Squibs and Discussions

Restriction and Termination in Parsing with Feature-Theoretic Grammars

S. P. Harrison*
University of Western Australia

T. M. Ellison*
University of Western Australia

Shieber (1987) describes a technique for limiting the number of active edges introduced into a chart by top-down prediction in chart parsers for PATR grammars, without affecting the correctness or completeness of the parser. That technique, termed restriction, *is extendable to other parsing algorithms. It can be employed to increase parsing efficiency and to induce termination for some classes of grammars that would not otherwise terminate.*

Here, we describe one class of grammars for which restriction, as described by Shieber, induces non-termination. *We do not suggest that the concept of restriction is fatally flawed, however. On the contrary, relatively minor modifications to the implementation of restriction can make it a more flexible tool for fine-tuning PATR grammars.*

Background

Shieber (1987) observes that the potentially infinite category domain of feature-theoretic grammar formalisms like PATR-II makes implementing efficient parsers for such formalisms difficult. He concludes that the two earliest approaches to the problem are at best ill-advised. Parsing with a context-free backbone effectively ignores information in fact available to guide the parse and violates the spirit of feature-theoretic grammar. Tailoring the parsing algorithm or the grammar itself to the exigencies of feature-theoretic formalisms puts the burden of coping with the problem in the wrong place, on the end user of a grammar development system.

The most obvious alternative to parsing with a context-free backbone is using graph unification, rather than atomic symbol identity, to drive the parsing process. Shieber notes that not only is that approach costly (in using information that in fact can never affect the outcome of a parse) but, for some grammars, induces non-termination. The class of grammars Shieber describes might be termed *top-down path building* grammars, because they are grammars in which the length of a path through the graph increases as one descends the parse tree.

The particular grammar Shieber uses is one that counts the number of terminal symbols in the string being parsed:

(G1)
Rule 'set end marker'
$S \rightarrow T$:

$$\begin{aligned} \langle Sf \rangle &= \langle Tf \rangle \\ \langle Tf \rangle &= a. \end{aligned}$$

* Centre for Linguistics, University of Western Australia, Perth, Western Australia

Rule 'recursive clause'

$T_0 \rightarrow T_1 A$:

$$\langle T_0 f \rangle = \langle T_1 ff \rangle.$$

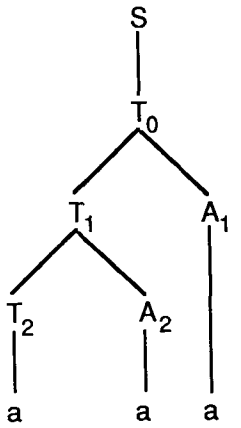
Rule 'base clause'

$$T \rightarrow A.$$

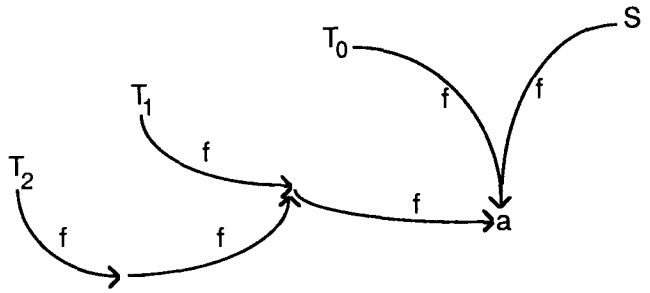
Rule 'lexical insertion'

$$A \rightarrow a.$$

G1 generates trees like the following:



In the course of a derivation, G1 builds a path $\langle T_n f^+ \rangle$ from the root of the graph T_n corresponding to each node T_n in the tree. The 'count' is reflected in the length of the path $\langle T_n f^+ \rangle$. As one descends the tree, the length of the path that is the value of the attribute f increases by 1 for each successive node T_{n+1}:



