PARSING AS DEDUCTION[1]

Fernando C. N. Pereira
David H. D. Warren
Artificial Intelligence Center
SRI International
333 Ravenswood Ave., Menlo Park CA 94025

Abstract

By exploring the relationship between parsing and deduction, a new and more general view of chart parsing is obtained, which encompasses parsing for grammar formalisms based on unification, and is the basis of the Earley Deduction proof procedure for definite clauses. The efficiency of this approach for an interesting class of grammars is discussed.

1. Introduction

The aim of this paper is to explore the relationship between parsing and deduction. The basic notion, which goes back to Kowalski (Kowalski, 1980) and Colmerauer (Colmerauer, 1978), has seen a very efficient, if limited, realization in the use of the logic programming language Prolog for parsing (Colmerauer, 1978; Pereira and Warren, 1980). The connection between parsing and deduction was developed further in the design of the Earley Deduction proof procedure (Warren, 1975), which will also be discussed at length here.

Investigation of the connection between parsing and deduction yields several important benefits:

- A theoretically clean mechanism to connect parsing with the inference needed for semantic interpretation.
- Handling of gaps and unbounded dependencies "on the fly" without adding special mechanisms to the parser.
- A reinterpretation and generalization of chart parsing that abstracts from unessential data-structure details.
- Techniques that are applicable to parsing in related formalisms not directly based on logic.
- Elucidation of parsing complexity issues for related formalisms, in particular lexical-functional grammar (LFG).

Our study of these topics is still far from complete; therefore, besides offering some initial results, we shall discuss various outstanding questions.

The connection between parsing and deduction is based on the axiomatization of context-free grammars in **definite clauses**, a particularly simple subset of first-order logic (Kowalski, 1980; van Emden and Kowalski, 1976). This axiomatization allows us to identify context-free parsing algorithms with proof procedures for a restricted class of definite clauses, those derived from context-free rules. This identification can then be generalized to include larger classes of definite clauses to which the same algorithms can be applied, with simple modifications. Those larger classes of definite clauses can be seen as grammar formalisms in which the atomic grammar symbols of context-free grammars have been replaced by complex symbols that are matched by unification (Robinson, 1965; Colmerauer, 1978; Pereira and Warren, 1980). The simplest of these formalisms is definite-clause grammars (DCG) (Pereira and Warren, 1980).

There is a close relationship between DCGs and other grammar formalisms based on unification, such as Unification Grammar (UG) (Kay, 1979), LFG, PATR-2 (Shieber, 1983) and the more recent versions of GPSG (Gazdar and Pullum, 1982).

The parsing algorithms we are concerned with are **online** algorithms, in the sense that they apply the constraints specified by the augmentation of a rule as soon as the rule is applied. In contrast, an **offline** parsing algorithm will consist of two phases: a context-free parsing algorithm followed by application of the constraints to all the resulting analyses.

The paper is organized as follows. Section 2 gives an overview of the concepts of definite clause logic, definite clause grammars, definite clause proof procedures, and chart parsing. Section 3 discusses the connection betwee DCGs and LFG. Section 4 describes the Earley Deduction definite-clause proof procedure. Section 5 then brings out the connection between Earley Deduction and chart parsing, and shows the added generality brought in by the proof procedure approach. Section 6 outlines some of the problems of implementing Earley Deduction and similar parsing procedures. Finally, Section 7 discusses questions of computational complexity and decidability.

[1]This work was partially supported by the Defense Advanced Research Projects Agency under Contract N00039-80-C-0575 with the Naval Electronic Systems Command. The views and conclusions contained in this article are those of the authors and should not be interpreted as representative of the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the United States Government.

2. Basic Notions

2.1. Definite Clauses

A **definite clause** has the form

$$P \Leftarrow Q_1 \ \& \ \ldots \ \& \ Q_n.$$

to be read as "P is true if Q_1 and ... and Q_n are true". If $n = 0$, the clause is a **unit** clause and is written simply as

$$P.$$

P and $Q_1, \ldots, Q_n$ are **literals**. P is the **positive** literal or **head** of the clause; $Q_1, \ldots, Q_n$ are the **negative** literals, forming the **body** of the clause. Literals have the form $p(t_1,\ldots,t_k)$, where p is the **predicate** of arity k and the t_i the **arguments**. The arguments are **terms**. A term may be: a **variable** (variable names start with capital letters); a **constant**; a **compound term** $f(t_1,\ldots,t_m)$ where f is a functor of arity m and the t_i are terms. All the variables in a clause are implicitly universally quantified.

