Nonminimal derivations in unification-based parsing

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

Abstract

Many algorithms for parsing unification grammars are extensions of well-known context-free parsing techniques, such as chart parsing. An example is Shieber's abstract parsing algorithm (Shieber, 1992), which can be viewed as an extension of Earley's algorithm. In this paper, we discuss a difficulty that exists in these algorithms in general, and in Shieber's algorithm in particular. The difficulty is that these algorithms produce what we call "nonminimal derivations". Nonminimal derivations are parse trees which contain more information (i.e., features) than they absolutely have to. We show examples of nonminimal derivations produced by Shieber's algorithm, and enumerate the situations in which such derivations are produced. We then suggest several possible ways to modify to the algorithm to address this problem.

1 Introduction

Unification grammar is a term often used to describe a family of feature-based grammar formalisms. In this style of formalisms, grammar rules are represented by *feature structures*, or a set of feature-value pairs. Within the last fifteen years or so, various unification grammars have been developed including Generalized Phrase Structure Grammar (Gazdar, *et al*, 1985), PATR-II (Shieber, 1986), Definite Clause Grammar (Pereira and Warren, 1980), and Head-driven Phrase Structure Grammar (Pollard and Sag, 1994).

In an effort to formalize the common elements of unification grammars, Shieber (1992) developed a logic for describing these grammars, and specified an *abstract parsing algorithm* in the logic. The algorithm is quite general, in two ways: first, it is expressed as a set of deduction rules which, when applied, derive new *items*.[1] The rules merely specify what items are derivable; a specific instantiation of Shieber's algorithm may actually compute only a subset of these items.[2] Second, the algorithm contains an underspecified function which controls how much and what kind of information is used in parser *predictions*. Depending on how the function is specified, the algorithm performs as a pure top-down parser, a pure bottom-up parser, or something in between.

In this paper, we describe a difficulty with Shieber's algorithm, and propose at least a partial solution to the problem. For certain grammars, it is possible for Shieber's algorithm to compute derivations for some sentences which contain more features than they absolutely have to. We will call these *nonminimal derivations*. Although Shieber proved his algorithm to be sound and complete, it turns out that the proof relies on the way in which he defined the notion of a valid parse tree in the context of unification grammar: Shieber's definition allows for nonmiminal derivations to be considered valid. This definition does not correspond to our intuitions as to what a correct parse ought to be.

We propose a modification to Shieber's definition of a valid parse tree, so as to rule out nonminimal derivations as being considered valid. Then, we outline the circumstances under which Shieber's algorithm produces nonminimal derivations. Finally, we outline several possible approaches to solving this problem.

[1] Items play the same role as in Earley's algorithm (Earley, 1970), and are analagous to edges in context-free chart parsing.

[2] The logic does implicitly specify a partial ordering of items (i.e., which items must be derived before other items in the parse), so any instantiation of the algorithm is constrained by the partial ordering as to exactly what subset of items it can produce.

$$p_1 = \langle 2, \Phi_1 \rangle$$

$$\begin{array}{ll} \Phi_1: & \langle\ cat \rangle = S \\ & \langle 1\ cat \rangle = NP \\ & \langle 2\ cat \rangle = VP \\ & \langle head \rangle = \langle 2\ head \rangle \\ & \langle head\ subj \rangle = \langle 1\ head \rangle \\ & \langle 1\ head\ agr \rangle = \langle 2\ head\ agr \rangle \end{array}$$

$$p_2 = \langle John, \Phi_2 \rangle$$

$$\begin{array}{ll} \Phi_2: & \langle cat \rangle = NP \\ & \langle head\ type \rangle = proper \\ & \langle head\ agr\ pers \rangle = 3rd \\ & \langle head\ agr\ numb \rangle = singular \end{array}$$