Hence the term *top-down path building*, characterizing grammars like G1. Recall that, in a chart parsing regime, for each active edge E_m of the form:

$$\langle i, j, X \rightarrow \alpha . Y \beta \rangle$$

and each rule R_i of the form:

$$Y \rightarrow \gamma$$

the top-down rule adds an active edge E_n of the form:

$$\langle j, j, Y \rightarrow . \gamma \rangle$$

If, associated with E_m, there is a graph G_m, and with R_i a graph G_i, then associated with E_n will be a graph $G_n = G_i \cup G_m(Y)$, where $G_m(Y)$ is the subgraph of G_m at Y. As Shieber points out, a parser employing this top-down rule will not terminate when interpreting grammar G1. On initialization (with Rule 'set end marker'), the parser generates an active edge seeking a constituent of category T. The top-down rule then uses that edge and the Rule 'recursive clause' to generate a new active edge seeking a constituent of category T. *That* active edge then generates a new (and, crucially, a *distinct*) active edge not subsumed by any existing edge, "and so forth, ad infinitum." In other words, each top-down recursion anticipates a different terminal string that is one symbol longer than that predicted on the previous recursion; because the graphs associated with those anticipated strings are distinct, the normal chart parsing checks fail to prevent looping.

Shieber's solution to the problem posed by grammar G1 is to introduce a notion he terms *restriction*. Under restriction, the graph associated with an edge generated by the top-down rule (applied to an active edge E_m and a grammar rule R_i) is not a graph $G = G_i \cup G_m(X)$, where G_i is the graph associated with R_i and $G_m(X)$ is that portion of the graph (associated with E_m) corresponding to the left hand side of R_i. Rather, $G = G_i \cup (G_m(X)!\Phi)$, where $G_m(X)!\Phi)$ is a graph G' resulting from filtering out of G_m all those paths not explicitly sanctioned by Φ. The filter Φ is termed a *restrictor*, and is represented as a set of paths P. The graph G' $(= G!\Phi)$ is derived from G by filtering out all those paths that are not members of P.[1]

Given a restrictor that passes through only the $\langle$cat$\rangle$ path, the termination problem for grammar G1 vanishes. Since the active edges generated by the top-down rule pass on only their category information, each time the rule 'recursive clause' is called recursively, the edge it generates is the same (underspecified) edge generated on the first call to that rule, and already on the chart.

The following observations regarding the restrictors Shieber describes might be made at this point:

1. The graph for the active edge generated by the top-down rule with a parser using restriction is no less specific (has no less information) than the graph stipulated *by the rule used*; i.e., it is information from the extant active edge that is suppressed.

2. The restrictor provides a *positive* restriction, i.e., it stipulates which paths are included in the restricted graph, not which paths are suppressed.

A Problem: Bottom-up Path Building

The following grammar might be considered almost the inverse of the grammar G1 described by Shieber (1987). It parses strings of length 1, and generates one node T in

1 Shieber's definition of restriction is as follows:

> "Given a relation F between paths and labels, and a dag D, we define $D!F$ to be the most specific dag D' $(D \sqsupseteq D')$ such that for every path p either $D'(p)$ is undefined, or $D'(p)$ is atomic, or for every $l \in \mathrm{dom}(D'(p)), p\Phi l$. That is, every path in the restricted dag is either undefined, atomic, or specifically allowed by the restrictor."

That definition might seem to suggest that *paths* that are instantiated (atom valued) are not filtered out by the restrictor. His examples demonstrate that that is not the correct interpretation of his definition. Rather, it would seem that atomic paths (i.e. 'leaves'), and not atomic-valued paths, fall through the filter.

the parse tree for each arc labeled g in the graph corresponding to the item recognized:

(G2)
Rule 'copy attribute'
$S \rightarrow AT$:

$$\langle Ag \rangle = \langle Tg \rangle.$$

Rule 'recursive clause'
$T_0 \rightarrow T_1$:

$$\langle T_0 gg \rangle = \langle T_1 g \rangle.$$

Rule 'base clause'
$T \rightarrow$:

$$\langle Tg \rangle = e.$$

Rule 'lexical insertion a'
$A \rightarrow a$:

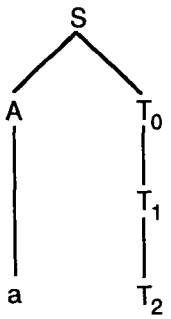
$$\langle Aggg \rangle = e.$$

Rule 'lexical insertion b'
$A \rightarrow b$:

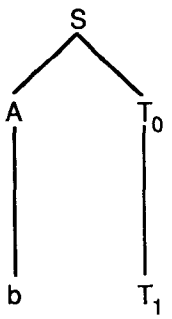
$$\langle Agg \rangle = e.$$

G2 generates exactly two trees:

Tree2a.