A set of definite clauses forms a **program**, and the clauses in a program are called **input clauses**. A program defines the relations denoted by the predicates appearing in the heads of clauses. When using a definite-clause proof procedure, such as Prolog (Roussel, 1975), a **goal statement**

$$\Leftarrow P.$$

requests the proof procedure to find provable instances of P.

2.2. Definite Clause Grammars

Any context-free rule

$$X \rightarrow \alpha_1 \ldots \alpha_n$$

can be translated into a definite clause

$$x(S_0,S_n) \Leftarrow \alpha_1(S_0,S_1) \ \& \ \ldots \ \& \ \alpha_n(S_{n-1},S_n).$$

The variables S_i are the **string arguments**, representing positions in the input string. For example, the context-free rule "S → NP VP" is translated into "s(S0,S2) ⇐ np(S0,S1) & vp(S1,S2)," which can be paraphrased as "there is an S from S0 to S2 in the input string if there is an NP from S0 to S1 and a VP from S1 to S2."

Given the translation of a context-free grammar G with start symbol S into a set of definite clauses G′ with corresponding predicate s, to say that a string w is in the grammar's language is equivalent to saying that the **start goal** $s(p_0,p)$ is a consequence of G′ ∪ W, where p_0 and p represent the left and right endpoints of w, and W is a set of unit clauses that represents w.

It is easy to generalize the above notions to define DCGs. DCG nonterminals have arguments in the same way that predicates do. A DCG nonterminal with **n** arguments is translated into a predicate of **n+2** arguments, the last two of which are the string points, as in the translation of context-free rules into definite clauses. The context-free grammar obtained from a DCG by dropping all nonterminal arguments is the **context-free skeleton** of the DCG.

2.3. Deduction in Definite Clauses

The fundamental inference rule for definite clauses is the following **resolution** rule: From the clauses

$$B \Leftarrow A_1 \ \& \ \ldots \ \& \ A_m. \tag{1}$$
$$C \Leftarrow D_1 \ \& \ \ldots \ \& \ D_i \ \& \ \ldots \ \& \ D_n. \tag{2}$$

when B and D_i are unifiable by substitution σ, infer

$$\sigma[C \Leftarrow D_1 \ \& \ \ldots \ D_{i-1} \ \& \ A_1 \ \& \ \ldots \ \& \ A_m \ \& \ D_{i+1} \ \ldots \ \& \ D_n.] \tag{3}$$

Clause (3) is a **derived** clause, the **resolvent** of (1) and (2).

The proof procedure of Prolog is just a particular embedding of the resolution rule in a search procedure, in which a goal clause like (2) is successively rewritten by the resolution rule using clauses from the program (1). The Prolog proof procedure can be implemented very efficiently, but it has the same theoretical problems of the top-down backtrack parsing algorithms after which it is modeled. These problems do not preclude its use for creating uniquely efficient parsers for suitably constructed grammars (Warren and Pereira, 1983; Pereira, 1982), but the broader questions of the relation between parsing and deduction and of the derivation of online parsing algorithms for unification formalisms require that we look at a more generally applicable class of proof procedures.

2.4. Chart Parsing and the Earley Algorithm

Chart parsing is a general framework for constructing parsing algorithms for context-free grammars and related formalisms. The Earley context-free parsing algorithm, although independently developed, can be seen as a particular case of chart parsing. We will give here just the basic terminology of chart parsing and of the Earley algorithm. Full accounts can be found in the articles by Kay (Kay, 1980) and Earley (Earley, 1970).

