Abstract Left-corner Parsing for Unification Grammars

Noriko Tomuro and Steven L. Lytinen
DePaul University
School of Computer Science, Telecommunications and Information Systems
243 S. Wabash Ave.
Chicago, IL 60604
{lytinen, tomuro}@cs.depaul.edu

Abstract

We present a formal description of an abstract left-corner parsing algorithm for unification grammars, using the logic developed in (Shieber, 1992). Our algorithm precompiles grammar rules into dag structures with a special feature named lc, and uses those structures throughout parsing. We compare our algorithm with Shieber's abstract parsing algorithm by showing a mapping of our algorithm's logical operations onto Shieber's, and discuss the efficiency benefits of our algorithm as compared to Shieber's and to other left-corner algorithms.

1 Introduction

Unification grammar is a term often used to describe a family of feature-based grammar formalisms, including GPSG (Gazdar et al., 1985), PATR-II (Shieber, 1986), DCG (Pereira and Warren, 1980), and HPSG (Pollard and Sag, 1994). In an effort to formalize the common elements of unification-style grammars, (Shieber, 1992) developed a logic for describing them, and used this logic to define an abstract parsing algorithm. His algorithm uses the same set of operations as Earley’s context-free parsing algorithm (Earley, 1970) but modified for unification grammars.

In this paper, we present a formal description of an abstract left-corner (LC) parsing algorithm for unification grammars. The algorithm is a variation of left-corner parsing for unification grammars, and as such is an optimization of Shieber's algorithm. The LC algorithm has a unique characteristic: every model produced by the algorithm contains a special feature named lc. This feature explicitly represents the relation between a parsing prediction and its partially instantiated left-corner constituent. The lc features are inserted in the models by grammar precompilation, and parsing operations build parse models by manipulating this feature. By utilizing such structures, the algorithm is fundamentally more efficient than Shieber's (although the theoretical complexity remains undecidable), and performs fewer unification operations than other left-corner algorithms (e.g., (Alshawi, 1992; van Noord, 1997)).

Shieber's abstract algorithm covers a range of possible parsing techniques for unification grammars, including pure top-down, pure bottom-up, and mixed-direction parsing (but not left-corner parsing). The abstract nature of the algorithm is primarily due to the inclusion of an unspecified parameter function \( \rho \) applied in the prediction-generation portion of the algorithm. The \( \rho \) function determines how much, and what kind, of information is used to generate parsing predictions. By adjusting this parameter, various instantiations of his algorithm range from expectation-driven top-down parsing using all available information for predictions, to data-driven bottom-up parsing using no predictive information, and anywhere in between.

Our LC algorithm is also defined in Shieber's logic, with the same function \( \rho \). Thus, our algorithm is abstract and, in a similar manner as Shieber's, covers a range of possible left-corner parsing techniques for unification grammars, although it is not a generalization of standard left-corner algorithm. Note that our algorithm presented here is a formal algorithm; it does not include any practical implementation techniques developed for unification grammars, such as ambiguity packing (Alshawi, 1992; van Noord, 1997; Lavie and Rose, 2000), grammar pre-analysis (Moore and Dowding, 1991; Lavie and Rose, 2000) and transformation (Maxwell and Kaplan, 1994; Carroll, 1993), as a part of the algorithm. Incorporating such techniques, along with the choice of \( \rho \), is left up to the implementation. In this sense, our work should be compared with other efforts such as (Carpenter, 1992; Sikkel, 1997) which formalized unification-based parsing schemes.
Below, we first summarize Shieber’s logic and parsing algorithm, then present the LC algorithm. We compare the two algorithms by showing an example, and describe the mapping from LC’s parsing operations onto Shieber’s as an informal proof of correctness of LC. Finally, we discuss the efficiency of LC as compared to Shieber’s algorithm as well as other left-corner unification-based algorithms.

