Principles and Implementation of Deductive Parsing

Stuart M. Shieber
Division of Applied Sciences
Harvard University, Cambridge, MA 02138

Yves Schabes
Mitsubishi Electric Research Laboratories
Cambridge, MA 02139

Fernando C. N. Pereira
AT&T Bell Laboratories
Murray Hill, NJ 07974

February 10, 2001

Abstract

We present a system for generating parsers based directly on the metaphor of parsing as deduction. Parsing algorithms can be represented directly as deduction systems, and a single deduction engine can interpret such deduction systems so as to implement the corresponding parser. The method generalizes easily to parsers for augmented phrase structure formalisms, such as definite-clause grammars and other logic grammar formalisms, and has been used for rapid prototyping of parsing algorithms for a variety of formalisms including variants of tree-adjoining grammars, categorial grammars, and lexicalized context-free grammars.

This paper is available from the Center for Research in Computing Technology, Division of Applied Sciences, Harvard University as Technical Report TR-11-94, and through the Computation and Language e-print archive as cmp-lg/9404008.

1 Introduction

Parsing can be viewed as a deductive process that seeks to prove claims about the grammatical status of a string from assumptions describing the grammatical properties of the string's elements and the linear order between them. Lambek's syntactic calculi (Lambek, 1958) comprise an early formalization of this idea, which more recently was explored in relation to grammar formalisms based on definite clauses (Colmerauer, 1978; Pereira and Warren, 1980; Pereira and Warren, 1983) and on feature logics (Shieber, 1992; Rounds and Manaster-Ramer, 1987; Carpenter, 1992).

The view of parsing as deduction adds two main new sources of insights and techniques to the study of grammar formalisms and parsing:

1. Existing logics can be used as a basis for new grammar formalisms with desirable representational or computational properties.

2. The modular separation of parsing into a logic of grammaticality claims and a proof search procedure allows the investigation of a wide range of parsing algorithms for existing grammar formalisms by selecting specific classes of grammaticality claims and specific search procedures.

While most of the work on deductive parsing has been concerned with (1), we will in this paper investigate (2), more specifically how to synthesize parsing algorithms by combining specific logics of grammaticality claims with a fixed search procedure. In this way, deduction can provide a metaphor for parsing that encompasses a wide range of parsing algorithms for an assortment of grammatical formalisms. We flesh out this metaphor by presenting a series of parsing algorithms literally as inference rules, and by providing a uniform deduction engine, parameterized by such rules, that can be used to parse according to any of the associated algorithms. The inference rules for each logic will be represented as unit clauses and the fixed deduction procedure, which we provide a Prolog implementation of, will be a version of the usual bottom-up consequence closure operator for definite clauses. As we will show, this method directly yields dynamic-programming versions of standard top-down, bottom-up, and mixed-direction (Earley) parsing procedures. In this, our method has similarities with the use of pure bottom-up deduction to encode dynamic-programming versions of definite-clause proof procedures in deductive databases (Bancilhon and Ramakrishnan, 1988; Naughton and Ramakrishnan, 1991).

The program that we develop is especially useful for rapid prototyping of and experimentation with new parsing algorithms, and was in fact developed for that purpose. We have used it, for instance, in the development of algorithms for parsing with tree-adjoining grammars, categorial grammars, and lexicalized context-free grammars.

Many of the ideas that we present are not new. Some have been presented before; others form part of the folk wisdom of the logic programming community.

However, the present work is to our knowledge the first to make the ideas available explicitly in a single notation and with a clean implementation. In addition, certain observations regarding efficient implementation may be novel to this work.

The paper is organized as follows: After reviewing some basic logical and grammatical notions and applying them to a simple example (Section 2), we describe how the structure of a variety of parsing algorithms for context-free grammars can be expressed as inference rules in specialized logics (Section 3). Then, we extend the method for stating and implementing parsing algorithms for formalisms other than context-free grammars (Section 4). Finally, we discuss how deduction should proceed for such logics, developing an agenda-based deduction procedure implemented in Prolog that manifests the presented ideas (Section 5).

2 Basic Notions

As introduced in Section 1, we see parsing as a deductive process in which rules of inference are used to derive statements about the grammatical status of strings from other such statements. Statements are represented by formulas in a suitable formal language. The general form of a rule of inference is

$$\frac{A_1 \quad \cdots \quad A_k}{B} \quad \langle \text{side conditions on } A_1, \ldots, A_k, B \rangle \qquad .$$

The *antecedents* $A_1, \ldots, A_k$ and the *consequent* B of the inference rule are formula schemata, that is, they may contain syntactic metavariables to be instantiated by appropriate terms when the rule is used. A grammatical deduction system is defined by a set of rules of inference and a set of *axioms* given by appropriate formula schemata.