The state of a chart parser is represented by the **chart**, which is a directed graph. The nodes of the chart represent positions in the string being analyzed. Each edge in the chart is either **active** or **passive**. Both types of edges are labeled. A passive edge with label N links node r to node s if the string between r and s has been analyzed as a phrase of type N. Initially, the only edges are passive edges that link consecutive nodes and are labeled with the words of the input string (see Figure 1). Active edges represent partially applied grammar rules. In the simplest case, active edges are labeled by **dotted** rules. A dotted rule is a grammar rule with a dot inserted somewhere on its right-hand side

$$X \rightarrow \alpha_1 \ldots \alpha_{i-1} \cdot \alpha_i \ldots \alpha_n \tag{4}$$

An edge with this label links node r to node s if the sentential form $\alpha_1 \ldots \alpha_{i-1}$ is an analysis of the input string between r and s. An active edge that links a node to

itself is called **empty** and acts like a top-down prediction. Chart-parsing procedures start with a chart containing the passive edges for the input string. New edges are added in two distinct ways. First, an active edge from r to s labeled with a dotted rule (4) combines with a passive edge from s to t with label α_i to produce a new edge from r to t, which will be a passive edge with label X if α_i is the last symbol in the right-hand side of the dotted rule; otherwise it will be an active edge with the dot advanced over α_i. Second, the parsing strategy must place into the chart, at appropriate points, new empty active edges that will be used to combine existing passive edges. The exact method used determines whether the parsing method is seen as top-down, bottom-up, or a combination of the two.

The Earley parsing algorithm can be seen as a special case of chart parsing in which new empty active edges are introduced top-down and, for all k, the edge combinations involving only the first k nodes are done before any combinations that involve later nodes. This particular strategy allows certain simplifications to be made in the general algorithm.

3. DCGs and LFG

We would like to make a few informal observations at this point to clarify the relationship between DCGs and other unification grammar formalisms — LFG in particular. A more detailed discussion would take us beyond the intended scope of this paper.

The different notational conventions of DCGs and LFG make the two formalisms less similar on the surface than they actually are from the computational point of view. The objects that appear as arguments in DCG rules are tree fragments every node of which has a number of children predetermined by the functor that labels the node. Explicit variables mark unspecified parts of the tree. In contrast, the functional structure nodes that are implicitly mentioned in LFG equations do not have a predefined number of children, and unspecified parts are either omitted or defined implicitly through equations.

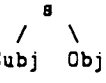
As a first approximation, a DCG rule such as

s(s(Subj,Obj)) → np(Subj) vp(Obj) (5)

might correspond to the LFG rule

S → NP VP (6)
↑ subj = ↓ ↑ obj = ↓

The DCG rule can be read as "an *s* with structure



is an *np* with structure *Subj* followed by a *vp* with structure *Obj*." The LFG rule can be read as "an S is an NP followed by a VP, where the value of the *subj* attribute of the S is the functional structure of the NP and the value of the attribute *obj* of the S is the functional structure of the VP." For those familiar with the details of the mapping from functional descriptions to functional structures in LFG, DCG variables are just "placeholder" symbols (Bresnan and Kaplan, 1982).

As we noted above, an apparent difference between LFG and DCGs is that LFG functional structure nodes, unlike DCG function symbols, do not have a definite number of children. Although we must leave to a separate paper the details of the application to LFG of the unification algorithms from theorem proving, we will note here that the formal properties of logical and LFG or UG unification are similar, and there are adaptations to LFG and UG of the algorithms and data structures used in the logical case.

4. Earley Deduction

The Earley Deduction proof procedure schema is named after Earley's context-free parsing algorithm (Earley, 1970), on which it is based. Earley Deduction provides for definite clauses the same kind of mixed top-down bottom-up mechanism that the Earley parsing algorithm provides for context-free grammars.

Earley Deduction operates on two sets of definite clauses called the **program** and the **state**. The program is just the set of **input clauses** and remains fixed. The state consists of a set of **derived clauses**, where each nonunit clause has one of its negative literals **selected**; the state is continually being added to. Whenever a nonunit clause is added to the state, one of its negative literals is selected. Initially the state contains just the goal statement (with one of its negative literals selected).

There are two inference rules, called **instantiation** and **reduction**, which can map the current state into a new one by adding a new derived clause. For an instantiation step, there is some clause in the current state whose selected literal unifies with the positive literal of a *nonunit* clause C in the program. In this case, the derived clause is $\sigma[C]$, where σ is a most general unifier (Robinson, 1965) of the two literals concerned. The selected literal is said to **instantiate** C to $\sigma[C]$.

For a reduction step, there is some clause C in the current state whose selected literal unifies with a *unit* clause from either the program *or* the current state. In this case, the derived clause is $\sigma[C']$, where σ is a most general unifier of the two literals concerned, and C' is C minus its selected literal. Thus, the derived clause is just the resolvent of C with the unit clause and the latter is said to **reduce** C to $\sigma[C']$.

