A UNIFICATION-BASED PARSER FOR RELATIONAL GRAMMAR

David E. Johnson
IBM Research Division
P.O. Box 218
Yorktown Heights, NY 10598
djohns@watson.ibm.com

Adam Meyers
Linguistics Department
New York University
New York, NY 10003
meyers@acf2.nyu.edu

Lawrence S. Moss
Mathematics Department
Indiana University
Bloomington, IN 47401
lmoss@indiana.edu

Abstract

We present an implemented unification-based parser for relational grammars developed within the **stratified feature grammar (SFG)** framework, which generalizes Kasper-Rounds logic to handle relational grammar analyses. We first introduce the key aspects of SFG and a lexicalized, graph-based variant of the framework suitable for implementing relational grammars. We then describe a head-driven chart parser for lexicalized SFG. The basic parsing operation is essentially ordinary feature-structure unification augmented with an operation of *label unification* to build the stratified features characteristic of SFG.

INTRODUCTION

Although the impact of relational grammar (RG) on theoretical linguistics has been substantial, it has never previously been put in a form suitable for computational use. RG's multiple syntactic strata would seem to preclude its use in the kind of monotonic, unification-based parsing system many now consider standard ([1], [11]). However, recent work by Johnson and Moss [2] on a Kasper-Rounds (KR) style logic-based formalism [5] for RG, called **Stratified Feature Grammar (SFG)**, has demonstrated that even RG's multiple strata are amenable to a feature-structure treatment.

Based on this work, we have developed a unification-based, chart parser for a lexical version of SFG suitable for building computational relational grammars. A lexicalized SFG is simply a collection of **stratified feature graphs (S-graphs)**, each of which is anchored to a lexical item, analogous to lexicalized TAGs [10]. The basic parsing operation of the system is **S-graph unification (S-unification)**: This is essentially ordinary feature-structure unification augmented with an operation of *label unification* to build the stratified features characteristic of SFG.

RELATED WORK

Rounds and Manaster-Ramer [9] suggested encoding multiple strata in terms of a "level" attribute, using path equations to state correspondences across strata. Unfortunately, "*unchanged*" relations in a stratum must be explicitly "carried over" via path equations to the next stratum. Even worse, these "carry over" equations vary from case to case. SFG avoids this problem.

STRATIFIED FEATURE GRAMMAR

SFG's key innovation is the generalization of the concept *feature* to a *sequence* of so-called **relational signs (R-signs)**. The interpretation of a **stratified feature** is that each R-sign in a sequence denotes a primitive relation in different strata. [1]

For instance, in *Joe gave Mary tea* there are, at the clause level, four *sister* arcs (arcs with the same source node), as shown in Figure 1: one arc labeled [H] with target *gave*, indicating *gave* is the head of the clause; one with label [1] and target *Joe*, indicating *Joe* is both the predicate-argument, and surface subject, of the clause; one with label [3,2] and target *Mary*, indicating that

[1] We use the following R-signs: 1 (subject), 2 (direct object), 3 (indirect object), 8 (chômeur), Cat (Category), C (comp), F (flag), H (head), LOC (locative), M (marked), as well as the special *Null* R-signs 0 and /, explained below.

$$\begin{bmatrix} [Cat] & \texttt{S} \\ [1] & \texttt{Joe} \\ [H] & \texttt{gave} \\ [3,2] & \texttt{Mary} \\ [2,8] & \texttt{tea} \end{bmatrix}$$

Figure 1: S-graph for *Joe gave Mary tea.*

Mary is the predicate-argument indirect object, but the surface direct object, of the clause; and one with label [2,8] and target *tea*, indicating *tea* is the predicate-argument direct object, but surface chômeur, of the clause. Such a structure is called a **stratified feature graph (S-graph)**.

