many types of constituents are predicted that could never be involved in a completed parse. This standard Earley's algorithm is presented in Section 4.2.

By extending the algorithm so that the prediction step determines which *dags* can start at a given point, we can use the information in the features to be more precise in the predictions and eliminate many hypotheses. However, because there are a potentially infinite number of such feature structures, the prediction step may never terminate. This extended Earley's algorithm is presented in Section 4.3.

We compromise by having the prediction step determine which *restricted dags* can start at a given point. If the restrictor is chosen appropriately, this can be as constraining as predicting on the basis of the whole feature structure, yet prediction is guaranteed to terminate because the domain of restricted feature structures is finite. This final extension of Earley's algorithm is presented in Section 4.4.

4.2 Parsing a context-free-based PATR-II

We start with the Earley algorithm for context-free-based PATR-II on which the other algorithms are based. The algorithm is described in a chart-parsing incarnation, vertices numbered from 0 to n for an n-word sentence $w_1 \cdots w_n$. An item of the form $[h, i, A \rightarrow \alpha.\beta, D]$ designates an edge in the chart from vertex h to i with dotted rule $A \rightarrow \alpha.\beta$ and dag D.

The chart is initialized with an edge $[0, 0, X_0 \rightarrow .\alpha, D]$ for each rule $\langle X_0 \rightarrow \alpha, D \rangle$ where $D(\langle X_0\ cat \rangle) = S$.

For each vertex i do the following steps until no more items can be added:

Predictor step: For each item ending at i of the form $[h, i, X_0 \rightarrow \alpha.X_j\beta, D]$ and each rule of the form $\langle X_0 \rightarrow \gamma, E \rangle$ such that $E(\langle X_0\ cat \rangle) = D(\langle X_j\ cat \rangle)$, add an edge of the form $[i, i, X_0 \rightarrow .\gamma, E]$ if this edge is not subsumed by another edge.

Informally, this involves *predicting top-down all rules whose left-hand-side category matches the category of some constituent being looked for.*

Completer step: For each item of the form $[h, i, X_0 \rightarrow \alpha., D]$ and each item of the form $[g, h, X_0 \rightarrow \beta.X_j\gamma, E]$ add the item $[g, i, X_0 \rightarrow \beta X_j.\gamma, E \sqcup [X_j : D(X_0)]]$ if the unification succeeds[4] and this edge is not subsumed by another edge.[5]

[4] Note that this unification will fail if $D(\langle X_0\ cat \rangle) \neq E(\langle X_j\ cat \rangle)$ and no edge will be added, i.e., if the subphrase is not of the appropriate category for insertion into the phrase being built.

[5] One edge subsumes another edge if and only if the first three elements of the edges are identical and the fourth element of the first edge subsumes that of the second edge.

Informally, this involves *forming a new partial phrase whenever the category of a constituent needed by one partial phrase matches the category of a completed phrase and the dag associated with the completed phrase can be unified in appropriately.*

Scanner step: If $i \neq 0$ and $w_i = a$, then for all items $[h, i-1, X_0 \rightarrow \alpha.a\beta, D]$ add the item $[h, i, X_0 \rightarrow \alpha a.\beta, D]$.

Informally, this involves *allowing lexical items to be inserted into partial phrases.*

Notice that the Predictor Step in particular assumes the availability of the *cat* feature for top-down prediction. Consequently, this algorithm applies only to PATR-II with a context-free base.

4.3 Removing the Context-Free Base: An Inadequate Extension

A first attempt at extending the algorithm to make use of more than just a single atomic-valued *cat* feature (or less if no such feature is mandated) is to change the Predictor Step so that instead of checking the predicted rule for a left-hand side that matches its *cat* feature with the predicting subphrase, we require that the whole left-hand-side subdag unifies with the subphrase being predicted from. Formally, we have

Predictor step: For each item ending at i of the form $[h, i, X_0 \rightarrow \alpha.X_j\beta, D]$ and each rule of the form $\langle X_0 \rightarrow \gamma, E\rangle$, add an edge of the form $[i, i, X_0 \rightarrow .\gamma, E \sqcup [X_0 : D(X_j)]]$ if the unification succeeds and this edge is not subsumed by another edge.