Before a derived clause is added to the state, a check is made to see whether the derived clause is subsumed by any clause already in the state. If the derived clause is subsumed, it is not added to the state, and that inference step is said to be **blocked**.

In the examples that follow, we assume that the selected literal in a derived clause is always the leftmost literal in the body. This choice is not optimal (Kowalski, 1980), but it is sufficient for our purposes.

For example, given the program

$$c(X,Z) \Leftarrow c(X,Y) \;\&\; c(Y,Z). \quad (7)$$
$$c(1,2). \quad (8)$$
$$c(2,3). \quad (9)$$

and goal statement

$$ans(Z) \Leftarrow c(1,Z). \quad (10)$$

here is a sequence of clauses derived by Early Deduction

```
ans(Z) ⇐ c(1,Z).                goal statement              (11)
c(1,Z) ⇐ c(1,Y) & c(Y,Z).       (11) instantiates (7)       (12)
ans(2).                         (8) reduces (11)            (13)
c(1,Z) ⇐ c(2,Z).                (8) reduces (12)            (14)
c(2,Z) ⇐ c(2,Y) & c(Y,Z).       (14) instantiates (7)       (15)
c(1,3).                         (9) reduces (14)            (16)
ans(3).                         (16) reduces (11)           (17)
c(2,Z) ⇐ c(3,Z).                (9) reduces (15)            (18)
c(3,Z) ⇐ c(3,Y) & c(Y,Z).       (18) instantiates (7)       (19)
```

At this point, all further steps are blocked, so the computation terminates.

Earley Deduction generalizes Earley parsing in a direct and natural way. Instantiation is analogous to the "predictor" operation of Earley's algorithm, while reduction corresponds to the "scanner" and "completer" operations. The "scanner" operation amounts to reduction with an input unit clause representing a terminal symbol occurrence, while the "completer" operation amounts to reduction with a derived unit clause representing a nonterminal symbol occurrence.

5. Chart Parsing and Earley Deduction

Chart parsing (Kay, 1980) and other tabular parsing algorithms (Aho and Ullman, 1972; Graham et al., 1980) are usually presented in terms of certain (abstract) data structures that keep a record of the alternatives being explored by the parser. Looking at parsing procedures as proof procedures has the following advantages: (i) unification, gaps and unbounded dependencies are automatically handled; (ii) parsing strategies become possible that cannot be formulated in chart parsing.

The chart represents completed nonterminals (passive edges) and partially applied rules (active edges). From the standpoint of Earley Deduction, both represent derived clauses that have been proved in the course of an attempt to deduce a goal statement whose meaning is that a string belongs to the language generated by the grammar. An active edge corresponds to a nonunit clause, a passive edge to a unit clause. Nowhere in this definition is there mention of the "endpoints" of the edges. The endpoints correspond to certain literal arguments, and are of no concern to the (abstract) proof procedure. Endpoints are just a convenient way of indexing derived clauses in an implementation to reduce the number of nonproductive (nonunifying) attempts at applying the reduction rule.

We shall give now an example of the application of Earley Deduction to parsing, corresponding to the chart of Figure 1.

The CFG

S → NP VP
NP → Det N
Det → NP Gen
Det → Art
Det → *A*
VP → V NP

corresponds to the following definite-clause program:

$$s(S0,S) \Leftarrow np(S0,S1) \;\&\; vp(S1,S). \quad (20)$$
$$np(S0,S) \Leftarrow det(S0,S1) \;\&\; n(S1,S). \quad (21)$$
$$det(S0,S) \Leftarrow np(S0,S1) \;\&\; gen(S1,S). \quad (22)$$
$$det(S0,S) \Leftarrow art(S0,S). \quad (23)$$
$$det(S,S). \quad (24)$$
$$vp(S0,S) \Leftarrow v(S0,S1) \;\&\; np(S1,S). \quad (25)$$

The lexical categories of the sentence

$$_0\text{Agatha}_1\text{'s}_2\text{husband}_3\text{hit}_4\text{Ulrich}_5 \quad (26)$$

can be represented by the unit clauses

$$n(0,1). \quad (27)$$
$$gen(1,2). \quad (28)$$
$$n(2,3). \quad (29)$$
$$v(3,4). \quad (30)$$
$$n(4,5). \quad (31)$$