Given a grammatical deduction system, a *derivation* of a formula B from assumptions $A_1, \ldots, A_m$ is, as usual, a sequence of formulas $S_1, \ldots, S_n$ such that $B = S_n$, and each S_i is either an axiom (one of the A_j) or there is a rule of inference R and formulas $S_{i_1}, \ldots, S_{i_k}$ with $i_1, \ldots, i_k < i$ such that for appropriate substitutions of terms for the metavariables in R, $S_{i_1}, \ldots, S_{i_k}$ match the antecedents of the rule, S_i matches the consequent, and the rule's side conditions are satisfied. We write $A_1, \ldots, A_m \vdash B$ and say that B is a *consequence* of $A_1, \ldots, A_m$ if such a derivation exists. If B is a consequence of the empty set of assumptions, it is said to be *derivable*, in symbols $\vdash B$.

In our applications of this model, rules and axiom schemata may refer in their side conditions to the rules of a particular grammar, and formulas may refer to string positions in the fixed string to be parsed $w = w_1 \cdots w_n$. With respect to the given string, *goal formulas* state that the string is grammatical according to the given grammar. Then parsing the string corresponds to finding a derivation witnessing a goal formula.

We will use standard notation for metavariables ranging over the objects under discussion: n for the length of the object language string to be parsed; $A, B, C \ldots$ for arbitrary formulas or symbols such as grammar nonterminals; $a, b, c, \ldots$ for arbitrary terminal symbols; $i, j, k, \ldots$ for indices into various strings, especially the string w; $\alpha, \beta, \gamma, \ldots$ for strings or terminal and nonterminal symbols. We will often use such notations leaving the type of the object implicit in the notation chosen for it. Substrings will be notated elliptically as, e.g., $w_i \cdots w_j$ for the i-th through j-th elements of w, inclusive. As is usual, we take $w_i \cdots w_j$ to be the empty string if $i > j$.

2.1 A First Example: CYK Parsing

As a simple example, the basic mechanism of the Cocke-Younger-Kasami (CYK) context-free parsing algorithm (Kasami, 1965; Younger, 1967) for a context-free grammar in Chomsky normal form can be easily represented as a grammatical deduction system.

We assume that we are given a string $w = w_1 \cdots w_n$ to be parsed and a context-free grammar $G = \langle N, \Sigma, P, S \rangle$, where N is the set of nonterminals including the start symbol S, Σ is the set of terminal symbols, ($V = N \cup \Sigma$ is the vocabulary of the grammar,) and P is the set of productions, each of the form $A \rightarrow \alpha$ for $A \in N$ and $\alpha \in V^*$. We will use the symbol $\Rightarrow$ for immediate derivation and $\stackrel{*}{\Rightarrow}$ for its reflexive, transitive closure, the derivation relation. In the case of a Chomsky-normal-form grammar, all productions are of the form $A \rightarrow B\ C$ or $A \rightarrow a$.

The *items* of the logic (as we will call parsing logic formulas from now on) are of the form $[A, i, j]$, and state that the nonterminal A derives the substring between indices i and j in the string, that is, $A \stackrel{*}{\Rightarrow} w_{i+1} \cdots w_j$. Sound axioms, then, are grounded in the lexical items that occur in the string. For each word w_{i+1} in the string and each rule $A \rightarrow w_{i+1}$, it is clear that the item $[A, i, i+1]$ makes a true claim, so that such items can be taken as axiomatic. Then whenever we know that $B \stackrel{*}{\Rightarrow} w_{i+1} \cdots w_j$ and $C \stackrel{*}{\Rightarrow} w_{j+1} \cdots w_k$ — as asserted by items of the form $[B, i, j]$ and $[C, j, k]$ — where $A \rightarrow B\ C$ is a production in the grammar, it is sound to conclude that $C \stackrel{*}{\Rightarrow} w_{i+1} \cdots w_k$, and therefore, the item $[C, i, k]$ should be inferable. This argument can be codified in a rule of inference:

$$\frac{[B, i, j] \qquad [C, j, k]}{[A, i, k]} \quad A \rightarrow B\ C$$

Using this rule of inference with the axioms, we can conclude that the string is admitted by the grammar if an item of the form $[S, 0, n]$ is deducible, since such an item asserts that $S \stackrel{*}{\Rightarrow} w_1 \cdots w_n = w$. We think of this item as the *goal item* to be proved.

In summary, the CYK deduction system (and all the deductive parsing systems we will define) can be specified with four components: a class of items; a

Item form:	$[A, i, j]$
Axioms:	$[A, i, i+1] \quad A \rightarrow w_{i+1}$
Goals:	$[S, 0, n]$
Inference rules:	$\dfrac{[B, i, j] \qquad [C, j, k]}{[A, i, k]} \quad A \rightarrow B\ C$

Figure 1: The CYK deductive parsing system.

set of axioms; a set of inference rules; and a subclass of items, the goal items. These are given in summary form in Figure 1.