This situation could be described in SFG logic with the following formula (the significance of the different label delimiters (,), [,] is explained below):

$$\textbf{R1} := \quad [\mathrm{H}] : \texttt{gave} \ \wedge \ [1) : \texttt{Joe}$$
$$\wedge \ [3,2) : \texttt{Mary} \quad \wedge \quad [2,8) : \texttt{tea} \ .$$

In RG, the clause-level syntactic information captured in R1 combines two statements: one characterizing *gave* as taking an initial 1, initial 2 and initial 3 (**Ditransitive**); and one characterizing the concomitant "advancement" of the 3 to 2 and the "demotion" of the 2 to 8 (**Dative**). In SFG, these two statements would be:

$$\textbf{Ditransitive} :=$$
$$[H] : \texttt{gave} \ \wedge \ [1) : \mathrm{T} \ \wedge \ [2) : \mathrm{T} \ \wedge \ [3) : \mathrm{T} \ ;$$
$$\textbf{Dative} := \quad (3, \underline{2}) : \mathrm{T} \ \& \ (2, \underline{8}) : \mathrm{T}.$$

Ditransitive involves standard Boolean conjunction ($\wedge$). Dative, however, involves an operator, &, unique to SFG. Formulas involving & are called *extension formulas* and they have a more complicated semantics. For example, Dative has the following informal interpretation: Two distinct arcs with labels 3 and 2 may be "extended" to (3,2) and (2,8) respectively. Extension formulas are, in a sense, the heart of the SFG description language, for without them RG analyses could not be properly represented. [2]

[2] We gloss over many technicalities, e.g., the SFG notion *data justification* and the formal semantics of stratified features; cf. [2].

RG-style analyses can be captured in terms of rules such as those above. Moreover, since the above formulas state positive constraints, they can be represented as S-graphs corresponding to the minimal satisfying models of the respective formulas. We compile the various rules and their combinations into **Rule Graphs** and associate sets of these with appropriate lexical anchors, resulting in a lexicalized grammar. [3]

S-graphs are formally feature structures: given a collection of sister arcs, the stratified labels are required to be functional. However, as shown in the example, the individual R-signs are not. Moreover, the lengths of the labels can vary, and this crucial property is how SFG avoids the "carry over" problem. S-graphs also include a strict partial order on arcs to represent linear precedence (cf. [3], [9]). The SFG description language includes a class of **linear precedence** statements, e.g., $(1) \prec (H]$ means that in a constituent "the final subject precedes the head".

Given a set $\mathcal{RS}$ of R-signs, a **(stratified) feature** (or **label**) is a sequence of R-signs which may be closed on the left or right or both. Closed sides are indicated with square brackets and open sides with parentheses. For example, [2, 1) denotes a label that is closed on the left and open on the right, and [3, 2, 1, 0] denotes a label that is closed on both sides. Labels of the form $[\cdots]$ are called **(totally) closed**; of the form $(\cdots)$ **(totally) open**; and the others **partially closed (open)** or **closed (open) on the right (left)**, as appropriate.

Let $\mathcal{BL}$ denote the set of features over $\mathcal{RS}^*$. $\mathcal{BL}$ is partially ordered by the smallest relation $\sqsubseteq$ permitting *extension* along open sides. For example,

$$(3) \sqsubseteq (3,2) \sqsubseteq [3,2,1) \sqsubseteq [3,2,1,0].$$

Each feature l subsuming ($\sqsubseteq$) a feature f provides a partial description of f. The left-closed bracket [allows reference to the "deepest" (*initial*) R-sign of a left-closed feature; the right-closed bracket] to the "most surfacy" (*final*) R-sign of a right-closed feature. The totally closed features are maximal (completely defined) and with respect to label unification, defined below, act like ordinary (atomic) features.

Formal definitions of *S-graph* and other definitions implicit in our work are provided in [2].

[3] We ignore negative constraints here.

AN EXAMPLE

Figure 2 depicts the essential aspects of the S-graph for *John seemed ill.* Focus on the features [0,1] and [2,1,0], both of which have the NP *John* as target (indicated by the [i]'s). The R-sign 0 is a member of **Null**, a distinguished set of R-signs, members of which can *only* occur next to brackets [or]. The prefix [2,1) of the label [2,1,0] is the SFG representation of RG's unaccusative analysis of adjectives. The suffix (1,0] of [2,1,0]; the prefix [0,1) of the label [0,1] in the matrix clause; and the structure-sharing collectively represent the raising of the embedded subject (cf. Figure 3).