Figure 1: Examples of productions

2 Unification Grammar and Parse Trees

Shieber's defines a unification grammar as a 3-tuple $\langle \Sigma, P, p_0 \rangle$, where Σ is the *vocabulary* of the grammar, P is the set of *productions*, and $p_0 \in P$ is the *start production.* Σ contains L, a set of *labels* (feature names); C, a set of *constants* (feature values), and W, a set of *lexical items.* There are two kinds of productions: *phrasal* and *lexical.* A phrasal production is a 2-tuple $\langle a, \Phi \rangle$, where a is the arity of the rule (a non-negative integer), and Φ is a logical formula using the elements of Σ. A lexical production is a 2-tuple $\langle w, \Phi \rangle$, where $w \in W$. Typically, the formulas Φ in phrasal and lexical productions are conjunctions of two kinds of atomic formulas:

1. $p_1 \doteq p_2$
2. $p_1 \doteq c$

where $p_1, p_2 \in L^*$ are *paths*, and $c \in C$. Formulas constrain the properties which a *model* of a rule may have; if two paths are equated, this means they must have the same value (i.e., they must unify). Figure 2 shows an example of a phrasal production, corresponding to the context-free rule $S \rightarrow NP\ VP$; and a lexical production, for the lexical item "John".

For the sake of brevity, we give a simplified version of Shieber's definition of a parse tree. The simplifications do not affect the points of this paper.

A parse tree is a *model*, which obeys the constraints of the appropriate phrasal and lexical productions in the grammar. Models can themselves be thought of a logical formulas, or they can be thought of as directed acyclic graphs (DAGs) which satisfy the constraints of the formulas. In any case, a satisfaction relation $\models$ relates a model to a formula, indicating that the model obeys the formula's constraints.

To define the notion of a valid parse tree, Shieber first defines the set of possible parse trees $\Pi = \bigcup_{i \geq 0} \Pi_i$ for a given grammar G, where each Π_i is defined as follows:

1. For all models τ, if $\tau \models \Phi$ and $\langle w, \Phi \rangle$ is a lexical production in G, then $\tau \in \Pi_0$.
2. For all models τ, if $\tau \models \Phi$ and $\langle a, \Phi \rangle$ is a phrasal production in G, and for $1 \leq d \leq a$, $\tau/\langle d \rangle \in \bigcup_{j \leq i} \Pi_j$, then $\tau \in \Pi_i$.

In (2), the *extraction* operator, denoted by /, can be thought of as retrieving the feature structure found at the end of a particular path; thus, $\tau/\langle 1 \rangle$ retrieves the first subconstituent in τ. Intuitively, then, Π_0 contains all models which satisfy any lexical production in the grammar, while Π_i contains all models which satisfy a phrasal production, and whose subconstituents all are in $\bigcup_{j \leq i} \Pi_j$.

To specify what constitutes a valid parse for a particular sentence, the next step is to define the *yield* of a parse tree. It is defined recursively as follows:

1. If $\tau \models \Phi$ for some lexical production $\langle w, \Phi \rangle$, then the yield of τ is w.
2. If $\tau \models \Phi$ for some phrasal production $\langle a, \Phi \rangle$, and if $\alpha_1, \ldots, \alpha_a$ are the yields of $\tau/\langle 1 \rangle, \ldots, \tau/\langle a \rangle$ respectively, then the yield of τ is $\alpha_1 \ldots \alpha_a$.

Finally, Shieber defines a valid parse tree τ for sentence $w_1 \ldots w_n$ as follows:

1. The yield of τ is $w_1 \ldots w_n$
2. $\tau \models p_0$
3. $\tau \in \Pi$

Note that, given this definition, there are an infinite number of parse trees for any sentence in the language generated by any grammar G. This is because for any valid parse tree τ, we can construct another parse tree τ' by simply adding an arbitrary (non-numeric) feature to any node in τ. Adding such a feature does not affect the yield of the tree; and it will still be the case that $\tau' \models p_0$ and $\tau' \in \Pi$.

3 Shieber's algorithm

Shieber's algorithm is specified as a set of 4 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 the model (parse tree), and d is the position of the "dot"; i.e., how many subconstituents in p have been completed so far.