This deduction system can be encoded straightforwardly by the following logic program:

```
nt(A, I1, I) :-
    word(I, W),
    (A ---> [W]),
    I1 is I - 1.
nt(A, I, K) :-
    nt(B, I, J),
    nt(C, J, K),
    (A ---> [B, C]).
```

where A `--->` `[`X_1`,...,`X_m`]` is the encoding of a production $A \rightarrow X_1 \cdots X_n$ in the grammar and `word(`i`,`w_i`)` holds for each input word w_i in the string to be parsed. A suitable bottom-up execution of this program, for example using the *semi-naïve* bottom-up procedure (Naughton and Ramakrishnan, 1991) will behave similarly to the CYK algorithm on the given grammar.

2.2 Proofs of Correctness

Rather than implement each deductive system like the CYK one as a separate logic program, we will describe in Section 5 a meta-interpreter for logic programs obtained from grammatical deduction systems. The meta-interpreter is just a variant of the semi-naïve procedure specialized to programs implementing grammatical deduction systems. We will show in Section 5 that our procedure generates only items derivable from the axioms (*soundness*) and will enumerate all the derivable items (*completeness*). Therefore, to show that a particular parsing algorithm is correctly simulated by our meta-interpreter, we basically need to show that the corresponding grammatical deduction system is also sound

and complete with respect to the intended interpretation of grammaticality items. By sound here we mean that every derivable item represents a true grammatical statement under the intended interpretation, and by complete we mean that the item encoding every true grammatical statement is derivable. (We also need to show that the grammatical deduction system is faithfully represented by the corresponding logic program, but in general this will be obvious by inspection.)

3 Deductive Parsing of Context-Free Grammars

We begin the presentation of parsing methods stated as deduction systems with several standard methods for parsing context-free grammars. In what follows, we assume that we are given a string $w = w_1 \cdots w_n$ to be parsed along with a context-free grammar $G = \langle N, \Sigma, P, S \rangle$.

3.1 Pure Top-Down Parsing (Recursive Descent)

The first full parsing algorithm for arbitrary context-free grammars that we present from this logical perspective is recursive-descent parsing. Given a context-free grammar $G = \langle N, \Sigma, P, S \rangle$, and a string $w = w_1 \cdots w_n$ to be parsed, we will consider a logic with items of the form $[\,\bullet\,\beta, j]$ where $0 \leq j \leq n$. Such an item asserts that the substring of the string w up to and including the j-th element, when followed by the string of symbols β, forms a sentential form of the language, that is, that $S \stackrel{*}{\Rightarrow} w_1 \cdots w_j \beta$. Note that the dot in the item is positioned just at the break point in the sentential form between the portion that has been recognized (up through index j) and the part that has not (β).

Taking the set of such items to be the [propositional] formulas of the logic, and taking the informal statement concluding the previous paragraph to provide a denotation for the sentences,[1] we can explore a proof theory for the logic. We start with an axiom

$$[\,\bullet\, S, 0] \qquad ,$$

which is sound because $S \stackrel{*}{\Rightarrow} S$ trivially.

Note that two items of the form $[\,\bullet\, w_{j+1}\beta, j]$ and $[\,\bullet\,\beta, j+1]$ make the same claim, namely that $S \stackrel{*}{\Rightarrow} w_1 \cdots w_j w_{j+1} \beta$. Thus, it is clearly sound to conclude the latter from the former, yielding the inference rule:

$$\frac{[\,\bullet\, w_{j+1}\beta, j]}{[\,\bullet\,\beta, j+1]} \qquad ,$$

[1] A more formal statement of the semantics could be given, e.g., as

$$[\![[\,\bullet\,\beta, j]]\!] = \begin{cases} \textit{truth} & \text{if } S \stackrel{*}{\Rightarrow} w_1 \cdots w_j \beta \\ \textit{falsity} & \text{otherwise} \end{cases} \quad .$$

Item form:	$[\bullet\,\beta, j]$
Axioms:	$[\bullet\, S, 0]$
Goals:	$[\bullet\,, n]$
Inference rules:	
Scanning	$\dfrac{[\bullet\, w_{j+1}\beta, j]}{[\bullet\,\beta, j+1]}$
Prediction	$\dfrac{[\bullet\, B\beta, j]}{[\bullet\,\gamma\beta, j]} \quad B \rightarrow \gamma$

Figure 2: The top-down recursive-descent deductive parsing system.

which we will call the *scanning* rule.