Given an S-graph G, *Null* R-signs permit the definitions of the **predicate-argument graph**, and the **surface graph**, of G. The predicate-argument graph corresponds to all arcs whose labels do *not begin* with a *Null* R-sign; the relevant R-signs are the *first* ones. The surface graph corresponds to all arcs whose labels do *not end* with a *Null* R-sign; the relevant R-signs are the *final* ones. In the example, the arc labeled [0,1] is not a predicate-argument arc, indicating that *John* bears no predicate-argument relation to the top clause. And the arc labeled [2,1,0] is not a surface arc, indicating that *John* bears no surface relation to the embedded phrase headed by *ill.*

The surface graph is shown in Figure 4 and the predicate-argument graph in Figure 5. Notice that the surface graph is a tree. The **treehood of surface graphs** is part of the definition of S-graph and provides the foundation for our parsing algorithm; it is the SFG analog to the "context-free backbone" typical of unification-based systems [11].

LEXICALIZED SFG

Given a finite collection of rule graphs, we could construct the finite set of S-graphs reflecting all consistent combinations of rule graphs and then associate each word with the collection of derived graphs it anchors. However, we actually only construct all the derived graphs *not* involving extractions. Since extractions can affect almost any arc, compiling them into lexicalized S-graphs would be impractical. Instead, extractions are handled by a novel mechanism involving multi-rooted graphs (cf. Concluding Remarks).

We assume that all lexically governed rules such as Passive, Dative Advancement and Raising are compiled into the lexical entries governing them.

$$\begin{bmatrix} [Cat] & \texttt{VP} \\ [0,1] & \boxed{\texttt{i}}\ \texttt{John} \\ [H] & \texttt{seemed} \\ [C] & \begin{bmatrix} [Cat] & \texttt{AP} \\ [2,1,0] & \boxed{\texttt{i}} \\ [H] & \texttt{ill} \end{bmatrix} \end{bmatrix}$$

Figure 2: S-graph for *John seemed ill*

$$\begin{bmatrix} [0,1) & \boxed{\texttt{i}} \\ [C] & \begin{bmatrix} (1,0] & \boxed{\texttt{i}} \end{bmatrix} \end{bmatrix}$$

Figure 3: Raising Rule Graph

$$\begin{bmatrix} [Cat] & \texttt{VP} \\ (1] & \boxed{\texttt{i}}\ \texttt{John} \\ [H] & \texttt{seemed} \\ [C] & \begin{bmatrix} [Cat] & \texttt{AP} \\ [H] & \texttt{ill} \end{bmatrix} \end{bmatrix}$$

Figure 4: Surface Graph for *John seemed ill*

$$\begin{bmatrix} [Cat] & \texttt{VP} \\ [H] & \texttt{seemed} \\ [C] & \begin{bmatrix} [Cat] & \texttt{AP} \\ [2) & \boxed{\texttt{i}}\ \texttt{John} \\ [H] & \texttt{ill} \end{bmatrix} \end{bmatrix}$$

Figure 5: Predicate-Argument Graph for *John seemed ill*

Thus, *given* has four entries (Ditransitive, Ditransitive + Dative, Passive, Dative + Passive). This aspect of our framework is reminiscent of LFG [4] and HPSG [7], except that in SFG, relational structure is transparently recorded in the stratified features. Moreover, SFG relies neither on LFG-style annotated CFG rules and equation solving nor on HPSG-style SUBCAT lists.

We illustrate below the process of constructing a lexical entry for *given* from rule graphs (ignoring morphology). The rule graphs used are for Ditransitive, Dative and (Agentless) Passive constructions. Combined, they yield a ditransitive-dative-passive S-graph for the use of *given* occurring in *Joe was given tea* (cf. Figure 6).

DItransitive:

$$\begin{bmatrix} [H] & \texttt{given} \\ [3) & \\ [2) & \\ [1) & \end{bmatrix}$$

DATive:

$$\begin{bmatrix} (2,8) \\ (3,2) \end{bmatrix}$$

DI ⊔ DAT:

$$\begin{bmatrix} [H] & \texttt{given} \\ [3,2) & \\ [2,8) & \\ [1) & \end{bmatrix}$$