The logical rules are shown in figure 2. The *Initial item* rule produces the first item, which represents the initial prediction as to what will be found in the input string. The item spans none of the input (i and j are both 0). It is constructed from the start

INITIAL ITEM

$$\overline{\langle 0, 0, p_0, mm(\Phi_0), 0\rangle}$$

SCANNING

$$\frac{\langle i, j, p = \langle a, \Phi\rangle, M, d\rangle}{\langle i, j+1, p, M \sqcup (mm(\Phi') \setminus \langle d+1\rangle), d+1}$$

PREDICTION

$$\frac{\langle i, j, p = \langle a, \Phi\rangle, M, d\rangle}{\langle j, j, p', mm(\Phi') \sqcup \rho(M/\langle d+1\rangle), 0\rangle}$$

COMPLETION

$$\frac{\langle i, j, p = \langle a, \Phi\rangle, M, d\rangle \qquad \langle j, k, p' = \langle a', \Phi'\rangle, M', a'\rangle}{\langle i, k, p, M \sqcup (M' \setminus \langle d+1\rangle), d+1\rangle}$$

Figure 2: Shiebers's parsing operations

production (p_0), and its model is the *minimal model* (mm) of this production (i.e., the model satisfying this production which contains the least amount of information).

The *Prediction* rule produces items which represent more specific predictions as to what will come next in the sentence. The function ρ, which is not specified as part of the abstract algorithm, determines how much and what kind of information is used in predictions. Intuitively, a prediction represents a top-down "filling in" of a parse tree (model) from a prior item. The rule produces a new item whose model is constructed from the subconstituent immediately to the right of the dot in a prior item, unified with the minimal model of a phrasal production. For example, assuming a rule such as $NP \rightarrow DET\ N$ in the grammar, the Prediction operator applied to the initial item would produce an item whose model satisfies this rule.

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 *Scanning* rule) or some other completed higher-level item (the *Completion* rule). Both rule perform the correct unification by utilizing the *embedding* operator (signified by $\setminus$). Embedding can be thought of as the inverse of extraction; that is, for all paths p, $M \setminus p/p = M$.

4 Minimal and Nonminimal Derivations

Earlier we noted that, according to Shieber's definition, there are an infinite number of valid parse trees for a sentence which is in the language generated by a particular grammar. Intuitively, we would like to exclude as valid those derivations which contain more information than is necessary.

Formally, we define a minimal derivation by defining a set of parse trees Π', where $\Pi' \subset \Pi$. Intuitively, parse trees in Π' contain the minimal amount of information necessary. Given a grammar G, we define Π' as $\bigcup_{i \geq 0} \Pi'_i$, where Π'_i is defined as follows:

1. For each lexical production $\langle w, \Phi\rangle$ in G, $mm(\Phi) \in \Pi'_0$.
2. For each phrasal production $\langle a, \Phi\rangle$ in G, let $\tau_1, \ldots, \tau_a \in \bigcup_{j \leq i} \Pi'_j$. if $\tau = mm(\Phi) \sqcup \tau_1 \setminus \langle 1\rangle \sqcup \ldots \sqcup \tau_a \setminus \langle a\rangle$, then $\tau \in \Pi'_i$.

A *minimal derivation* for a sentence is a model τ which is a valid parse (according to Shieber's definition) which is in Π'. Any parse which is not in Π' we will call a *nonminimal derivation*.

Nonminimal derivations are produced in at least three situations in Shieber's algorithm. All of them occur in conjunction with Completion operation, and can potentially generate items which have extra, irrelevant features (i.e., *nonminimal items*). These cases essentially originate from the same phenomenon: after an expectation spawns a sequence of Prediction operations top-down (forming a single Prediction path), it is later filled/completed by a complete item that is not the result of the exact bottom-up restoration of that path. When those cases happen, the inappropriate complete item may carry features that are collected from irrelevant path(s), either during its top-down propagation or bottom-up construction, or both. Those features essentially came from different contexts, but they are used to fill the expectation nonetheless because they did not cause any conflict with the existing features in the expectation.