A similar argument shows the soundness of the *prediction* rule:

$$\frac{[\bullet\, B\beta, j]}{[\bullet\,\gamma\beta, j]} \quad B \rightarrow \gamma \qquad .$$

Finally, the item $[\bullet\,, n]$ makes the claim that $S \stackrel{*}{\Rightarrow} w_1 \cdots w_n$, that is, that the string w is admitted by the grammar. Thus, if this goal item can be proved from the axiom by the inference rules, then the string must be in the grammar. Such a proof process would constitute a sound recognition algorithm. As it turns out, the recognition algorithm that this logic of items specifies is a pure top-down left-to-right regime, a recursive-descent algorithm. The four components of the deduction system for top-down parsing — class of items, axioms, inference rules, and goal items — are summarized in Figure 2.

To illustrate the operation of these inference rules for context-free parsing, we will use the toy grammar of Figure 3. Given that grammar and the string

$$w_1 w_2 w_3 = \text{a program halts} \tag{1}$$

S	$\rightarrow$	$NP\ VP$	Det	$\rightarrow$	a
NP	$\rightarrow$	$Det\ N\ OptRel$	N	$\rightarrow$	program
NP	$\rightarrow$	PN	PN	$\rightarrow$	Terry
VP	$\rightarrow$	$TV\ NP$	PN	$\rightarrow$	Shrdlu
VP	$\rightarrow$	IV	IV	$\rightarrow$	halts
$OptRel$	$\rightarrow$	$RelPro\ VP$	TV	$\rightarrow$	writes
$OptRel$	$\rightarrow$	ϵ	$RelPro$	$\rightarrow$	that

Figure 3: An example context-free grammar.

we can construct the following derivation using the rules just given:

1	$[\bullet\, S, 0]$	AXIOM
2	$[\bullet\, NP\ VP, 0]$	PREDICT from 1
3	$[\bullet\, Det\ N\ OptRel\ VP, 0]$	PREDICT from 2
4	$[\bullet\,\text{a}\ N\ OptRel\ VP, 0]$	PREDICT from 3
5	$[\bullet\, N\ OptRel\ VP, 1]$	SCAN from 4
6	$[\bullet\,\text{program}\ OptRel\ VP, 1]$	PREDICT from 5
7	$[\bullet\, OptRel\ VP, 2]$	SCAN from 6
8	$[\bullet\, VP, 2]$	PREDICT from 7
9	$[\bullet\, IV, 2]$	PREDICT from 8
10	$[\bullet\,\text{halts}, 2]$	PREDICT from 9
11	$[\bullet\,, 3]$	SCAN from 10

The last item is a goal item, showing that the given sentence is accepted by the grammar of Figure 3.

The above derivation, as all the others we will show, contains just those items that are strictly necessary to derive a goal item from the axiom. In general, a complete search procedure, such as the one we describe in Section 5, generates items that are either dead-ends or redundant for a proof of grammaticality. Furthermore, with an ambiguous grammar there will be several essentially different proofs of grammaticality, each corresponding to a different analysis of the input string.

3.1.1 Proof of Completeness

We have shown informally above that the inference rules for top-down parsing are sound, but for any such system we also need the guarantee of *completeness*:

if a string is admitted by the grammar, then for that string there is a derivation of a goal item from the initial item.

In order to prove completeness, we prove the following lemma: If $S \stackrel{*}{\Rightarrow} w_1 \cdots w_j \gamma$ is a leftmost derivation (where $\gamma \in V^*$), then the item $[\bullet\, \gamma, j]$ is generated. We must prove all possible instances of this lemma. Any specific instance can be characterized by specifying the string γ and the integer j, since S and $w_1 \cdots w_j$ are fixed. We shall denote such an instance by $\langle \gamma, j \rangle$. The proof will turn on ranking the various instances and proving the result by induction on the rank. The rank of the instance $\langle \gamma, j \rangle$ is computed as the sum of j and the length of a shortest leftmost derivation of $S \stackrel{*}{\Rightarrow} w_1 \cdots w_j \gamma$.

If the rank is zero, then $j = 0$ and $\gamma = S$. Then, we need to show that $[\bullet S, 0]$ is generated, which is the case since it is an axiom of the top-down deduction system.

For the inductive step, let $\langle \gamma, j \rangle$ be an instance of the lemma of some rank $r > 0$, and assume that the lemma is true for all instances of smaller rank. Two cases arise.

Case 1: $S \stackrel{*}{\Rightarrow} w_1 \cdots w_j \gamma$ in one step. Therefore, $S \to w_1 \cdots w_j \gamma$ is a rule of the grammar. However, since $[\bullet\, S, 0]$ is an axiom, by one application of the prediction rule (predicting the rule $S \to w_1 \cdots w_j \gamma$) and j applications of the scanning rule, the item $[\bullet\, \gamma, j]$ will be generated.