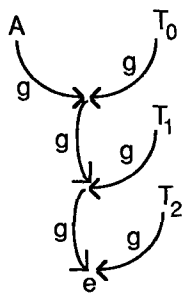


Tree2b.

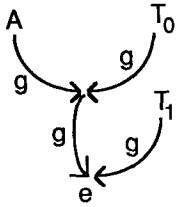


each of which is associated with a distinct graph:

Dag2a.



Dag2b.



For this grammar, the length of the path $\langle T_n g^+ \rangle$ in each subgraph T_n increases as one *ascends* the tree. Hence the term *bottom-up path building* is used to characterize grammars of this sort.

Using the same restrictor (i.e. ⟨cat⟩) employed for G1, a parser interpreting G2 will not terminate on any grammatical input. As in G1, only one active edge generated from the Rule 'recursive clause' is put on the chart:

E_1

$$\left\langle 1, 1, X_0 \rightarrow .X_1, \left[\begin{array}{ll} X_0: & \left[\begin{array}{l} cat: T \\ g: [g: \boxed{1}\,] \end{array} \right] \\ X_1: & \left[\begin{array}{l} cat: T \\ g: \boxed{1}\,[\,] \end{array} \right] \end{array} \right] \right\rangle$$

The rule 'base clause' of G2 generates the passive edge:

E_2

$$\left\langle 1, 1, X \rightarrow ., \left[X: \left[\begin{array}{ll} cat: & T \\ g: & e \end{array} \right] \right] \right\rangle$$

The fundamental rule then applies to E_1 and E_2 to yield the passive edge:

E_3

$$\left\langle 1, 1, X_0 \rightarrow X_1., \left[\begin{array}{ll} X_0: & \left[\begin{array}{l} cat: T \\ g: [g: \boxed{1}\,] \end{array} \right] \\ X_1: & \left[\begin{array}{l} cat: T \\ g: \boxed{1}\, e \end{array} \right] \end{array} \right] \right\rangle$$

and then applies to E_1 and E_3 to yield:

E_4

$$\left\langle 1, 1, X_0 \rightarrow X_1., \left[\begin{array}{ll} X_0: & \left[\begin{array}{l} cat: T \\ g: [g: \boxed{1}\,] \end{array} \right] \\ X_1: & \left[\begin{array}{l} cat: T \\ g: \boxed{1}\,[g: e] \end{array} \right] \end{array} \right] \right\rangle$$

and so forth, ad infinitum. Ultimately, the fundamental rule will generate a passive edge with a graph G such that $G(g) = \{G_i : T'(g)\}$, for the passive edge:

$$\langle 0, 1, S \rightarrow A.T, G_i \rangle$$

where $\{G_i : T'(g)\}$ is the subgraph corresponding to the constituent T. At that point, the parser, in effect, will have parsed the input string. Nonetheless, it will continue to add new passive edges, each with a graph whose $\langle g^+ \rangle$ path is one arc longer than that of the passive edge from which it is generated.

Bottom-up path building grammars are not without interesting linguistic applications. Consider, for example, the following PATR implementation of the *append* operation on lists employed in HPSG (Pollard and Sag 1987; Section 2.2):

(G3)
Rule 'collect lists'

Top → Left Right Append:

$$\begin{array}{lcl} \langle \text{Append list1} \rangle & = & \langle \text{Left list} \rangle \\ \langle \text{Append list2} \rangle & = & \langle \text{Right list} \rangle \\ \langle \text{Top list} \rangle & = & \langle \text{Append list} \rangle. \end{array}$$

Rule 'base case'
Append →:

$$\begin{aligned} \langle \text{Append list} \rangle &= \langle \text{Append list2} \rangle \\ \langle \text{Append list1} \rangle &= (). \end{aligned}$$

Rule 'recursive clause'
$\text{Append}_0 \rightarrow \text{Append}_1$:

$$\begin{aligned} \langle \text{Append}_0 \text{ list first} \rangle &= \langle \text{Append}_0 \text{ list1 first} \rangle \\ \langle \text{Append}_0 \text{ list rest} \rangle &= \langle \text{Append}_1 \text{ list} \rangle \\ \langle \text{Append}_0 \text{ list1 rest} \rangle &= \langle \text{Append}_1 \text{ list1} \rangle \\ \langle \text{Append}_1 \text{ list2} \rangle &= \langle \text{Append}_0 \text{ list2} \rangle. \end{aligned}$$