We have identified the following three spurious situations: the first involves a single expectation and a single Prediction path (Case 1), the second involves a single expectation and multiple Prediction paths (Case 2), and the third involves multiple expectations and multiple Prediction paths (Case 3).

Case 1 (skipped left-recursion): This case happens when Completion is applied prematurely to fill the expectation by skipping left-recursive rules in the beginning of a Prediction path.

Suppose an n successive Prediction operations are performed (forming a single top-down Prediction path) using a rule sequence $s = (p_1, p_2, .., p_n)$ from an expectation, generating items $I_1, I_2, .., I_n$ respectively. After items I_n through I_2 are completed bottom-up, one more Completion should be applied to I_1 with (the complete) I_2 to fill its left-corner constituent (since the dot position of I_1 is still 0). It is only after I_1 becomes complete that the expectation should get filled, by the complete I_1 (i.e., complete

$p_3 = \langle 2, \Phi_3 \rangle$

$$\Phi_3 : \begin{array}{l} \langle\ cat \rangle = VP \\ \langle 1\ cat \rangle = VP \\ \langle 2\ cat \rangle = SLOWADV \\ \langle head \rangle = \langle 1\ head \rangle \\ \langle head\ speed \rangle = slow \end{array}$$

$p_4 = \langle 2, \Phi_4 \rangle$

$$\Phi_4 : \begin{array}{l} \langle\ cat \rangle = VP \\ \langle 1\ cat \rangle = V \\ \langle head \rangle = \langle 1\ head \rangle \\ \langle head\ type \rangle = intrans \end{array}$$

Figure 3: Additional grammar rules

restoration of the Prediction path s). However, if I_1 is left-recursive, the (syntactic) category of the left-hand side constituent of I_2 is the same as the category of the left-hand side constituent of I_1, which in turn is the same as the category of the expectation. Therefore, I_2 may well be able to fill the expectation directly, if they unify. Thus, Completion may be applied to the expectation with I_2 prematurely, skipping the last left-recursive I_1.

Note this skipped Prediction path essentially represents another path $s' = (p_2, .., p_n)$: a portion of s through which bottom-up construction actually took place (i.e., *minimal derivation*, as described in the next section).[3]

This situation will become problematic when the left-recursive p_1 had features which are not in the expectation originally, or not in the features that would have been collected by purely bottom-up parsing. Consider a sentence "John sleeps." is parsed using the start production shown previously in figure 2), along with the rules in figure 3.

Notice p_3 is left-recursive, and p_4 is non-recursive. After the word "John" is processed as an **NP** using p_0, the next expectation is **VP**. The item for this expectation is shown on the top-left corner in Figure 4.[4] Suppose two Predictions are applied successively from the expectation using rule sequence $s = (p_3, p_4)$[5] (indicated by arrows marked with (1) Prediction and (2) Prediction in Figure 4), and items **I1** and **I2** are created. Here in **I1**, $\langle head\ speed \rangle = slow$ feature was added to the original **VP** expectation by

[3] Since this path is also legitimate, parsing will generate the result for this path as another derivation, as well as one more derivation which is the result of complete Prediction path restoration for $s = (p_1, p_2, .., p_n)$.

[4] In Figures 4 and 6, items are represented in a simplified form, using their context-free backbone portion with a dot to indicate the arity (as in chart parsing).

[5] If parsing performs lookahead, p_3 will not be selected because SLOWADV will never be filled (end of the input sentence). However, without lookahead, parsing can validly select this rule.

$p_5 = \langle 2, \Phi_5 \rangle$

$$\Phi_5 : \begin{array}{l} \langle cat \rangle = VP \\ \langle 1\ cat \rangle = VP \\ \langle 2\ cat \rangle = SILENTADV \\ \langle head \rangle = \langle 1\ head \rangle \\ \langle head\ manner \rangle = silent \end{array}$$

Figure 5: Another grammar rule for verb phrases

the first Prediction using p_3, and propagated down to **I2** by the second Prediction through *head* arcs. As a result, the left-hand side **VP** and left-corner **V** in **I2** have this feature. Then, after "sleeps" is added by the next Scanning operation ((3) Scanning), the resulting item **I2'** still has $\langle head\ speed \rangle = slow$ feature under **VP** and **V**, since it did not cause any conflict.