Case 2: $S \stackrel{*}{\Rightarrow} w_1 \cdots w_j \gamma$ in more than one step. Let us assume therefore that $S \stackrel{*}{\Rightarrow} w_1 \cdots w_{j-k} B \gamma' \Rightarrow w_1 \cdots w_j \beta \gamma'$ where $\gamma = \beta\gamma'$ and $B \to w_{j-k+1} \cdots w_j \beta$. The instance $\langle B\gamma', j-k \rangle$ has a strictly smaller rank than $\langle \gamma, j \rangle$. Therefore, by the induction hypothesis, the item $[\bullet\, B\gamma', j-k]$ will be generated. But then, by prediction, the item $[\bullet\, w_{j-k+1} \cdots w_j \beta, j-k]$ will be generated and by k applications of the scanning rule, the item $[\bullet B, j]$ will be generated.

This concludes the proof of the lemma. Completeness of the parser follows as a corollary of the lemma since if $S \stackrel{*}{\Rightarrow} w_1 \cdots w_n$, then by the lemma the item $[\bullet, n]$ will be generated.

Completeness proofs for the remaining parsing logics discussed in this paper could be provided in a similar way by relating an appropriate notion of normal-form derivation for the grammar formalism under consideration to the item invariants.

3.2 Pure Bottom-Up Parsing (Shift-Reduce)

A pure bottom-up algorithm can be specified by such a deduction system as well. Here, the items will have the form $[\alpha \bullet, j]$. Such an item asserts the dual of the assertion made by the top-down items, that $\alpha w_{j+1} \cdots w_n \stackrel{*}{\Rightarrow} w_1 \cdots w_n$ (or, equivalently but less transparently dual, that $\alpha \stackrel{*}{\Rightarrow} w_1 \cdots w_j$). The algorithm is

Item form: $[\alpha \bullet, j]$

Axioms: $[\bullet, 0]$

Goals: $[S \bullet, n]$

Inference Rules:

Shift $$\frac{[\alpha \bullet, j]}{[\alpha w_{j+1} \bullet, j+1]}$$

Reduce $$\frac{[\alpha\gamma \bullet, j]}{[\alpha B \bullet, j]} \quad B \to \gamma$$

Figure 4: The bottom-up shift-reduce deductive parsing system.

then characterized by the deduction system shown in Figure 4. The algorithm mimics the operation of a nondeterministic shift-reduce parsing mechanism, where the string of symbols preceding the dot corresponds to the current parse stack, and the substring starting at the index j corresponds to the as yet unread input.

The soundness of the inference rules in Figure 4 is easy to see. The antecedent of the shift rule claims that $\alpha w_{j+1} \cdots w_n \stackrel{*}{\Rightarrow} w_1 \cdots w_n$, but that is also what the consequent claims. For the reduce rule, if $\alpha\gamma w_{j+1} \cdots w_n \stackrel{*}{\Rightarrow} w_1 \cdots w_n$ and $B \to \gamma$, then by definition of $\stackrel{*}{\Rightarrow}$ we also have $\alpha B w_{j+1} \cdots w_n \stackrel{*}{\Rightarrow} w_1 \cdots w_n$. As for completeness, it can be proved by induction on the steps of a reversed rightmost context-free derivation in a way very similar to the completeness proof of the last section.

The following derivation shows the operation of the bottom-up rules on example sentence (1):

1	$[\bullet, 0]$	AXIOM
2	$[\text{a} \bullet, 1]$	SHIFT from 1
3	$[\mathit{Det} \bullet, 1]$	REDUCE from 2
4	$[\mathit{Det}\ \text{program} \bullet, 2]$	SHIFT from 3
5	$[\mathit{Det}\ N \bullet, 2]$	REDUCE from 4
6	$[\mathit{Det}\ N\ \mathit{OptRel} \bullet, 2]$	REDUCE from 5
7	$[\mathit{NP} \bullet, 2]$	REDUCE from 6
8	$[\mathit{NP}\ \text{halts} \bullet, 3]$	SHIFT from 7
9	$[\mathit{NP}\ \mathit{IV} \bullet, 3]$	REDUCE from 8
10	$[\mathit{NP}\ \mathit{VP} \bullet, 3]$	REDUCE from 9
11	$[S \bullet, 3]$	REDUCE from 10

The last item is a goal item, which shows that the sentence is parsable according to the grammar.

3.3 Earley's Algorithm

Stating the algorithms in this way points up the duality of recursive-descent and shift-reduce parsing in a way that traditional presentations do not. The summary presentation in Figure 5 may further illuminate the various interrelationships. As we will see, Earley's algorithm (Earley, 1970) can then be seen as the natural combination of these two algorithms.