2 Shieber’s logic

(Shieber, 1992) defines a unification grammar as a 3-tuple \( \langle \Sigma, P, p_0 \rangle \), where \( \Sigma \) is the vocabulary of the grammar, \( P \) is the set of productions, and \( p_0 \in P \) is the start production. \( \Sigma \) contains \( L \), a set of labels (feature names); \( C \), a set of constants (feature values); and \( W \), a set of terminals. There are two kinds of productions in \( P \): phrasal and lexical. A phrasal production is a 2-tuple \( (a, \Phi) \), where \( a \) is the arity of the rule (the number of right-hand side (RHS) constituents), and \( \Phi \) is a logical formula. Typically, \( \Phi \) is a conjunction of equations of the form \( p_1 = p_2 \) or \( p_1 = c \), where \( p_1, p_2 \in L^* \) are paths,\(^1\) and \( c \in C \). In an equation, any path which begins with an integer \( i \) (\( 1 \leq i \leq a \)) represents the \( i^{th} \) RHS constituent of the rule.\(^2\)

A lexical production is a 2-tuple \( \langle w, \Phi \rangle \), where \( w \in W \) and \( \Phi \) is the same as above, except that there are no RHS constituents. Figure 1 shows a simple example grammar which can be used to parse the sentence “John likes Mary.”

Based on the logic described above, Shieber defines his abstract parsing algorithm as a set of four logical deduction rules. Each rule derives a new item, from previous items and/or productions in the grammar. An item is a 5-tuple \( \langle i, j, p, M, d \rangle \), where \( i \) and \( j \) are indices into the sentence and specify which words in the sentence have been used to construct the item; \( p \) is the production used to construct the item; \( M \) is a model,\(^3\) and \( d \) is the position of the “dot”; i.e., the number of RHS constituents in \( p \) completed so far.

The logical rules of the abstract algorithm are shown in Figure 2. The \textit{Initial item} rule produces the first item, and is constructed from the start production \( p_0 \). It spans none of the input (\( i \) and \( j \) are both 0), and its model is the \textit{minimal model} (mm) of \( p_0 \).

The \textit{Prediction} rule is essentially the top-down rewriting of the expectation (a subconstituent just after the dot) in a prior item. In this rule, the \textit{extraction} operator, denoted by \( \langle / \rangle \), retrieves the feature structure found at the end of a particular path; so \( M/(d+1) \) retrieves the \( d+1^{st} \) submodel in \( M \) (i.e., an expectation). The unification operation is denoted by \( \uplus \), and combines the features of two models. The function \( \rho \), which is the unspecified parameter in the abstract algorithm, may filter out some (pre-defined) features (or it may leave all features in the prediction). Here, it is applied to the expectation, by which it effectively controls the top-down predictive power of the algorithm and provides flexibility to the instantiated algorithms. Then the expectation is unified with a production \( \Phi' \) which can consistently rewrite it. By this operation, features retained by \( \rho \) will be propagated down in the production.

The remaining two rules advance the dot in a prior item, by unifying the subconstituent to the right of the dot with either a lexical item from the input string (the \textit{Scanning rule}) or some other completed higher-level item (the \textit{Completion rule})...
INITIAL ITEM:
\[\{0, 0, p_0, mm(\Phi_0), 0\}\]

PREDICTION:
\[
\langle i, j, p = \langle a, \Phi \rangle, M, d \rangle
\]
\[
\langle j, j', p', \rho(M/(d+1)) \cup mm(\Phi'), 0 \rangle
\]
where \(d < a\) and \(p' = \langle a', \Phi' \rangle \in P\)

SCANNING:
\[
\langle i, j, p = \langle a, \Phi \rangle, M, d \rangle
\]
\[
\langle i, j + 1, p, M \cup (mm(\Phi') \setminus \{d+1\}), d + 1 \rangle
\]
where \(d < a\) and \(\langle w_{j+1}, \Phi' \rangle \in P\)

COMPLETION:
\[
\langle i, j, p = \langle a, \Phi \rangle, M, d \rangle
\]
\[
\langle j, k, p, M \cup (M' \setminus \{d+1\}), d + 1 \rangle
\]
where \(d < a\)

Figure 2: Shieber’s parsing operations

Both rules utilize the \textit{embedding} operator (signified by \(\setminus\)), which places a model \(M\) under a path \(p\) \((M/p)\).