PASsive:

$$\begin{bmatrix} (2,1) \\ [1,8,0] \end{bmatrix}$$

$$\begin{bmatrix} [Cat] & \texttt{S} & \\ [0,1] & \boxed{\texttt{i}}\ \texttt{Joe} & \\ [H] & \texttt{was} & \\ & \begin{bmatrix} [Cat] & \texttt{VP} \\ [H] & \texttt{given} \\ [3,2,1,0] & \boxed{\texttt{i}} \\ [2,8] & \texttt{tea} \\ [1,8,0] & \end{bmatrix} \\ [C] & \end{bmatrix}$$

Figure 6: S-graph for *Joe was given tea.*

$\Big(\textbf{DI} \sqcup \textbf{DAT}\Big) \sqcup$ **PAS:**

$$\begin{bmatrix} [H] & \texttt{given} \\ [3,2,1) & \\ [2,8) & \\ [1,8,0] & \end{bmatrix}$$

The idea behind **label unification** is that two compatible labels combine to yield a label with **maximal nonempty overlap.** Left (right) closed labels unify with left (right) open labels to yield left (right) *closed* labels. There are ten types of label unification, determined by the four types of bracket pairs: totally closed (open), closed only on the left (right). However, in parsing (as opposed to building a lexicalized grammar), we stipulate that successful label unification must result in a *totally closed label.* Additionally, we assume that all labels in well-formed lexicalized graphs (the input graphs to the parsing algorithm) are at least partially closed. This leaves only four cases:

Case 1. $[\alpha] \sqcup [\alpha] = [\alpha]$

Case 2. $(\alpha) \sqcup [\alpha\beta] = [\alpha\beta]$

Case 3. $(\alpha] \sqcup [\beta\alpha] = [\beta\alpha]$

Case 4. $[\alpha\beta) \sqcup (\beta\gamma] = [\alpha\beta\gamma]$

Note: $\alpha, \beta, \gamma \in \mathcal{RS}^*$ and β is the longest common, nonempty string.

The following list provides examples of each.

1. [1,0] ⊔ [1,0] = [1,0]
2. [1) ⊔ [1,0] = [1,0]
3. (1,0] ⊔ [2,1,0] = [2,1,0]
4. [2,1) ⊔ (1,0] = [2,1,0]

Case 1 is the same as ordinary label unification under identity. Besides their roles in unifying rule-graphs, Cases 2, 3 and 4 are typically used in parsing bounded control constructions (e.g., "equi" and "raising") and extractions by means of "splicing" *Null* R-signs onto the open ends of labels and closing off the labels in the process. We note in passing that cases involving totally open labels may not result in unique unifications, e.g., $(1,2) \sqcup (2,1)$ can be either (2,1,2) or (1,2,1). In practice, such aberrant cases seem not to arise. Label unification thus plays a central role in building a lexicalized grammar and in parsing.

THE PARSING ALGORITHM

S-unification is like normal feature structure unification ([1], [11]), except that in certain cases two arcs with distinct labels l and l' are replaced by a single arc whose label is obtained by unifying l and l'.

S-unification is implemented via the procedures **Unify-Nodes**, **Unify-Arcs**, and **Unify-Sets-of-Arcs**:

1. **Unify-Nodes**(n,n') consists of the steps:
   a. Unify label(n) and label(n'), where node labels unify under identity
   b. Unify-Sets-of-Arcs(Out-Arcs(n), Out-Arcs(n'))
2. **Unify-Arcs**(A,A') consists of the steps:
   a. Unify label(A) and label(A')
   b. Unify-Nodes(target(A),target(A'))
3. **Unify-Sets-of-Arcs**(Set_1, Set_2), where $Set_1 = \{A_j, \ldots, A_k\}$ and $Set_2 = \{A_m, \ldots, A_n\}$, returns a set of arcs Set_3, derived as follows:
   a. For each arc $A_i \in Set_1$, attempt to find some arc $A'_i \in Set_2$, such that Step 2a of Unify-arcs(A_i, A'_i) succeeds. If Step 2a succeeds, proceed to Step 2b and remove A'_i from Set_2. There are three possibilities:
      i. If no A'_i can be found, $A_i \in Set_3$.
      ii. If Step 2a and 2b both succeed, then Unify-arcs(A_i, A'_i) $\in Set_3$.
      iii. If Step 2a succeeds, but Step 2b fails, then the procedure fails.
   b. Add each remaining arc in Set_2 to Set_3.