This step *predicts top-down all rules whose left-hand side matches the* dag *of some constituent being looked for.*

Completer step: As before.

Scanner step: As before.

However, this extension does not preserve termination. Consider a "counting" grammar that records in the dag the number of terminals in the string.[6]

$S \rightarrow T$:
$\langle S f\rangle = a.$
$T_1 \rightarrow T_2\ A$:
$\langle T_1 f\rangle = \langle T_2 f f\rangle.$
$S \rightarrow A.$
$A \rightarrow a.$

[6]Similar problems occur in natural language grammars when keeping *lists* of, say, subcategorized constituents or gaps to be found.

Initially, the $S \rightarrow T$ rule will yield the edge

$$[0,0,X_0 \rightarrow .X_1, \left[\begin{array}{ll} X_0: & \left[\begin{array}{ll} cat: & S \end{array}\right] \\ X_1: & \left[\begin{array}{ll} cat: & T \\ f: & a \end{array}\right] \end{array}\right]]$$

which in turn causes the Prediction step to give

$$[0,0,X_0 \rightarrow .X_1, \left[\begin{array}{ll} X_0: & \left[\begin{array}{ll} cat: & T \\ f: & \boxed{1}_a \end{array}\right] \\ X_1: & \left[\begin{array}{ll} cat: & T \\ f: & \left[\begin{array}{ll} f: & \boxed{1} \end{array}\right] \end{array}\right] \\ X_2: & \left[\begin{array}{ll} cat: & A \end{array}\right] \end{array}\right]]$$

yielding in turn

$$[0,0,X_0 \rightarrow .X_1, \left[\begin{array}{ll} X_0: & \left[\begin{array}{ll} cat: & T \\ f: & \boxed{1}_a \end{array}\right] \\ X_1: & \left[\begin{array}{ll} cat: & T \\ f: & \left[\begin{array}{ll} f: & \left[\begin{array}{ll} f: & \boxed{1} \end{array}\right] \end{array}\right] \end{array}\right] \\ X_2: & \left[\begin{array}{ll} cat: & A \end{array}\right] \end{array}\right]]$$

and so forth ad infinitum.

4.4 Removing the Context-free Base: An Adequate Extension

What is needed is a way of "forgetting" some of the structure we are using for top-down prediction. But this is just what restriction gives us, since a restricted dag always subsumes the original, i.e., it has strictly less information. Taking advantage of this property, we can change the Prediction Step to restrict the top-down information before unifying it into the rule's dag.

Predictor step: For each item ending at i of the form $[h, i, X_0 \rightarrow \alpha.X_j\beta, D]$ and each rule of the form $\langle X_0 \rightarrow \gamma, E\rangle$, add an edge of the form $[i, i, X_0 \rightarrow .\gamma, E \sqcup (D(X_j)\upharpoonright\Phi)]$ if the unification succeeds and this edge is not subsumed by another edge.

This step *predicts top-down all rules whose left-hand side matches the* restricted dag *of some constituent being looked for.*

Completer step: As before.

Scanner step: As before.