Now that **I2'** is complete (since the arity of p_4 is 1), next operation should be a Completion, to fill the left-corner **VP** in **I1** (to create **I1'**, by (4) Completion). However, if this Completion is skipped and Completion is applied to the original expectation with **I2'** prematurely (indicated by a thick dashed line with **Completion***), the filled **VP** expectation has the feature $\langle head\ speed \rangle = slow$ which is propagated up from **VP** and **V** in **I2'** through *head* arcs. This feature came from the top-down prediction using s, but not from the actual bottom-up construction using another path $s' = (p_4)$, thus it is irrelevant with respect to s'. Therefore, the resulting item (with filled **VP** expectation) ends up carring this feature in addition to the relevant features collected bottom-up through s', and it becomes nonminimal.

Case 2 (mixed-up completion): This case happens when multiple Prediction paths, which are spawned from a single expectation, are mixed together while they are restored by Completion operations bottom-up. This is the case when multiple Prediction paths interact.

Suppose an expectation spawned two Prediction paths generating items $I_1, I_2, .., I_n$ and $J_1, J_2, .., J_m$ using rule sequences $s = (p_1, p_2, .., p_n)$ and $s' = (p'_1, p'_2, .., p'_m)$ respectively. While I path is being completed bottom-up, some intermediate result may be used in the bottom-up completions of J path. When it happens, (irrelevant) features in I path will be "mixed in" with the (relevant) features in J path in the resulting item.

Consider a sentence "John sleeps silently" is parsed using rules p_0, p_3, p_4 from the above example, and p_5 shown in figure 5.[6]

After "John" is processed as an **NP** using p_0, next expectation is **VP**, as shown at the top of in Figure 6. This expectation can spawn two Prediction paths

[6] Although p_3 and p_5 are left-recursive, this spurious situation can occur with non-recursive rules also.

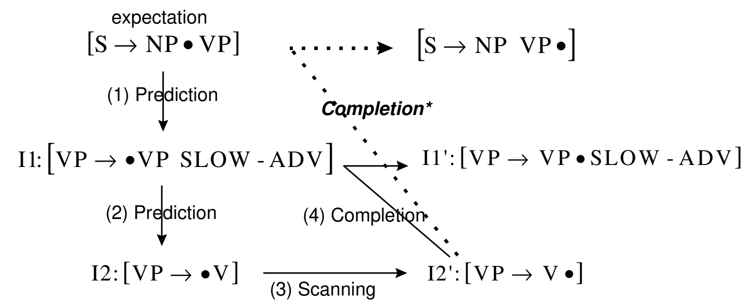

Figure 4: nonminimal Derivation for Case 1

$s = (p_3, p_4)$ and $s' = (p_5, p_4)$. For s path, item **I1** is first generated ((1**I**) Prediction, using p_3), and for s' path, item **J1** is first generated ((1**J**) Prediction, using p_5). At this point, **I1** has $\langle head\ speed \rangle = slow$ feature under its left-corner **VP**, and **J1** has $\langle head\ manner \rangle = silent$ feature under its left-corner **VP**. For each of them, another Prediction is applied with p_4 to bring down **VP** expectation to **V** to be matched with the next input word "sleeps", generating **I2** ((2**I**) Prediction) and **J2** ((2**J**) Prediction). In the resulting items, **I2** has $\langle head\ speed \rangle = slow$ feature under its left-corner **V** (pushed down from **I1**), and **J2** has $\langle head\ manner \rangle = silent$ feature under its left-corner **V** (pushsed down from **J1**). When "sleeps" is added to each of them (assuming no conflict), generating complete items **I2'** ((3**I**) Scanning) and **J2'** (not shown), next two Completions should use those complete items to restore their respective Prediction paths, namely **I2'** to fill the left-corner **VP** of **I1** ((4**I**) Completion), and **J2'** to fill the left-corner **VP** of **J1**. However, if there is no conflict between the left-hand side **VP** of **I2'** and the **VP** of **J1** (because *speed* and *manner* are different features), it may well happen that **I2'** is used to fill the **VP** of **J1** (indicated by a thick dashed line with **Completion***). Then, the resulting item **J1'** ends up with both $\langle head\ speed \rangle = slow$ and $\langle head\ manner \rangle = silent$ features.