In recursive-descent parsing, we keep a partial sentential form for the material yet to be parsed, using the dot at the beginning of the string of symbols to remind us that these symbols come after the point that we have reached in the recognition process. In shift-reduce parsing, we keep a partial sentential form for the material that has already been parsed, placing a dot at the end of the string to remind us that these symbols come before the point that we have reached in the recognition process. In Earley's algorithm we keep both of these partial sentential forms, with the dot marking the point somewhere in the middle where recognition has reached. The dot thus changes from a mnemonic to a necessary role. In addition, Earley's algorithm localizes the piece of sentential form that is being tracked to that introduced by a single production. (Because the first two parsers do not limit the information stored in an item to only local information, they are not practical algorithms as stated. Rather some scheme for sharing the information among items would be necessary to make them tractable.)

The items of Earley's algorithm are thus of the form $[i, A \to \alpha \bullet \beta, j]$ where α and β are strings in V^* and $A \to \alpha\beta$ is a production of the grammar. As was the case for the previous two algorithms, the j index provides the position in the string that recognition has reached, and the dot position marks that point in the partial sentential form. In these items, however, an extra index i marks the starting position of the partial sentential form, as we have localized attention to a single production. In summary, an item of the form $[i, A \to \alpha \bullet \beta, j]$ makes the top-down claim that $S \stackrel{*}{\Rightarrow} w_1 \cdots w_i A \gamma$, and the bottom-up claim that $\alpha w_{j+1} \cdots w_n \stackrel{*}{\Rightarrow} w_{i+1} \cdots w_n$. The two claims are connected by the fact that $A \to \alpha\beta$ is a production in the grammar.

The algorithm itself is captured by the specification found in Figure 5. Proofs of soundness and completeness are somewhat more complex than those for the pure top-down and bottom-up cases shown above, and are directly related to the corresponding proofs for Earley's original algorithm (Earley, 1970).

The following derivation, again for sentence (1), illustrates the operation of

Algorithm	*Bottom-Up*	*Top-Down*	*Earley's*
Item form	$[\alpha \bullet, j]$	$[\bullet \beta, j]$	$[i, A \to \alpha \bullet \beta, j]$
Invariant	$\alpha w_{j+1} \cdots w_n \stackrel{*}{\Rightarrow} w_1 \cdots w_n$	$S \stackrel{*}{\Rightarrow} w_1 \cdots w_j \beta$	$S \stackrel{*}{\Rightarrow} w_1 \cdots w_i A \gamma$ $\alpha w_{j+1} \cdots w_n \stackrel{*}{\Rightarrow} w_{i+1} \cdots w_n$
Axioms	$[\bullet, 0]$	$[\bullet S, 0]$	$[0, S' \to \bullet S, 0]$
Goals	$[S \bullet, n]$	$[\bullet, n]$	$[0, S' \to S \bullet, n]$
Scanning	$\frac{[\alpha \bullet, j]}{[\alpha w_{j+1} \bullet, j+1]}$	$\frac{[\bullet w_{j+1} \beta, j]}{[\bullet \beta, j+1]}$	$\frac{[i, A \to \alpha \bullet w_{j+1} \beta, j]}{[i, A \to \alpha w_{j+1} \bullet \beta, j+1]}$
Prediction		$\frac{[\bullet B \beta, j]}{[\bullet \gamma \beta, j]} \quad B \to \gamma$	$\frac{[i, A \to \alpha \bullet B \beta, j]}{[j, B \to \bullet \gamma, j]} \quad B \to \gamma$
Completion	$\frac{[\alpha \gamma \bullet, j]}{[\alpha B \bullet, j]} \quad B \to \gamma$		$\frac{[i, A \to \alpha \bullet B \beta, k] \quad [k, B \to \gamma \bullet, j]}{[i, A \to \alpha B \bullet \beta, j]}$

Figure 5: Summary of parsing algorithms presented as deductive parsing systems. (In the axioms and goal items of Earley's algorithm, S' serves as a new nonterminal not in N.)

the Earley inference rules:

1	$[0, S' \to \bullet\, S, 0]$	AXIOM
2	$[0, S \to \bullet\, NP\ VP, 0]$	PREDICT from 1
3	$[0, NP \to \bullet\, Det\ N\ OptRel, 0]$	PREDICT from 2
4	$[0, Det \to \bullet\, \mathrm{a}, 0]$	PREDICT from 3
5	$[0, Det \to \mathrm{a} \bullet, 1]$	SCAN from 4
6	$[0, NP \to Det \bullet N\ OptRel, 1]$	COMPLETE from 3 and 5
7	$[1, N \to \bullet\, \mathrm{program}, 1]$	PREDICT from 6
8	$[1, N \to \mathrm{program} \bullet, 2]$	SCAN from 7
9	$[0, NP \to Det\ N \bullet OptRel, 2]$	COMPLETE from 6 and 8
10	$[2, OptRel \to \bullet, 2]$	PREDICT from 9
11	$[0, NP \to Det\ N\ OptRel \bullet, 2]$	COMPLETE from 9 and 10
12	$[0, S \to NP \bullet VP, 2]$	COMPLETE from 2 and 11
13	$[2, VP \to \bullet\, IV, 2]$	PREDICT from 12
14	$[2, IV \to \bullet\, \mathrm{halts}, 2]$	PREDICT from 13
15	$[2, IV \to \mathrm{halts} \bullet, 3]$	SCAN from 14
16	$[2, VP \to IV \bullet, 3]$	COMPLETE from 13 and 15
17	$[0, S \to NP\ VP \bullet, 3]$	COMPLETE from 12 and 16
18	$[0, S' \to S \bullet, 3]$	COMPLETE from 1 and 17

The last item is again a goal item, so we have an Earley derivation of the grammaticality of the given sentence.