This algorithm on the previous grammar, using a restrictor that allows through only the *cat* feature of a dag, operates as before, but predicts the first time around the more general edge:

$$[0, 0, X_0 \rightarrow .X_1, \begin{bmatrix} X_0: & \begin{bmatrix} cat: & T \\ f: & \boxed{1}[] \end{bmatrix} \\ X_1: & \begin{bmatrix} cat: & T \\ f: & \begin{bmatrix} f: & \boxed{1} \end{bmatrix} \end{bmatrix} \\ X_2: & \begin{bmatrix} cat: & A \end{bmatrix} \end{bmatrix}]$$

Another round of prediction yields *this same edge* so the process terminates immediately. duck Because the predicted edge is more general than (i.e., subsumes) all the infinite number of edges it replaced that were predicted under the nonterminating extension, it preserves *completeness.* On the other hand, because the predicted edge is not more general than the rule itself, it permits no constituents that violate the constraints of the rule; therefore, it preserves *correctness.* Finally, because restriction has a finite range, the prediction step can only occur a finite number of times before building an edge identical to one already built; therefore, it preserves *termination.*

5 Applications

5.1 Some Examples of the Use of the Algorithm

The algorithm just described has been implemented and incorporated into the PATR-II Experimental System at SRI International, a grammar development and testing environment for PATR-II grammars written in Zetalisp for the Symbolics 3600.

The following table gives some data suggestive of the effect of the restrictor on parsing efficiency. It shows the total number of active and passive edges added to the chart for five sentences of up to eleven words using four different restrictors. The first allowed only category information to be used in prediction, thus generating the same behavior as the unextended Earley's algorithm. The second added subcategorization information in addition to the category. The third added filler-gap dependency information as well so that the gap proliferation problem was removed. The final restrictor added verb form information. The last column shows the percentage of edges that were eliminated by using this final restrictor.

	Prediction				%
Sentence	*cat*	+ *subcat*	+ *gap*	+ *form*	elim.
1	33	33	20	16	52
2	85	50	29	21	75
3	219	124	72	45	79
4	319	319	98	71	78
5	812	516	157	100	88

Several facts should be kept in mind about the data above. First, for sentences with no Wh-movement or relative clauses, no gaps were ever predicted. In other words, the top-down filtering is in some sense maximal with respect to gap hypothesis. Second, the subcategorization information used in top-down filtering removed all hypotheses of constituents except for those directly subcategorized for. Finally, the grammar used contained constructs that would cause nontermination in the unrestricted extension of Earley's algorithm.

5.2 Other Applications of Restriction

This technique of restriction of complex-feature structures into a finite set of equivalence classes can be used for a wide variety of purposes.

First, parsing algorithms such as the above can be modified for use by grammar formalisms other than PATR-II. In particular, definite-clause grammars are amenable to this technique, and it can be used to extend the Earley deduction of Pereira and Warren [11]. Pereira has used a similar technique to improve the efficiency of the BUP (bottom-up left-corner) parser [7] for DCG, LFG and GPSG parsers can make use of the top-down filtering device as well. FUG parsers might be built that do not require a context-free backbone.

Second, restriction can be used to enhance other parsing algorithms. For example, the ancillary function to compute LR closure—which, like the Earley algorithm, either does not use feature information, or fails to terminate—can be modified in the same way as the Earley predictor step to terminate while still using significant feature information. LR parsing techniques can thereby be used for efficient parsing of complex-feature-based formalisms. More speculatively, schemes for scheduling LR parsers to yield parses in preference order might be modified for complex-feature-based formalisms, and even tuned by means of the restrictor.

Finally, restriction can be used in areas of parsing other than top-down prediction and filtering. For instance, in many parsing schemes, edges are indexed by a category symbol for efficient retrieval. In the case of Earley's algorithm, active edges can be indexed by the category of the constituent following the dot in the dotted rule. However, this again forces the primacy and atomicity of major category information. Once again, restriction can be used to solve the problem. Indexing by the restriction of the dag associated

with the need permits efficient retrieval that can be tuned to the particular grammar, yet does not affect the completeness or correctness of the algorithm. The indexing can be done by discrimination nets, or specialized hashing functions akin to the partial-match retrieval techniques designed for use in Prolog implementations [16].