Thus, the task of determining whether (26) is a sentence can be represented by the goal statement

$$ans \Leftarrow s(0,5). \quad (32)$$

If the sentence is in the language, the unit clause *ans* will be derived in the course of an Earley Deduction proof. Such a proof could proceed as follows:

```
ans ⇐ s(0,5).                   goal statement              (33)
s(0,5) ⇐ np(0,S1) & vp(S1,5).
                                (33) instantiates (20)      (34)
np(0,S) ⇐ det(0,S1) & n(S1,S).
                                (34) instantiates (21)      (35)
det(0,S) ⇐ np(0,S1) & gen(S1,S).
                                (35) instantiates (22)      (36)
det(0,S) ⇐ art(0,S).            (35) instantiates (23)      (37)
np(0,S) ⇐ n(0,S).               (24) reduces (35)           (38)
np(0,1).                        (27) reduces (38)           (39)
s(0,5) ⇐ vp(1,5).               (39) reduces (34)           (40)
vp(1,5) ⇐ v(1,S1) & np(S1,5).
                                (40) instantiates (25)      (41)
det(0,S) ⇐ gen(1,S).            (39) reduces (36)           (42)
det(0,2)                        (28) reduces (42)           (43)
np(0,S) ⇐ n(2,S).               (43) reduces (35)           (44)
np(0,3).                        (29) reduces (44)           (45)
s(0,5) ⇐ vp(3,5).               (45) reduces (34)           (46)
det(0,S) ⇐ gen(3,S).            (45) reduces (36)           (47)
vp(3,5) ⇐ v(3,S1) & np(S1,5).
                                (46) instantiates (25)      (48)
vp(3,5) ⇐ np(4,5).              (30) reduces (48)           (49)
np(4,5) ⇐ det(4,S1) & n(S1,5).
                                (49) instantiates (21)      (50)
det(4,S) ⇐ np(4,S1) & gen(S1,S).
                                (50) instantiates (22)      (51)
det(4,S) ⇐ art(4,S).            (50) instantiates (23)      (52)
np(4,S) ⇐ det(4,S1) & n(S1,S).
                                (51) instantiates (21)      (53)
np(4,5) ⇐ n(4,5).               (24) reduces (50)           (54)
np(4,S) ⇐ n(4,S).               (24) reduces (53)           (55)
np(4,5).                        (31) reduces (54)           (56)
vp(3,5).                        (56) reduces (49)           (57)
det(4,S) ⇐ gen(5,S).            (56) reduces (51)           (58)
```

s(0,5). (67) reduces (46) (59)
ans. (59) reduces (33) (60)

Note how subsumption is used to curtail the left recursion of rules (21) and (22), by stopping extraneous instantiation steps from the derived clauses (35) and (36). As we have seen in the example of the previous section, this mechanism is a general one, capable of handling complex grammar symbols within certain constraints that will be discussed later.

The Earley Deduction derivation given above corresponds directly to the chart in Figure 1.

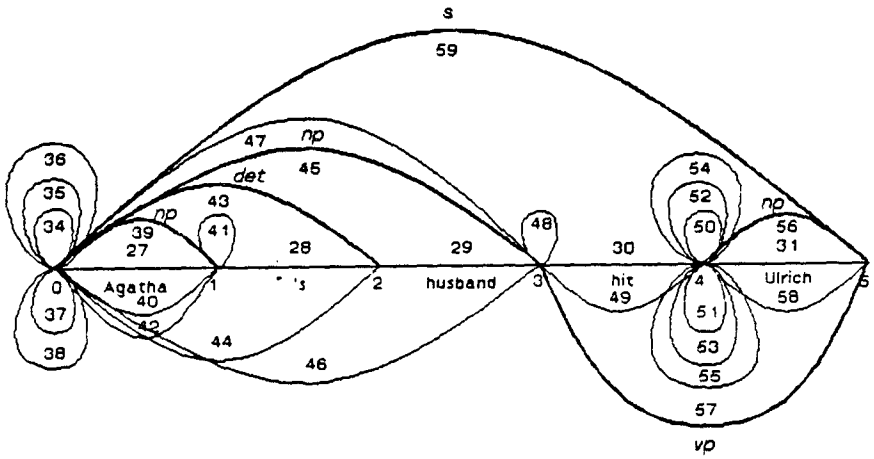


Figure 1: Chart vs. Earley Deduction Proof

Each arc in the chart is labeled with the number of a clause in the proof. In each clause that corresponds to a chart arc, two literal arguments correspond to the two endpoints of the arc. These arguments have been underlined in the derivation. Notice how the endpoint arguments are the two string arguments in the head for unit clauses (passive edges) but, in the case of nonunit clauses (passive edges), are the first string argument in the head and the first in the leftmost literal in the body.