Given the rules 'base case' and 'recursive clause,' one need only introduce a 'dummy' constituent Append (whose only function is to collect the list-valued attributes list1 and list2 in a single list), and appropriate constraints on some list-valued attribute, in order to simulate the HPSG *append* in PATR. Unfortunately, because these two rules are an instance of the bottom-up path building problem, no derivation employing these rules will terminate.[2]

Solutions

The source of the bottom-up path building problem is the active edge E_1 above, whose $\langle g^+ \rangle$ path is of indeterminate length. Because the parser does not know how long that path should be, it keeps adding to it *ad infinitum*. Observe, however, that the parser had the required information (obtained from the $\langle g \rangle$ attribute of constituent A), but was induced *by the restrictor* to throw that information away.

There are at least three solutions to the problem posed by bottom-up path building grammars under restriction:

1. Impose a finite limit on the path $\langle g^+ \rangle$.
2. Convert the grammar to top-down path building.
3. Change the nature and implementation of restriction.

We consider each of these proposals in turn in the following paragraphs.

Solution 1 involves picking an arbitrary upper limit on the length of $\langle g^+ \rangle$, and adding a corresponding path to the restrictor. The parser would then terminate for any input whose $\langle g^+ \rangle$ path length is less than or equal to that of the stipulated path. For example, a parser interpreting G2 above would terminate with the restrictor $\{\langle cat \rangle \langle g\ g\ g \rangle\}$. But that is a "Band-Aid" solution at best.

2 The *append* operation is necessary whenever a grammar demands that two list-valued attributes be combined; for example, when the SUBCAT of a constituent is defined as the union of the values of SUBCAT for its daughters, or when the list-valued SKELETON of a syllable is defined as the SKELETON of the onset followed by the SKELETON of the rhyme. One would, of course, prefer to have PATR extended to accommodate *append*, rather than have to simulate the *append* operation in pure PATR, but that is not the issue here. The point we are making is simply that one *could* simulate *append* in pure PATR, were it not for the restriction regime of the PATR standard defined in Shieber (1989).

Solution 2 might involve a grammar like G4, a top-down path building grammar weakly equivalent to G2. G4 does not, however, have the termination problem of G2:

(G4)
Rule 'base case'
$S \rightarrow T$:

$$\langle Tg \rangle = e.$$

Rule 'recursive clause'
$T_0 \rightarrow T_1$:

$$\langle T_0 g \rangle = \langle T_1 gg \rangle.$$

Rule 'copy path'
$T \rightarrow A$:

$$\langle Tg \rangle = \langle Ag \rangle.$$

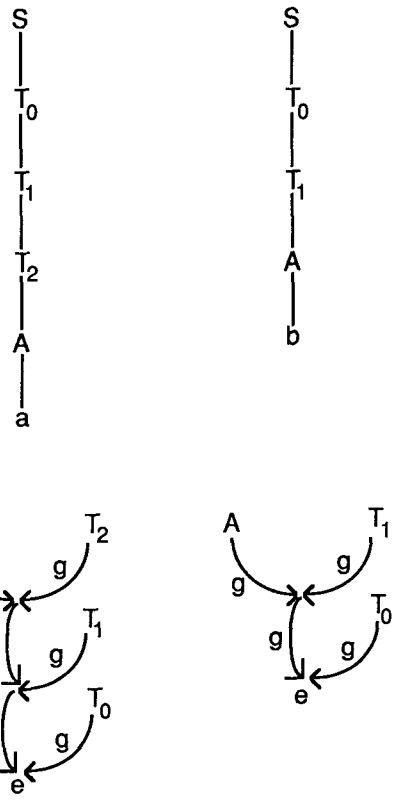
Rule 'lexical insertion a'
$A \rightarrow a$:

$$\langle Aggg \rangle = e.$$

Rule 'lexical insertion b'
$A \rightarrow b$:

$$\langle Agg \rangle = e.$$

(The following are the trees, and corresponding complex feature structures, generated by G4:



Note that the length of the path g^+ increases as one *descends* the tree.) Similarly, grammar G5 provides a top-down path building solution to the *append* operation:

(G5)
Rule 'collect lists'
Append → Left Right:

$$\begin{aligned} \langle \text{Append residue} \rangle &= \langle \text{Append list} \rangle \\ \langle \text{Append list1} \rangle &= \langle \text{Left list} \rangle \\ \langle \text{Append list2} \rangle &= \langle \text{Right list} \rangle. \end{aligned}$$