6 Conclusion

We have presented a general technique of restriction with many applications in the area of manipulating complex-feature-based grammar formalisms. As a particular example, we presented a complete, correct, terminating extension of Earley's algorithm that uses restriction to perform top-down filtering. Our implementation demonstrates the drastic elimination of chart edges that can be achieved by this technique. Finally, we described further uses for the technique—including parsing other grammar formalisms, including definite-clause grammars; extending other parsing algorithms, including LR methods and syntactic preference modeling algorithms; and efficient indexing.

We feel that the restriction technique has great potential to make increasingly powerful grammar formalisms computationally feasible.

References

[1] Ades, A. E. and M. J. Steedman. On the order of words. *Linguistics and Philosophy*, 4(4):517–558, 1982.

[2] Ford, M., J. Bresnan, and R. Kaplan. A competence-based theory of syntactic closure. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*, MIT Press, Cambridge, Massachusetts, 1982.

[3] Gawron, J. M., J. King, J. Lamping, E. Loebner, E. A. Paulson, G. K. Pullum, I. A. Sag, and T. Wasow. Processing English with a generalized phrase structure grammar. In *Proceedings of the 20th Annual Meeting of the Association for Computational Linguistics*, pages 74–81, University of Toronto, Toronto, Ontario, Canada, 16-18 June 1982.

[4] Gazdar, G., E. Klein, G. K. Pullum, and I. A. Sag. *Generalized Phrase Structure Grammar*. Blackwell Publishing, Oxford, England, and Harvard University Press, Cambridge, Massachusetts, 1985.

[5] Kaplan, R. and J. Bresnan. Lexical-functional grammar: a formal system for grammatical representation. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*, MIT Press, Cambridge, Massachusetts, 1983.

[6] Kay, M. An algorithm for compiling parsing tables from a grammar. 1980. Xerox Palo Alto Research Center, Palo Alto, California.

[7] Matsumoto, Y., H. Tanaka, H. Hirakawa, H. Miyoshi, and H. Yasukawa. BUP: a bottom-up parser embedded in Prolog. *New Generation Computing*, 1:145–158, 1983.

[8] Montague, R. The proper treatment of quantification in ordinary English. In R. H. Thomason, editor, *Formal Philosophy*, pages 188–221, Yale University Press, New Haven, Connecticut, 1974.

[9] Pereira, F. C. N. Logic for natural language analysis. Technical Note 275, Artificial Intelligence Center, SRI International, Menlo Park, California, 1983.

[10] Pereira, F. C. N. and S. M. Shieber. The semantics of grammar formalisms seen as computer languages. In *Proceedings of the Tenth International Conference on Computational Linguistics*, Stanford University, Stanford, California, 2-7 July 1984.

[11] Pereira, F. C. N. and D. H. D. Warren. Parsing as deduction. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, pages 137–144, Massachusetts Institute of Technology, Cambridge, Massachusetts, 15-17 June 1983.

[12] Shieber, S. M. Criteria for designing computer facilities for linguistic analysis. To appear in *Linguistics*.

[13] Shieber, S. M. The design of a computer language for linguistic information. In *Proceedings of the Tenth International Conference on Computational Linguistics*, Stanford University, Stanford, California, 2-7 July 1984.

[14] Shieber, S. M. Sentence disambiguation by a shift-reduce parsing technique. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, pages 113–118, Massachusetts Institute of Technology, Cambridge, Massachusetts, 15-17 June 1983.

[15] Shieber, S. M., H. Uszkoreit, F. C. N. Pereira, J. J. Robinson, and M. Tyson. The formalism and implementation of PATR-II. In *Research on Interactive Acquisition and Use of Knowledge*, SRI International, Menlo Park, California, 1983.

[16] Wise, M. J. and D. M. W. Powers. Indexing Prolog clauses via superimposed code words and field encoded words. In *Proceedings of the 1984 International Symposium on Logic Programming*, pages 203–210, IEEE Computer Society Press, Atlantic City, New Jersey, 6-9 February 1984.