As we noted before, our view of parsing as deduction makes it possible to derive general parsing mechanisms for augmented phrase-structure grammars with gaps and unbounded dependencies. It is difficult (especially in the case of pure bottom-up parsing strategies) to augment chart parsers to handle gaps and dependencies (Thompson, 1981). However, if gaps and dependencies are specified by extra predicate arguments in the clauses that correspond to the rules, the general proof procedures will handle those phenomena without further change. This is the technique used in DCGs and is the basis of the specialized extraposition grammar formalism (Pereira, 1981).

The increased generality of our approach in the area of parsing strategy stems from the fact that chart parsing strategies correspond to specialized proof procedures for definite clauses with string arguments. In other words, the origin of these proof procedures means that string arguments are treated differently from other arguments, as they correspond to the chart nodes.

In general, chart parsing cannot support strategies that would create active edges by reducing the symbols in the right-hand side of a rule in any arbitrary order. This is because an active edge must correspond to a contiguous sequence of analyzed symbols. Definite clause proof procedures do not have this limitation. For example, it is very simple to define a strategy, "head word parsing"[2] (McCord, 1980), which would use the reduction rule to infer

np(S0,S) ⇐ det(S0,2) & rel(3,S).

from the clauses

np(S0,S) ⇐ det(S0,S1) & n(S1,S2) & rel(S2,S).
[NP → Det N Rel]
n(2,3).
[There is an N between points 2 and 3 in the input]

This example shows that the class of parsing strategies allowed in the deductive approach is broader than what is possible in the chart parsing approach. It remains to be shown which of those strategies will have practical importance as well.

6. Implementing Earley Deduction

To implement Earley Deduction with an efficiency comparable, say, to Prolog, presents some challenging problems. The main issues are

- How to represent the derived clauses, especially the substitutions involved.
- How to avoid the very heavy computational cost of subsumption.
- How to recognize when derived clauses are no longer

[2]This particular strategy could be implemented in a chart parser, by changing the rules for combining edges but the generality demonstrated here would be lost.

needed and space can be recovered.

There are two basic methods for representing derived clauses in resolution systems: the more direct **copying** method, in which substitutions are applied explicitly; the **structure-sharing** method of Boyer and Moore, which avoids copying by representing derived clauses implicitly with the aid of variable binding environments. A promising strategy for Earley Deduction might be to use copying for derived *unit* clauses, structure sharing for other derived clauses. When copying, care should be taken not to copy variable-free subterms, but to copy just pointers to those subterms instead.

It is very costly to implement subsumption in its full generality. To keep the cost within reasonable bounds, it will be essential to **index** the derived clauses on at least the predicate symbols they contain — and probably also on symbols in certain key argument positions. A simplification of full subsumption checking that would appear adequate to block most redundant steps is to keep track of selected literals that have been used exhaustively to generate instantiation steps. If another selected literal is an instance of one that has been exhaustively explored, there is no need to consider using it as a candidate for instantiation steps. Subsumption would then be only applied to derived *unit* clauses.

A major efficiency problem with Earley deduction is that it is difficult to recognize situations in which derived clauses are no longer needed and space can be reclaimed. There is a marked contrast with purely top-down proof procedures, such as Prolog, to which highly effective space recovery techniques can be applied relatively easily. The Earley algorithm pursues all possible parses in parallel, indexed by string position. In principle, this permits space to be recovered, as parsing progresses, by deleting information relating to earlier string positions. It may be possible to generalize this technique to Earley Deduction, by recognizing, either automatically or manually, certain special properties of the input clauses.