In this example, since the actual bottom-up construction was done using p_4 (in **I2'**) and p_5 (in **J1**), the feature $\langle head\ speed \rangle = slow$ that came from p_3 is irrelevant and should not exist in the resulting item **J1'**. Therefore, this nonminimal item was created as a result of spurious interaction between s and s'.

Note that interaction can be even more subtle when there are multiple expectations or an expectation spawns many Prediction paths, or when Prediction paths are long; in those cases, spurious interaction may happen with many combinations at several levels during bottom-up construction.

Case 3 (mixed-up expectation): This case happens when there are multiple expectations, each of which spawns its own Prediction path(s), one expectation may be filled by a completion of another expectation (a totally different context). When this happens, features in the two expectations along with features from the wrong Prediction path are mixed together in the resulting item. Thus, this is the case when multiple expectations interact.

This case can be considered as a subcase of Case 2 in that multiple expectations at the input position i were spawned from the same original expectation further up in the Prediction path.

5 Discussion

Shieber chose *derivation* as the algorithm invariant, since derivability implies (partial) correctness in context-free parsing. However, derivability in the context of parsing algorithms is a notion associated with top-down processing only. Unfortunately, in the context of unification-based parsing, his invariant is effective only in the definition of Prediction operation. As discussed at the end of last section, once Prediction propagates expectation, Shieber's invariant is automatically satisfied in his algorithm; since unification is a monotonic operation, once this invariant is satisfied, it will never be refuted by subsequent unifications in bottom-up processing. Therefore, his invariant can not preclude nonminimal derivations.

Then, why do nonminimal derivations occur to begin with? We speculate the conjunction of four factors as the source: top-down processing using item-based parsing with unification grammars, particularly the ones with certain properties.

First, nonminimal derivations will not occur if parsing is done purely bottom-up, or only the "safe" features are passed from top-down (eg. context-free backbone). In both Case 1 and Case 2, extra features in nonminimal derivations were inserted during top-down Prediction, which will never appear in the (pure) bottom-up construction (i.e., in the minimal derivations).

Second, nonminimal derivations occur because inappropriate items are used to fill expectation. That

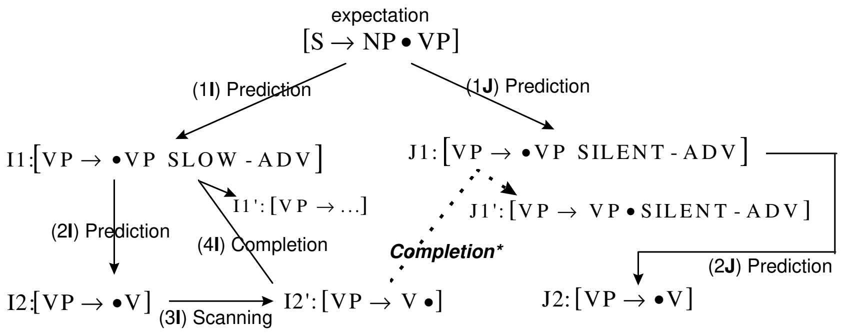

Figure 6: nonminimal Derivation for Case 2

is because items are accessible any time during parsing. This is an inherent problem with item-based parsing or chart-parsing (memoization) which records all contexts/items in one global structure. If items were kept local, interaction among items will be less likely to happen.