We note that the result of S-unification can be a set of S-graphs. In our experience, the unification of linguistically well-formed lexical S-graphs has never returned more than one S-graph. Hence, S-unification is stipulated to fail if the result is not unique. Also note that due to the nature of label unification, the unification procedure does not guarantee that the unification of two S-graphs will be functional and thus well-formed. To insure functionality, we filter the output.

We distinguish several classes of Arc: (i) Surface Arc vs. Non-Surface, determined by absence or presence of a *Null* R-sign in a label's *last* position; (ii) Structural Arc vs. Constraint Arc (stipulated by the grammar writer); and (iii) Relational Arc vs. Category Arc, determined by the kind of label (category arcs are atomic and have R-signs like Case, Number, Gender, etc.). The parser looks for arcs to complete that are **Surface, Structural and Relational (SSR)**.

A simplified version of the parsing algorithm is sketched below. It uses the predicates **Left-Precedence** , **Right-Precedence** and **Complete**:

1. **Precedence**: Let $Q_i = [n_i, L_i, R_i]$, F $\in$ SSR-Out-Arcs(n_i) such that *Target*(F) = *Anchor*(*Graph*(n_i)), and A $\in$ SSR-Out-Arcs(n_i) be an incomplete terminal arc. Then:
   A. **Left-Precedence**(A, n_i) is true iff:
      a. All surface arcs which must follow F are incomplete.
      b. A can precede F.
      c. All surface arcs which must both precede F and follow A are complete.
   B. **Right-Precedence**(A, n_i) is true iff:
      a. All surface arcs which must precede F are complete.
      b. A can follow F.
      c. All surface arcs which must both follow F and precede A are complete.

2. **Complete** : A node is complete if it is either a lexical anchor or else has (obligatory) outgoing SSR arcs, all of which are complete. An arc is complete if its target is complete.

The algorithm is **head-driven** [8] and was inspired by parsing algorithms for lexicalized TAGs ([6], [10]).

Simplified Parsing Algorithm:

Input: A string of words $w_1, \ldots, w_k$.

Output: A chart containing all possible parses.

Method:

A. **Initialization:**

1. Create a list of k state-sets $S_1, \ldots, S_k$, each empty.
2. For $\mathbf{c} = 1, \ldots, k$, for each $Graph(n_i)$ of $w_\mathbf{c}$, add $[n_i, \mathbf{c}-1, \mathbf{c}]$ to $S_\mathbf{c}$.

B. **Completions:**

For $\mathbf{c} = 1, \ldots, k$, do repeatedly until no more states can be added to $S_\mathbf{c}$:

1. **Leftward Completion:**

For all

$Q_i = [n_i, L_i, \mathbf{c}] \in S_\mathbf{c}$,

$Q_j = [n_j, L_j, L_i] \in S_{L_i}$, such that *Complete*$(n_j)$ and

$A \in$ SSR-Out-Arcs(n_i), such that *Left-Precedence*(A, n_i)

IF *Unify-at-end-of-Path*$(n_i, n_j, A) \simeq n'_i$,

THEN Add $[n'_i, L_j, \mathbf{c}]$ to $S_\mathbf{c}$.

2. **Rightward Completion:**

For all

$Q_i = [n_i, L_i, R_i] \in S_{R_i}$,

$Q_j = [n_j, R_i, \mathbf{c}] \in S_\mathbf{c}$ such that *Complete*(n_j), and

$A \in$ SSR-Out-Arcs(n_i), such that *Right-Precedence*(A, n_i)

IF *Unify-at-end-of-Path*$(n_i, n_j, A) \simeq n'_i$,

THEN Add $[n'_i, L_i, \mathbf{c}]$ to $S_\mathbf{c}$.

To illustrate, we step through the chart for *John seemed ill* (cf. Figure 7). In the string *0 John 1 seemed 2 ill 3*, where the integers represent string positions, each word w is associated via the lexicalized grammar with a finite set of anchored S-graphs. For expository convenience, we will assume counterfactually that for each w there is only one S-graph G_w with root r_w and anchor w. Also in the simplified case, we assume that the anchor is always the target of an arc whose source is the root. This is true in our example, but false in general.