7. Decidability and Computational Complexity

It is not at all obvious that grammar formalisms based on unification can be parsed within reasonable bounds of time and space. In fact, unrestricted DCGs have Turing machine power, and LFG, although decidable, seems capable of encoding exponentially hard problems. However, we need not give up our interest in the complexity analysis of unification-based parsing. Whether for interesting subclasses of grammars or specific grammars of interest, it is still important to determine how efficient parsing can be. A basic step in that direction is to estimate the cost added by unification to the operation of combining (reducing or expanding) a nonterminal in a derivation with a nonterminal in a grammar rule.

Because definite clauses are only semidecidable, general proof procedures may not terminate for some sets of definite clauses. However, the specialized proof procedures we have derived from parsing algorithms are **stable**: if a set of definite clauses G is the translation of a context-free grammar, the procedure will always terminate (in success or failure) when to proving any start goal for G. More interesting in this context is the notion of **strong stability**, which depends on the following notion of **offline parsability**. A DCG is offline-parsable if its context-free skeleton is not infinitely ambiguous. Using different terminology, Bresnan and Kaplan (Bresnan and Kaplan, 1982) have shown that the parsing problem for LFG is decidable because LFGs are offline parsable. This result can be adapted easily to DCGs, showing that the parsing problem for offline-parsable DCGs is decidable. Strong stability can now be defined: a parsing algorithm is strongly stable if it always terminates for offline-parsable grammars. For example, a direct DCG version of the Earley parsing algorithm is stable but not strongly so.

In the following complexity arguments, we restrict ourselves to offline-parsable grammars. This is a reasonable restriction for two reasons: (i) since general DCGs have Turing machine power, there is no useful notion of computational complexity for the parser on its own; (ii) there are good reasons to believe that linguistically relevant grammars must be offline-parsable (Bresnan and Kaplan, 1982).

In estimating the added complexity of doing online unification, we start from the fact that the length of any derivation of a terminal string in a finitely ambiguous context-free grammar is linearly bounded by the length of the terminal string. The proof of this fact is omitted for lack of space, but can be found elsewhere (Pereira and Warren, 1983).

General definite-clause proof procedures need to access the values of variables (**bindings**) in derived clauses. The structure-sharing method of representation makes the time to access a variable binding at worst linear in the length of the derivation. Furthermore, the number of variables to be looked up in a derivation step is at worst linear in the size of the derivation. Finally, the time (and space) to finish a derivation step, once all the relevant bindings are known, does not depend on the size of the derivation. Therefore, using this method for parsing offline-parsable grammars makes the time complexity of each step at worst $o(n^2)$ in the length of the input.

Some simplifications are possible that improve that time bound. First, it is possible to use a **value array** representation of bindings (Boyer and Moore, 1972) while exploring any given derivation path, reducing to a constant the variable lookup time at the cost of having to save and restore $o(n)$ variable bindings from the value array each time the parsing procedure moves to explore a different derivation path. Secondly, the unification cost can be made independent of the derivation length, if we forgo the **occurs check** that prevents a variable from being bound to a term containing it. Finally, the combination of structure sharing and copying suggested in the last section eliminates the overhead of switching to a different derivation path in the value array method at the cost of a uniform $o(\log n)$ time to look up or create a variable binding in a balanced binary tree.

When adding a new edge to the chart, a chart parser

must verify that no edge with the same label between the same nodes is already present. In general DCG parsing (and therefore in online parsing with any unification-based formalism), we cannot check for the "same label" (same lemma), because lemmas in general will contain variables. We must instead check for subsumption of the new lemma by some old lemma. The obvious subsumption checking mechanism has an $o(n^3)$ worst case cost, but the improved binding representations described above, together with the other special techniques mentioned in the previous section, can be used to reduce this cost in practice.

We do not yet have a full complexity comparison between online and offline parsing, but it is easy to envisage situations in which the number of edges created by an online algorithm is much smaller than that for the corresponding offline algorithm, whereas the cost of applying the unification constraints is the same for both algorithms.

8. Conclusion

We have outlined an approach to the problems of parsing unification-based grammar formalisms that builds on the relationship between parsing and definite-clause deduction.

Several theoretical and practical problems remain. Among these are the question of recognizing derived clauses that are no longer useful in Earley-style parsing, the design of restricted formalisms with a polynomial bound on the number of distinct derived clauses, and independent characterizations of the classes of offline-parsable grammars and languages.