4 Deductive Parsing for Other Formalisms

The methods (and implementation) that we developed have also been used for rapid prototyping and experimentation with parsing algorithms for grammatical frameworks other than context-free grammars. They can be naturally extended to handle augmented phrase-structure formalisms such as logic grammar and constraint-based formalisms. They have been used in the development and testing of algorithms for parsing categorial grammars, tree-adjoining grammars, and lexicalized context-free grammars. In this section, we discuss these and other extensions.

4.1 Augmented Phrase-Structure Formalisms

It is straightforward to see that the three deduction systems just presented can be extended to constraint-based grammar formalisms with a context-free backbone. The basis for this extension goes back to metamorphosis grammars (Colmerauer, 1978) and definite-clause grammars (DCG) (Pereira and Warren, 1980). In those formalisms, grammar symbols are first-order terms, which can be understood as abbreviations for the sets of all their ground instances. Then

an inference rule can also be seen as an abbreviation for all of its ground instances, with the metagrammatical variables in the rule consistently instantiated to ground terms. Computationally, however, such instances are generated lazily by accumulating the consistency requirements for the instantiation of inference rules as a conjunction of equality constraints and maintaining that conjunction in normal form — sets of variable substitutions — by unification. (This is directly related to the use of unification to avoid "guessing" instances in the rules of existential generalization and universal instantiation in a natural-deduction presentation of first-order logic).

We can move beyond first-order terms to general constraint-based grammar formalisms (Shieber, 1992; Carpenter, 1992) by taking the above constraint interpretation of inference rules as basic. More explicitly, a rule such as Earley completion

$$\frac{[i, A \to \alpha \bullet B\beta, k] \quad [k, B \to \gamma \bullet, j]}{[i, A \to \alpha B \bullet \beta, j]}$$

is interpreted as shorthand for the constrained rule:

$$\frac{[i, A \to \alpha \bullet B\beta, k] \quad [k, B' \to \gamma \bullet, j]}{[i, A' \to \alpha B'' \bullet \beta, j]} \qquad A = A' \text{ and } B = B' \text{ and } B = B''$$

When such a rule is applied, the three constraints it depends on are conjoined with the constraints for the current derivation. In the particular case of first-order terms and antecedent-to-consequent rule application, completion can be given more explicitly as

$$\frac{[i, A \to \alpha \bullet B\beta, k] \quad [k, B' \to \gamma \bullet, j]}{[i, \sigma(A \to \alpha B \bullet \beta), j]} \qquad \sigma = \mathrm{mgu}(B, B') \qquad .$$

where $mgu(B, B')$ is the most general unifier of the terms B and B'. This is the interpretation implemented by the deduction procedure described in the next section.

The move to constraint-based formalisms raises termination problems in proof construction that did not arise in the context-free case. In the general case, this is inevitable, because a formalism like DCG (Pereira and Warren, 1980) or PATR-II (Shieber, 1985a) has Turing-machine power. However, even if constraints are imposed on the context-free backbone of the grammar productions to guarantee decidability, such as *offline parsability* (Bresnan and Kaplan, 1982; Pereira and Warren, 1983; Shieber, 1992), the prediction rules for the top-down and Earley systems are problematic. The difficulty is that prediction can feed on its own results to build unboundedly large items. For example, consider the DCG

$$\begin{array}{l} s \to r(0, N) \\ r(X, N) \to r(s(X), N)\ b \\ r(N, N) \to a \end{array}$$

It is clear that this grammar accepts strings of the form ab^n with the variable N being instantiated to the unary (successor) representation of n. It is also clear that the bottom-up inference rules will have no difficulty in deriving the analysis of any input string. However, Earley prediction from the item $[0, s \rightarrow \bullet\, r(0, N), 0]$ will generate an infinite succession of items:

$$
\begin{array}{l}
[0, s \rightarrow \bullet\, r(0, N), 0] \\
[0, r(0, N) \rightarrow \bullet\, r(s(0), N)\ b, 0] \\
[0, r(s(0), N) \rightarrow \bullet\, r(s(s(0)), N)\ b, 0] \\
[0, r(s(s(0)), N) \rightarrow \bullet\, r(s(s(s(0))), N)\ b, 0] \\
\cdots
\end{array}
$$