3 Abstract Left-corner Parsing Algorithm

Left-corner parsing is a mixed-direction parsing technique. Constituents are constructed from the input in a bottom-up fashion, but top-down expectations are used in conjunction with reachability information (i.e., reflexive, transitive closure of the left-corner derivations) so as to only allow the bottom-up construction of constituents which are consistent with their left context. Reachability information is usually precompiled from the grammar, and stored in a table.

Normally, a reachability relation is defined as a 2-tuple which relates an expectation and a left-corner constituent. For instance, given the following two rules (using the context-free backbone of the example grammar shown in Figure 1):

**R1:** \(\text{VP} \rightarrow \text{VG NP}\)

**R2:** \(\text{VG} \rightarrow \text{V}\)

reachability relations exist between the ordered pairs \([\text{VP}, \text{VG}], [\text{VP}, \text{V}]\) and \([\text{VG}, \text{V}]\).

In the LC algorithm, the reachability relation is represented differently: between an expectation and a grammar rule, rather than just its left-corner constituent. Thus, the reachability relation in LC will become \([\text{VP}, \text{R1}],[\text{VP}, \text{R2}],[\text{VG}, \text{R2}]\) respectively.\(^4\) Then, reachability is represented not as a 2-tuple but in a single structure using a feature called \text{lc}, in which the expectation is placed at the root node, and the rule is connected under the \text{lc} arc. Thus, the \text{lc} arc represents an unspecified number of levels of left-corner derivations that are possible from a high-level expectation to a rule, and functions as a compressed link between the two. By this scheme, reachability is represented in the same way as other linguistic information. Figure 6, the right-most dag, shows the reachability entry for the relation \([\text{VP}, \text{R2}]\). The LC algorithm uses dags with the \text{lc} feature both in reachability entries and in models produced during parsing. Note that a given \text{lc} arc is eventually deleted during parsing, as will be described in section 3.2.

To define our algorithm, in addition to Shieber’s extraction, embedding, and unification operators, we use three more operators. The first one is the \textit{restriction} operator denoted \(\Rightarrow\). This operator is defined in Shieber’s logic, as follows: for a model \(M\) and a set of labels/arcs \(F\), \(M \upharpoonright F\) retains all non-empty paths in \(M\) which start with an element in \(F\) and removes all others. The second one, which is also defined in Shieber’s logic, is the \textit{domain} operator denoted \(\text{dom}\). When this operator is applied to a model \(M\) as \(\text{dom}(M)\), it returns a set of all arcs that exist at the root node of \(M\). The third one is the \textit{path replacement} operator denoted \(\Rightarrow\). We newly define this operator for LC, and it is defined as follows: for a model \(M\) and paths \(p1\) and \(p2\), \(M[p1 \Rightarrow p2]\) moves the submodel under \(p1\) in \(M\) to path \(p2\).\(^5\) In the LC algorithm, path replacement is critically applied to stretch/shrink \text{lc} arcs in the parse models. Figure 3 shows an example of this operator in which the path \(\langle f, g \rangle\) is replaced by \(\langle h \rangle\).

The LC algorithm is defined by two sets of logical deduction rules, where the first set of rules produce reachability entries during grammar precompilation, and the second set derive items during parsing. We describe each rule in detail below.

\(^4\)In reality, expectations are not just grammatical categories such as \text{VP}, but may themselves contain many features; see Figure 6.