For each G_w, r_w has one or more outgoing SSR arcs, the set of which we denote SSR-Out-Arcs(r_w). For each w between integers x and y in the string, the Initialization step (step A of the algorithm) adds [n_w, x, y] to state set y. We denote state Q in state-set S_i as state i:Q. For an input string $\omega = w_1, \ldots, w_n$, initialization creates n state-sets and for $1 \leq i \leq n$, adds states $i : Q_j, 1 \leq j \leq k$, to S_i , one for each of the k S-graphs G_{w_i} associated with w_i. After initialization, the example chart consists of states 1:1, 2:1, 3:1.

Then the parser traverses the chart from left to right starting with state-set 1 (step B of the algorithm), using left and right completions, according to whether left or right precedence conditions are used. Each completion looks in a state-set to the *left* of S_c for a state meeting a set of conditions. In the example, for c = 1, step B of the algorithm does not find any states in any state-set preceding S_1 to test, so the parser advances c to 2. A left completion succeeds with Q_i = state 2:1 = $[n_i, 1, 2]$ and Q_j = state 1:1 = $[n_j, 0, 1]$. State 2:2 = $[n'_i, 0, 2]$ is added to state-set S_2, where n'_i = Unify-at-end-of-Path$(n_i, n_j, [0, 1))$. Label $[0, 1)$ is closed off to yield $[0, 1]$ in the output graph, since no further R-signs may be added to the label once the arc bearing the label is complete.

The precedence constraints are interpreted as strict partial orders on the sets of outgoing SSR arcs of each node (in contrast to the totally ordered lexicalized TAGs). Arc [0, 1) satisfies left-precedence because: (i) [0, 1) is an incomplete terminal arc, where a **terminal** arc is an SSR arc, the target of which has no incomplete outgoing surface arcs; (ii) all surface arcs (here, only [C]) which must follow the [H] arc are incomplete; (iii) [0 1) can precede [H]; and (iv) there are no (incomplete) surface arcs which must occur *between* [0 1) and [H]. (We say *can* in (iii) because the parser accomodates variable word order.)

The parser precedes to state-set S_3. A right completion succeeds with Q'_i = state 2:2 = $[n'_i, 0, 2]$ and Q'_j = state 3:1 = $[n'_j, 2, 3]$. State 3:2 = $[n''_i, 0, 3]$ is added to state set S_3, n''_i = Unify-at-

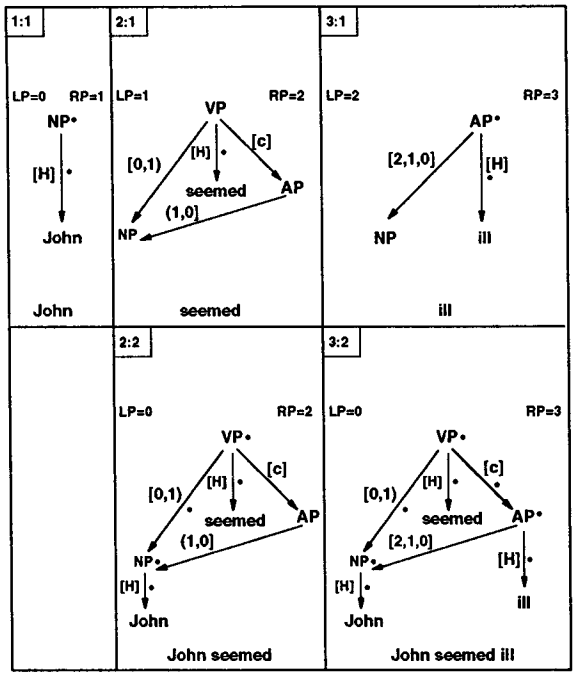

Figure 7: Chart for *John seemed ill.*

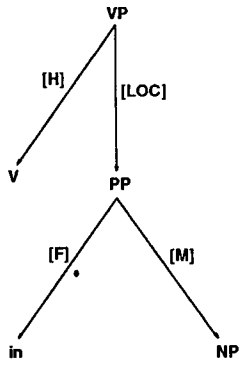

Figure 8: Example: *in*

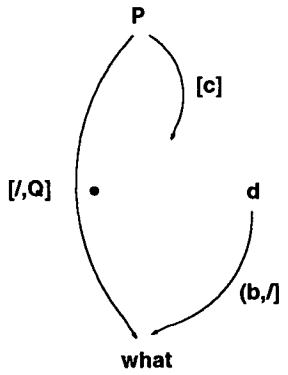

Figure 9: Example: *What*

end-of-Path(n_i', n_j', [C]). State 3:2 is a successful parse because n_i'' is complete and spans the entire input string.