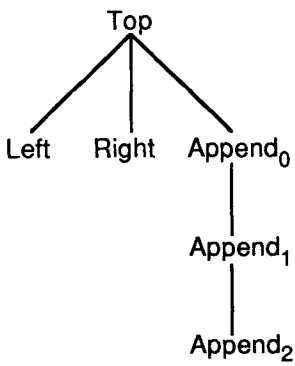
Rule 'append base case'
Top → Append:

$$\begin{aligned} \langle \text{Top list} \rangle &= \langle \text{Append list} \rangle \\ \langle \text{Append residue} \rangle &= \langle \text{Append list2} \rangle \\ \langle \text{Append list1} \rangle &= (). \end{aligned}$$

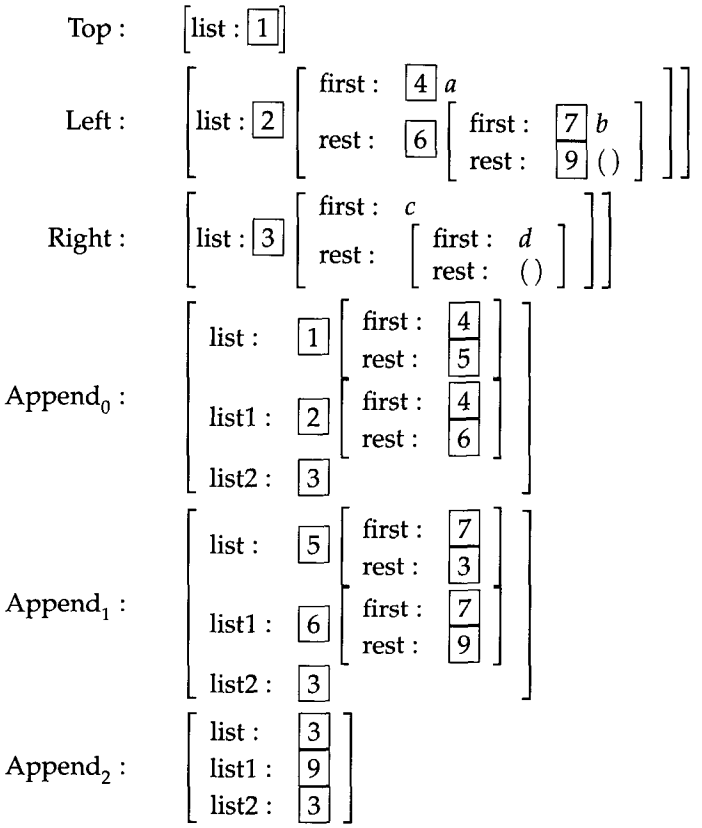
Rule 'append recursive clause'
$\text{Append}_0 \rightarrow \text{Append}_1$:

$$\begin{aligned} &\langle \text{Append}_0 \text{ list} \rangle = \langle \text{Append}_1 \text{ list} \rangle \\ &\langle \text{Append}_0 \text{ list1} \rangle = \langle \text{Append}_1 \text{ list1 rest} \rangle \\ &\langle \text{Append}_0 \text{ list2} \rangle = \langle \text{Append}_1 \text{ list2} \rangle \\ &\langle \text{Append}_0 \text{ residue} \rangle = \langle \text{Append}_1 \text{ residue rest} \rangle \\ &\langle \text{Append}_1 \text{ residue first} \rangle = \langle \text{Append}_1 \text{ list1 first} \rangle. \end{aligned}$$

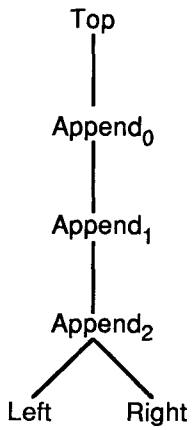
Consider the case of two descendants, Left and Right, of a node Top, where the value of ⟨Left list⟩ is (a, b), the value of ⟨Right list⟩ is (c, d), and the value of ⟨Top list⟩ is *append* (⟨L list⟩, ⟨R list⟩). G3 assigns to that construction the parse tree:



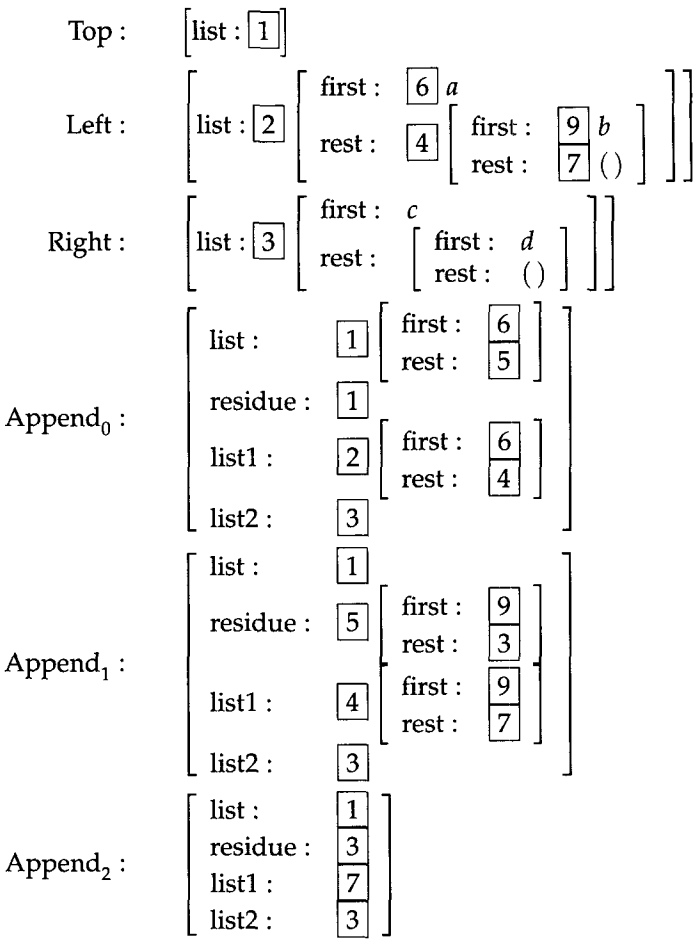
and the complex feature structure:



G5, by contrast, assigns it the parse tree:



and the complex feature structure:



As noted above, G3 provides a 'dummy' constituent Append that one might want ultimately to prune, since its only function is to assemble the appended list. By contrast, G5 assembles the appended list in successive Append nodes dominating the constituents Left and Right. One might want to argue that the latter solution is less felicitous than the former because the Append nodes dominate lexical material. In any event, as Shieber stresses, the grammar writer should not be constrained to choose between weakly equivalent grammars by the demands of the grammar development system.

Let us now consider solution 3, modifying the nature and implementation of restriction. We choose to term the sort of restriction Shieber describes *positive restriction*. In positive restriction, the restrictor tells the parser *how much* of the information for *which* paths to retain (in the graph corresponding to the active edge used by the top-down rule). If a path is not explicitly mentioned in the restrictor, no information about that path is retained. An empty restrictor in effect tells the parser to throw away all the information in the restricted graph.

(An anonymous reviewer for *Computational Linguistics* points out that grammars G2 and G3, in contrast to Shieber's G1, are not offline parsable [in the sense of Bresnan and Kaplan 1982:263ff and Johnson 1988:95ff], and suggests that this failure of offline parsability is a necessary condition for membership in the class of grammars that are

nonterminating under positive restriction. The constituent structures generated by a grammar are offline parsable iff:

i. they contain no nonbranching dominance chains in which the same category [label] appears twice, and

ii. all terminal nodes dominate lexical material

G2 and G3 fail both these conditions, since neither the T node chain of G2 nor the Append node chain of G3 either branch or dominate lexical material. We make two observations in this regard. First, while it may indeed be the case that failure of offline parsability is a *necessary* condition for non-termination under positive restriction, it is certainly not a *sufficient* one, since G4 and G5 both fail condition i. above, and are thus not offline parsable, but do terminate under positive restriction. Second, though G2 through G5 fail the letter of an offline parsability constraint, they do not fail its spirit.

Offline parsability has been proposed as a constraint on grammars guaranteeing the decidability of the recognition problem for feature-theoretic grammars of the sort considered here. For grammars that satisfy offline parsability, it will be the case that the number of nodes of any parse tree assigned to some string will be a computable function of the length of the string being parsed. This is not the case for G2 through G5. What is the case, however, is that the number of nodes in a parse tree generated by those grammars is a computable function of the graphs associated with the lexical material covered by the tree. This is because each of these grammars makes crucial reference either to a list-valued or path-valued attribute of lexical items, where the length of the list or path is finite and stipulated in the lexicon. The number of identical nodes in any nonbranching dominance chain generated by these grammars is a function of the length of a [bounded] list or path.)