\(^5\)Although not shown in this paper, the \(\Rightarrow\) operator is composed of Shieber’s four operators \(\cup, /, \setminus, \Rightarrow\), so it can be expressed in his logic.
3.1 Reachability Rules

In the LC algorithm, a reachability entry is produced for every consistent left-corner derivation path. It is constructed from two rules: RN1 and RN2 (shown in Figure 4). For a sequence of productions \( s_n = \langle p_1, \ldots, p_n \rangle \) which constitute a derivation path, one or more application of RN1/RN2 rules essentially creates a model in which the LHS constituent of \( p_1 \) (as an expectation) is placed at the root and \( p_n \) is connected under the \( \text{lc} \) arc. For the purposes of the logic, reachability entries are 3-tuples of the form \( \langle s_n, p_n, M \rangle \), where \( s_n = \langle p_1, \ldots, p_n \rangle \) is a sequence of the phrasal productions giving rise to the entry, \( p_n \) is the last production in \( s_n \) and \( M \) is the resulting model.

The RN1 rule creates a model which represents a reflexive reachability relation. It is applied only at the beginning of a derivation sequence, and it directly converts a phrasal production to a reachability entry by prefixing all paths that start with a numbered arc (i.e., the RHS constituents) with a \( \text{lc} \) arc and unifying the production again under the \( \text{lc} \) arc. Figure 5 shows the result of applying RN1 to the grammar rule \( R_1 \) (shown previously in Figure 1). A model created by the RN1 rule, which we call an \( \text{RN1} \) model, has a particular property where constituents at the root and under the \( \text{lc} \) arc have the same category feature. In the figure, the resulting RN1 model can be written using the context-free backbone as \( \text{VP} \xrightarrow{\text{R}n} \text{VP} \rightarrow \text{VG} \text{NP} \). Since this model represents a reflexive relation, the two \( \text{VP} \)'s intentionally denote the same constituent, that is, the LHS constituent. This form can represent left recursive rules uniformly with other rules in the reachability entries, and effectively allows any number of applications of such rules during parsing (in particular, in the bottom-up processing).

The RN2 rule creates entries which represent a transitive reachability relation. We call such a model an \( \text{RN2} \) model or a \( \text{non-RN1} \) model. RN2 is applied from the second rule on in a derivation sequence. It first compresses the left-corner path \( \langle \text{lc}1 \rangle \) into \( \langle \text{lc} \rangle \) by using the path replacement operator. Then it unifies this intermediate model with a rule that can rewrite the LHS constituent. Note that \( \rho \) in our algorithm is analogous to Shieber’s.

As the reachability relation is extended further by RN2, some features which influence the left-corner constituents are propagated down. Figure 6 shows the result of applying the RN2 rule to the model \( M \) in Figure 5 and the production \( R_2 \) (see

\[\text{RN1}:\]
\[
\langle \langle \rho, p, \text{mm}(\Phi) \rangle \rangle_n \Rightarrow \langle \langle \text{lc} n \rangle \rangle_n^{a-1} \cup \text{mm}(\Phi) \backslash \langle \text{lc} \rangle
\]
where \( p = \langle a, \Phi \rangle \in P \)

\[\text{RN2}:\]
\[
\langle \langle s_n = \langle p_1, \ldots, p_n \rangle, p_n, M \rangle \rangle_n^{a-1} \cup \text{mm}(\Phi) \backslash \langle \text{lc} \rangle
\]
where \( p' = \langle a, \Phi \rangle \in P \)

**INITIAL ITEM:**
\[
\langle 0, 0, p_0, RM, 0 \rangle
\]

**SCANNING 1:**
\[
\langle i, j, p = \langle a, \Phi \rangle, M, d \rangle
\]
\[
\langle i, j + 1, p, M \cup \text{mm}(\Phi) \backslash \langle \text{lc} d + 1 \rangle, d + 1 \rangle
\]
where \( d < a \) and \( \langle w_{j+1}, \Phi' \rangle \in P \)

**SCANNING 2:**
\[
\langle i, j, p = \langle a, \Phi \rangle, M, d \rangle, s_n, p', RM \rangle
\]
\[
\langle j, j + 1, p', \rho(M) \backslash \langle \text{lc} d + 1 \rangle \cup \text{mm}(\Phi) \backslash \langle \text{lc} \rangle \cup RM, 1 \rangle
\]
where \( d < a \) and \( \langle w_{j+1}, \Phi' \rangle \in P \)