To sum up: a completion finds a state $Q_i = [n_i, L_i, R_i]$ and a state $Q_j = [n_j, L_j, R_j]$ in adjacent state-sets ($L_i = R_j$ or $R_i = L_j$) such that n_i is *incomplete* and n_j is *complete*. Each successful completion completes an arc A $\in$ SSR-Out-Arcs(n_i) by unifying n_j with the target of A. Left completion operates on a state $Q_i = [n_i, L_i, c]$ in the current state-set S_c looking for a state $Q_j = [n_j, L_j, L_i]$ in state-set S_{L_i} to complete some arc A $\in$ SSR-Out-Arcs(n_i). Right completion is the same as left completion except that the roles of the two states are reversed: in both cases, success adds a new state to the current state-set S_c. The parser completes arcs first leftward from the anchor and then rightward from the anchor.

CONCLUDING REMARKS

The algorithm described above is simpler than the one we have implemented in a number of ways. We end by briefly mentioning some aspects of the general algorithm.

Optional Arcs: On encountering an optional arc, the parser considers two paths, skipping the optional arc on one and attempting to complete it on the other.

Constraint Arcs These are reminiscent of LFG constraint equations. For a parse to be good, each constraint arc must unify with a structural arc.

Multi-tiered S-graphs: These are S-graphs having a *non-terminal* incomplete arc I (e.g., the [LOC] arc in Figure 8. Essentially, the parser searches I depth-first for incomplete terminal arcs to complete.

Pseudo-R-signs: These are names of sets of R-signs. For a parse to be good, each pseudo-R-sign must unify with a member of the set it names.

Extractions: Our approach is novel: it uses pseudo-R-signs and **multirooted** S-graphs, illustrated in Figure 9, where p is the *primary* root and d, the *dangling* root, is the source of a "slashed arc" with label of the form (**b**, /] (**b** a pseudo-R-sign). Since well-formed final parses must be

single-rooted, slashed arcs must eventually unify with another arc.

To sum up: We have developed a unification-based, chart parser for relational grammars based on the SFG formalism presented by Johnson and Moss [2]. The system involves compiling (combinations) of rules graphs and their associated lexical anchors into a lexicalized grammar, which can then be parsed in the same spirit as lexicalized TAGs. Note, though, that SFG does not use an adjunction (or substitution) operation.

References

[1] Bob Carpenter. *The Logic of Typed Feature Structures.* Cambridge UP, Cambridge, 1992.

[2] David E. Johnson and Lawrence S. Moss. Some formal properties of stratified feature grammars. To appear in *Annals of Mathematics and Artificial Intelligence*, 1993.

[3] David E. Johnson and Paul M. Postal. *Arc Pair Grammar.* Princeton University Press, 1980.

[4] Ronald Kaplan and Joan Bresnan. Lexical-functional grammar, a formal system for grammatical representation. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations.* MIT Press, 1982.

[5] Robert Kasper and William C. Rounds. The logic of unification in grammar. *Linguistics and Philosophy*, 13:35–58, 1990.

[6] Alberto Lavelli and Giorgio Satta. Bidirectional parsing of lexicalized tree adjoining grammars. In *Proceedings of the 5th Conference of the European Chapter of the Association of Computational Linguistics*, 1991.

[7] Carl Pollard and Ivan Sag. *Information-based Syntax and Semantics.* CSLI Lecture Notes. University of Chicago Press, Chicago, 1987.

[8] Derek Proudian and Carl Pollard. Parsing head-driven phrase structure grammar. In *Proceedings of the 23rd Annual Meeting of the ACL*, 1985.

[9] William C. Rounds and Alexis Manaster-Ramer. A logical version of functional grammar. In *Proceedings of The 25th Annual Meeting of the Association for Computational Linguistics*, 1987.

[10] Yves Schabes. *Mathematical and Computational Properties of Lexicalized Grammars.* PhD thesis, University of Pennsylvania, 1990.

[11] Stuart Shieber. *Constraint-Based Grammar Formalisms.* MIT Press, 1992.