Positive restriction can be contrasted with what might be termed *negative restriction*. Under negative restriction, the restrictor tells the parser what information to throw away. For stipulated paths, the effects of positive and negative restriction are the same: a restrictor $\langle ggg \rangle$ tells the parser to (positively) retain all information up to and including an atomic value for the path $\langle ggg \rangle$, or to (negatively) throw away all information about any nonatomic extension of $\langle ggg \rangle$. The difference is in the interpretation of paths not explicitly mentioned; under positive restriction these are thrown away; under negative restriction they are retained.

In formal terms, a restrictor, for Shieber, is a relation between paths and the single edges that are permitted to extend those paths: If Φ is a restrictor, p a path and l some edge label, then the restrictor permits the path pl iff $p\Phi l$. But Shieber does not use the full power of this formalism. He considers only what we term *positive restrictors*. Mathematically, if P is some finite set of paths, one can define a positive restrictor Φ_P^+ to be the relation defined by:

$$p\Phi_P^+ l \text{ iff } \exists\, q \in P,\ p \leq q$$

where $p \leq q$ means that p is a (perhaps improper) prefix to the path q. By contrast, the *negative restriction* regime proposed here allows paths not explicitly disallowed:

$$p\Phi_P^- l \text{ iff } (\exists\, r \in P,\ r < p) \Rightarrow \exists\, q \in P,\ p \leq q \tag{1}$$

In other words, whereas the positive restrictor admits those paths obeying the condition "is a prefix of a member of the restrictor set," the negative restrictor only applies

this condition to paths extending members of the restrictor set. Paths not extending members of the restrictor set are themselves not subject to restriction.

It is an easy matter to show that any positive restrictor p can be defined in terms of a negative restrictor. If the null path is included in the restrictor set, then the antecedent of the implication in the definition of Φ^- (1 above) is always true (except if $p = \langle\ \rangle$), and the consequent $\exists q \in P \cup \{\langle\ \rangle\},\ p \leq q$ is equivalent to $\exists q \in P,\ p \leq q$ (except when $p = \langle\ \rangle$). If p is the null path, then the antecedent is always true and so $\langle\ \rangle\Phi^-_{P\cup\{\langle\rangle\}}l$ is equivalent to $\langle\rangle\Phi^+_P l$. For other p, $\langle\rangle\Phi^-_{P\cup\{\langle\rangle\}}l = \langle\rangle\Phi^+_P l$. So, $\Phi^-_{P\cup\{\langle\rangle\}} = \Phi^+_P$. Thus, an implementation using negative restriction can simulate any positive restrictor and, therefore, can necessarily solve all problems solvable by positive restriction, as well as those problems, described earlier, where positive restriction fails.

Note that we have limited ourselves, as Shieber has, to finite restrictor sets. This limitation is not necessary. A more general proposal allows restrictor sets consisting of all paths expressable by some regular expression. Restrictors can then be implemented as finite state automata. An implementation of this sort would not impose a large computational burden and would allow much greater flexibility in the choice of restrictor relations.

We have demonstrated that any restriction definable by a positive restrictor can also be defined by a negative restrictor. It follows that any parser requiring positive restriction to guarantee termination will also terminate for the same set of grammars using negative restriction. But we have also demonstrated that the reverse does not hold. Therefore, it would seem that negative restriction is superior to positive restriction in the implementation of parsers for feature–theoretic grammars.

References

Johnson, M. (1988). "Attribute-value logic and the theory of grammar." *CSLI Lecture Notes*, 16.

Kaplan, R., and Bresnan, J. (1982). "Lexical-functional grammar, a formal system for grammatical representation." In *The Mental Representation of Grammatical Relations*, edited by J. Bresnan, 173–281. MIT Press.

Pollard, C., and Sag, I. A. (1987). "Information-based syntax and semantics." *CSLI Lecture Notes*, 13(1).

Shieber, S. M. (1987). "Using restriction to extend parsing algorithms for complex-feature-based formalisms." *Proceedings, 23rd Annual Meeting of the ACL*. 154–52.