**COMPLETION:**
\[
\langle i, j, p = \langle a, \Phi \rangle, M, d \rangle, s_n, p', RM \rangle
\]
\[
\langle k, j, p' = \langle a', \Phi' \rangle, M', a' \rangle
\]
\[
\langle k, k, p, M \cup M' \backslash \langle \text{lc} \rangle \cup M' \backslash \langle F \rangle \backslash \langle \text{lc} d + 1 \rangle, d + 1 \rangle
\]
where \( F = \text{dom}(M') - \langle \text{lc} \rangle, d < a \)

**CONTINUATION:**
\[
\langle i, j, p = \langle a, \Phi \rangle, M, a \rangle, s_n, p', RM \rangle
\]
\[
\langle k, j, p' = \langle a', \Phi' \rangle, M', a' \rangle
\]
\[
\langle k, k, p, M \cup RM, 1 \rangle
\]
where \( d < a \)

Figure 4: LC parsing operations

Figure 5: Application of RN1 rule to VP rule (R1)

Figure 6: Application of RN2 rule to VP rule (R2)
Figure 6: Application of RN2 rule to $M$ and VG rule (R2)

Figure 1. A feature $\langle lc$ 1 head type $\rangle \equiv$ trans in the left-corner constituent VG in $M$ is first transformed to $\langle lc$ head type $\rangle \equiv$ trans by path replacement, and pushed into $M_2$'s new left-corner constituent V through a path equation $\langle lc$ 1 head $\rangle \equiv \langle lc$ head $\rangle$ in R2.

3.2 Parsing Operations

Parsing is defined by five operations in LC: Initial Item, Scanning 1, Scanning 2, Completion and Continuation (shown in Figure 4). As with Shieber’s algorithm, parsing begins by generation of an initial item. Assume that $R_0$ in Figure 1 has been designated as the grammar’s start production. Then at the beginning of parsing, the Initial Item rule inserts an item which has an RN1 model created from the start production $\left(\langle R_0 \rangle, R_0, RM \right)$ as the initial expectation. Notice the dot position in the resulting item is 0, implying that no RHS constituent is filled yet.

In LC, there are two scanning operations (which match the next input token (terminal) against existing expectations). Two operations are necessary because an expectation may directly predict the next terminal, or it may be satisfied via a reachability entry.

The Scanning 1 rule is analogous to Scanning in Shieber’s algorithm. It unifies an expectation in an item model $\left( M/\{lc\ d+1\} \right)$ with the input word $\left( mm(\Phi') \right)$ where $\Phi'$ is a lexical production), and advances the dot.

The Scanning 2 rule matches an expectation for a terminal through a reachability entry. It extracts the expectation $\left( p(M/\{lc\ d+1\}) \right)$, combines it with input word $\left( mm(\Phi') \right)$, and then unifies with a reachability model $\left( RM \right)$ that connects the two constituents. Figure 7 graphically depicts the actions of the two scanning rules.7

When the dot position reaches the arity of its phrasal production, the item is complete. Depending on whether the model is an RN1 model or a non-RN1 model, two different operations are necessary.

The Completion rule is essentially equivalent to Completion in Shieber’s algorithm, and applies to a completed RN1 model. It first merges the the expectation at the root and the LHS constituent under lc arc (by $M'/\{lc\} \cup M' \setminus dom(M') - \{lc\}$). It then removes the arc from the model. Then the rule unifies the merged LHS constituent with the expectation in the previous item model $\left( M/\{lc\ d+1\} \right)$, and advances the dot. The actions of this rule are graphically depicted in Figure 8.