This problem can be solved in the case of the Earley inference rules by observing that prediction is just used to narrow the number of items to be considered by scanning and completion, by maintaining the top-down invariant $S \stackrel{*}{\Rightarrow} w_1 \cdots w_i A\gamma$. But this invariant is not required for soundness or completeness, since the bottom-up invariant is sufficient to guarantee that items represent well-formed substrings of the input. The only purpose of the top-down invariant is to minimize the number of completions that are actually attempted. Thus the only indispensable role of prediction is to make available appropriate instances of the grammar productions. Therefore, any relaxation of prediction that makes available items of which all the items predicted by the original prediction rule are instances will not affect soundness or completeness of the rules. More precisely, it must be the case that any item $[i, B \rightarrow \bullet\, \gamma, i]$ that the original prediction rule would create is an instance of some item $[i, B' \rightarrow \bullet\, \gamma', i]$ created by the relaxed prediction rule. A relaxed prediction rule will create no more items than the original predictor, and in fact may create far fewer. In particular, repeated prediction may terminate in cases like the one described above. For example, if the prediction rule applied to $[i, A \rightarrow \alpha \bullet B'\beta, j]$ yields $[i, \sigma(B \rightarrow \bullet\, \gamma), i]$ where $\sigma = \mathrm{mgu}(B, B')$, a relaxed prediction rule might yield $[i, \sigma'(B \rightarrow \bullet\, \gamma), i]$, where σ' is a less specific substitution than σ chosen so that only a finite number of instances of $[i, B \rightarrow \bullet\, \gamma, i]$ are ever generated. A similar notion for general constraint grammars is called restriction (Shieber, 1985b; Shieber, 1992), and a related technique has been used in partial evaluation of logic programs (Sato and Tamaki, 1984).

The problem with the DCG above can be seen as following from the computation of derivation-specific information in the arguments to the nonterminals. However, applications frequently require construction of the derivation for a string (or similar information), perhaps for the purpose of further processing. It is simple enough to augment the inference rules to include with each item a derivation. For the Earley deduction system, the items would include a fourth component whose value is a sequence of derivation trees, nodes labeled by productions of the grammar, one derivation tree for each element of the right-hand side of the item before the dot. The inference rules would be modified as shown in Figure 6. The system makes use of a function *tree* that takes a node label l

Item form:	$[i, A\alpha \bullet \beta, j, D]$	
Axioms:	$[0, S' \rightarrow \bullet\, S, 0, \langle\rangle]$	
Goals:	$[0, S' \rightarrow S \bullet, n, D]$	
Inference rules:		
Scanning	$\dfrac{[i, A \rightarrow \alpha \bullet w_{j+1}\beta, j, D]}{[i, A \rightarrow \alpha w_{j+1} \bullet \beta, j+1, D]}$	
Prediction	$\dfrac{[i, A \rightarrow \alpha \bullet B\beta, j, D]}{[j, B \rightarrow \bullet\, \gamma, j, \langle\rangle]}$	$B \rightarrow \gamma$
Completion	$\dfrac{[i, A \rightarrow \alpha \bullet B\beta, k, D_1] \quad [k, B \rightarrow \gamma \bullet, j, D_2]}{[i, A \rightarrow \alpha B \bullet \beta, j, D_1 \cup \mathit{tree}(B \rightarrow \gamma, D_2)]}$	

Figure 6: The Earley deductive parsing system modified to generate derivation trees.

(a production in the grammar) and a sequence of derivation trees D and forms a tree whose root is labeled by l and whose children are the trees in D in order.

Of course, use of such rules makes the caching of lemmas essentially useless, as lemmas derived in different ways are never identical. Appropriate methods of implementation that circumvent this problem are discussed in Section 5.4.

4.2 Combinatory Categorial Grammars

A combinatory categorial grammar (Ades and Steedman, 1982) consists of two parts: (1) a lexicon that maps words to sets of categories; (2) rules for combining categories into other categories.

Categories are built from atomic categories and two binary operators: forward slash (/) and backward slash (\). Informally speaking, words having categories of the form X/Y, $X\backslash Y$, $(W/X)/Y$ etc. are to be thought of as functions over Y's. Thus the category $S\backslash NP$ of intransitive verbs should be interpreted as a function from noun phrases (NP) to sentences (S). In addition, the direction of the slash (forward as in X/Y or backward as in $X\backslash Y$) specifies where the argument must be found, immediately to the right for / or immediately to the left for \.

For example, a CCG lexicon may assign the category $S\backslash NP$ to an intransitive verb (as the word *sleeps*). $S\backslash NP$ identifies the word (*sleeps*) as combining with a (subject) noun phrase (NP) to yield a sentence (S). The back slash (\) indicates that the subject must be found immediately to the left of the verb. The forward