Acknowledgments

We would like to thank Barbara Grosz and Stan Rosenschein for their comments on earlier versions of this paper.

References

A. V. Aho and J. D. Ullman, *The Theory of Parsing, Translation and Compiling* (Prentice-Hall, Englewood Cliffs, New Jersey, 1972).

R. S. Boyer and J S. Moore, "The Sharing of Structure in Theorem-Proving Programs," in *Machine Intelligence 7*, B. Meltzer and D. Michie, eds., pp. 101-116 (John Wiley & Sons, New York, New York, 1972).

J. Bresnan and R. Kaplan, "Lexical-Functional Grammar: A Formal System for Grammatical Representation," in *The Mental Representation of Grammatical Relations*, J. Bresnan, ed., pp. 173-281 (MIT Press, Cambridge, Massachusetts, 1982).

A. Colmerauer, "Metamorphosis Grammars," in *Natural Language Communication with Computers*, L. Bolc, ed. (Springer-Verlag, Berlin, 1978). First appeared as 'Les Grammaires de Metamorphose', Groupe d'Intelligence Artificielle, Université de Marseille II, November 1975.

J. Earley, "An Efficient Context-Free Parsing Algorithm," *Communications of the ACM*, Vol. 13, No. 2, pp. 94-102 (February 1970).

G. Gazdar and G. Pullum, *Generalized Phrase Structure Grammar: A Theoretical Synopsis* (Indiana University Linguistics Club, Bloomington, Indiana, 1982).

S. L. Graham, M. A. Harrison and W. L. Ruzzo, "An Improved Context-Free Recognizer," *ACM Transactions on Programming Languages and Systems*, Vol. 2, No. 3, pp. 415-462 (July 1980).

M. Kay, "Functional Grammar," *Proc. of the Fifth Annual Meeting of the Berkeley Linguistic Society*, pp. 142-158, Berkeley Linguistic Society, Berkeley, California (February 17-19 1979).

M. Kay, "Algorithm Schemata and Data Structures in Syntactic Processing," Technical Report , XEROX Palo Alto Research Center , Palo Alto, California (1980). A version will appear in the proceedings of the Nobel Symposium on Text Processing, Gothenburg, 1980.

R. A. Kowalski, *Logic for Problem Solving* (North Holland, New York, New York, 1980).

M. C. McCord, "Slot Grammars," *American Journal of Computational Linguistics*, Vol. 6, No. 1, pp. 255-286 (January-March 1980).

F. C. N. Pereira, "Extraposition Grammars," *American Journal of Computational Linguistics*, Vol. 7, No. 4, pp. 243-256 (October-December 1981).

F. C. N. Pereira, *Logic for Natural Language Analysis*, Ph.D. thesis, University of Edinburgh, Scotland, 1982.

F. C. N. Pereira and D. H. D. Warren, "Definite Clause Grammars for Language Analysis - a Survey of the Formalism and a Comparison with Augmented Transition Networks," *Artificial Intelligence*, Vol. 13, pp. 231-278 (1980).

F. C. N. Pereira and D. H. D. Warren, "Parsing as Deduction," Forthcoming technical note , Artificial Intelligence Center, SRI International , Menlo Park, California (1983).

J. A. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," *Journal of the ACM*, Vol. 12, pp. 23-44 (January 1965).

P. Roussel, "Prolog : Manuel de Référence et Utilisation," Technical Report , Groupe d'Intelligence Artificielle, Université d'Aix-Marseille II , Marseille, France (1975).

S. Shieber, Personal communication, 1983.

H. Thompson, "Chart Parsing and Rule Schemata in GPSG," *Proc. of the 19th Annual Meeting of the Association for Computational Linguistics*, pp. 167-172, Association for Computational Linguistics, Stanford University, Stanford, California (June 29-July 1 1981).

M. H. van Emden and R. A. Kowalski, "The Semantics of Predicate Logic as a Programming Language," *Journal of the ACM*, Vol. 23, No. 4, pp. 733-742 (October 1976).

D. H. D. Warren, Earley Deduction. Unpublished note, 1975.

D. H. D. Warren and F. C. N. Pereira, An Efficient Easily Adaptable System for Interpreting Natural Language Queries. To appear in the American Journal of Computational Linguistics., 1983.