In the Continuation operation, the rule sequence compressed in the lc arc in the base reachability entry is expanded. Continuation first stretches the $\langle lc \rangle$ path in a completed model $\left( M \right)$ to $\langle lc 1 \rangle$ by the path replacement operator, and unifies the resulting model with another reachability model $\left( RM \right)$, which is one level above in the lc that are connected to the rules under lc arc. However, those paths are omitted in the pictures for simplicity.

Note that a non-RN1 model could successfully allow this step as well, if the licensing rule is left-recursive. However for a non-RN1 model, the Continuation operation (discussed next) instead of Completion should be applied to produce the correct result.

---

7In Figures 7-8, shaded nodes and trees are directly affected by unification. Also note that in these figures, the root nodes of the models $M$, $M'$ and reachability model $RM$ may have some other paths besides...
left-corner derivation. This way, the compressed rule sequence in the old complete model is unfolded in the reverse (bottom-up) order. Figure 8 graphically depicts the actions of this rule.

4 Shieber’s vs. LC

To show the LC algorithm is an optimization of Shieber’s algorithm, we map LC’s logical operations onto Shieber’s and show an example. This also serves as an informal proof of correctness of the LC algorithm, since Shieber (1992) proved his algorithm to be correct.9

The mapping from the operations in LC to Shieber’s is shown in Table 1. In Shieber’s algorithm, top-down processing by n Prediction operations (where \( n > 0 \)) generates an item whose dot position is 0, and this item triggers bottom-up processing by Scanning or Completion, which advances the dot position to 1. Subsequent dot advancing is done by bottom-up processing only. Therefore, Scanning and Completion can be broken into two cases: one preceded by n Prediction operations (which creates items whose dot position is 1), and the other not preceded by Prediction (which creates items whose dot position is \( d \) where \( d > 1 \)). The first operation sequence is equivalent to Scanning 2 and Continuation in LC, and the second sequence is equivalent to Scanning 1 and Completion in LC. Since those four sequences do not overlap, they cover exhaustively all possible operation sequences that take place in Shieber’s algorithm.

We illustrate the equivalency between the LC and Shieber’s algorithms by showing a parse of ‘John likes Mary’ using the grammar in Figure 1. In this example, we focus on the sync point between the two algorithms mentioned above. Figure 9 shows a diagram of operations applied and items created after each parsing operation in Shieber’s algorithm. Only the context-free backbone of each item’s model is shown. Notice after ‘John’ is scanned, two Prediction operations (3) and (4) are necessary to link the high-level VP expectation to the input word ‘likes’. Then after scanning ‘likes’ (5) scanning, Completion is applied to fill the VG and advance the dot (6). Figure 10 shows a similar diagram for the LC algorithm. At the top of the figure, reachability entries are listed. In LC, parsing starts with Initial Item using a reachability entry t0. ‘John’ is scanned in the same way as Shieber’s (using the Scanning 1 rule – (2) scanning 1). Then, the word ‘likes’ is immediately scanned by the Scanning 2 operation using the reachability entry t2: VP \( \rightarrow \) VG \( \rightarrow \) V (3 scanning 2). After that, Continuation is applied and creates an item whose left-corner VG is filled (4 continuation). Here, the reachability entry t1 (VP \( \rightarrow \) VP \( \rightarrow \) VG NP) was used to realize the VG.

Figures 11 and 12 show the items created after the above operation sequences in each algorithm with detailed models.10 Notice the model in Shieber’s (5) scanning is the same

---

9A complete formal proof of correctness of the LC algorithm is beyond the scope of this paper. For those interested, it is given in (Tomuro, 1999).

10In Figure 11, nodes in the next expectation are shaded for clarity. Also, in Figures 11 and 12, the function \( \rho \) is the identity function.
as the submodel under lc in the model in LC’s (3) scanning 2, and Shieber’s model in (6) completion is the same as the submodel under lc in the model in LC’s (4) continuation, indicating that the two Predictions followed by a Scanning in Shieber’s is equivalent to LC’s Scanning 2, and one Prediction followed by a Completion in Shieber’s is equivalent to LC’s Continuation.