Third, nonminimal derivations will not occur with context-free grammars, as mentioned above. It is the additional features in unification grammars that cause the problem. In context-free parsing, the only information a constituent carries is the context-free symbol. Therefore, the notion of minimal or nonminimal derivation never exists.

Fourth, nonminimal derivations will occur only for certain unification grammars. As mentioned above, if the grammar can ensure that no "unsafe" features appear in the rules, or if those features can be pruned statically or dymanically, top-down Prediction propagation in Earley-like parsing will not produce nonminimal derivations.[7]

In summary, we must note that top-down prossesing with unification grammars can be problematic. Due to the additional information in feature structures, parsing with unification grammars requires more consideration than straight extension of context-free parsing, especially if top-down features are used.[8] To solve the nonminimal derivation problem, there seems to be several ways. The simplest way may be to use subsumption to restrict the items that are more specific than any other ones. This technique has been used to eliminate redundancy in several unification-based systems (eg. (Alshawi, 1992), (van Noord, 1997)), and it may be applicable in restricting the nonminimal derivations in implementation. However, this scheme requires more control aspect of the parsing algorithm in the logical specification (i.e., proving the absense of more general items at a given time during parsing). Another way is to define a stronger algorithm invariant to prevent such derivations. Or, we can develop a technique to filter out problematic features from models (or grammar rules). This scheme will essentially *weaken* the top-down constraints, therefore could be incorporated into the function ρ. Lastly, we can adopt another data structure, possibly based on some kind of stack similar to (Tomita, 1986, 1991), to keep the items local and to ensure the appropriate models are used to fill the expectation. However, coping with left-recursion may become a major difficulty. This is the subject of future research.

[7]For Head-corner parsing, (Sikkel, 1997) proposes the use of *transitive features*: the features that propagate only through head arcs. However, this scheme does not seem to solve the problem of nonminimal derivations completely.

[8]Another prominent issue concerning top-down prossesing with unification grammars is the Prediction nontermination. There have been several solutions proposed for this problem (eg. Shieber, 1985; Haas, 1989; Samuelsson, 1993).

References

Alshawi, H. (ed) (1992). *The Core Language Engine*, Cambridge, MA: MIT Press.

Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2).

Gazdar, G., G. Klein, G. Pullum, and I. Sag. (1985). *Generalized Phrase Structure Grammar*, Oxford, England: Blackwell Publishing and Cambridge, MA: Harvard University Press.

Haas, A. (1989). A parsing algorithm for unification grammar, *Computational Linguistics*, **15**(4), pp. 219-232.

Pereira, F. and D. Warren (1980). Definite clause grammars for language analysis – a survey of the formalism and a comparison with augmented transition networks. In *Artificial Intelligence*, 13, pp. 231-278.

Pollard, C. and I. Sag (1994). *Head-driven Phrase Structure Grammar*, Stanford, CA: Center for the Study of Language and Information. The University of Chicago Press.

Samuelsson, C. (1993). Avoiding non-termination in unification grammars. In *Proceedings of Natu-*

ral Language Understanding and Logic Programming IV, Nara, Japan.

Shieber, S. (1985). Using restriction to extend parsing algorithms for complex-feature-based formalisms. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, Chicago, IL, pp. 145-152.

Shieber, S. (1986). *An Introduction to Unification-Based Approaches to Grammar.* Stanford, CA: Center for the Study of Language and Information.

Shieber, S. (1992). *Constraint-based Grammar Formalisms*. Cambridge, MA: MIT Press.

Sikkel, K. (1997). *Parsing Schemata*, Berlin, Germany: Springer-Verlag.

Tomita, M. (1986). *Efficient Parsing for Natural Language.* Boston: Kluwer Academic Publishers.

Tomita, M. (1991). *Generalized LR Parsing.* Boston: Kluwer Academic Publishers

van Noord, G. (1997). An efficient implementation of the Head-Corner Parser, *Computational Linguistics*, **23**(3), pp. 425-456.