5 LC vs. Standard Left-corner Parsing

Our LC algorithm is a variation of left-corner parsing for unification grammars, and as such is an optimization of Shieber’s algorithm. In left-corner parsing, a reachability relation is usually defined by a 2-tuple relating an expectation and a left-corner constituent which sanctions the expectation. For instance, the Core Language Engine (Alshawi, 1992) precompiles grammar rules into reachability relations into this (Prolog) form:

\[
\text{cat1:feature1=\text{X}, feature2=\text{Y}}, \\
\text{cat2:feature5=\text{X}, feature6=\text{Y}}
\]

where cat1 is the expectation and cat2 is the left-corner constituent. Shared variables X and Y communicate the feature values between the two. A similar representation is used in other parsers (Carroll, 1993; van Noord, 1997).

Our LC algorithm is more efficient than this standard left-corner parsing. The difference is that the standard algorithm must apply unification twice, once to unify an expectation with the reachability relation and another to unify the resulting model with a rule whose left-corner constituent is consistent. For example, the head-corner parser of (van Noord, 1997) uses this rule:

\[
\text{head\_corner} (\text{Small, Q0, Q, Cat, P0, P, E0}) \leftarrow \\
\text{head\_rule} (\text{Small, Mother, RevLeftDs, RightDs}), \\
\text{head\_link} (\text{Cat, P0, P, Mother, QL, QR}), \\
\text{parse\_left\_ds} (\text{RevLeftDs, QL, Q0, E0}), \\
\text{parse\_right\_ds} (\text{RightDs, Q0, QR, E}), \\
\text{head\_corner} (\text{Mother, QL, QR, Cat, P0, P, E0, E}).
\]

The head\_corner predicate is a subgoal produced during parsing. Grammar rules are represented by head\_rule, and reachability in-
formation by head_link/6: thus the Prolog variable Mother must be unified first with the LHS of grammar rule and then a second time with a reachability entry.

LC, on the other hand, only requires one unification, to combine the expectation dag with the reachability entry. This is because LC models carry the original expectation at the root (above the lc arc), thereby allowing the algorithm to access and manipulate the expected features at the appropriate time.

Note the efficiency discussion above is a theoretical one – empirical analysis still must be conducted to determine how significant the efficiency gain is. However, based on previous efforts toward efficient unification-based parsing (e.g., (Carroll, 1993; van Noord, 1997; Lavie and Rose, 2000)), empirical results seem to largely depend on various factors in the experiments, including the degree of ambiguity of the grammar used, the length of input sentences, the interleaving schedule of unification (whether unification is applied for every RHS constituent or delayed until the end of a rule when all RHS constituents are complete), and the amount of top-down information propagated (i.e., ρ). 11 Our theoretical analysis, on the other hand, is independent of any implementation factor. Therefore, we feel the theoretical analysis gives sufficient evidence for the efficiency of the LC algorithm over the standard left-corner algorithm.

Also note that, although the LC algorithm reduces the number of unifications, it does build models that are larger than those created by the standard left-corner algorithm. However, the overhead is the features above the lc arc, and the amount is rather small and constant – features are added below the lc arc as parsing proceeds.

6 Conclusions

The LC algorithm presented here extends Shieber’s abstract parsing algorithm to incorporate left-corner parsing techniques. By showing the mapping of LC operations on Shieber’s, we have demonstrated that the LC algorithm represents a substantial improvement in parsing efficiency over Earley-style parsing for unification grammars. In addition, the LC algorithm improves upon previous work in left-corner unification parsing by integrating reachability more fully into the parsing process. By utilizing the lc arc in both reachability entries and parse daggs, the application of reachability information is more efficient than in other left-corner parsers.

References


Shieber, S. 1986. An Introduction to Unification-Based Approaches to Grammar. CSLI.


11In an extreme case, if ρ does not pass down any information at all, LC as well as Shieber’s algorithm essentially become unification-based bottom-up chart parser (Kay, 1980).