

# Extensions to Constraint Dependency Parsing for Spoken Language Processing

Mary P. Harper and Randall A. Helzerman  
School of Electrical Engineering  
Purdue University  
West Lafayette, IN 47907

February, 1995

## **Abstract**

A text-based and spoken language processing framework based on the Constraint Dependency Grammar (CDG) developed by Maruyama [24, 25] is discussed. The scope of CDG is expanded to allow for the analysis of sentences containing lexically ambiguous words, to allow feature analysis in constraints, and to efficiently process multiple sentence candidates that are likely to arise in spoken language processing. The benefits of the CDG parsing approach are summarized. Additionally, the development of CDG grammars using our grammar tools and parser is discussed.

# 1 Introduction

In this paper, we adapt the constraint dependency grammar (CDG) formalism introduced by Maruyama [24, 25, 26] to the problem of analyzing spoken language. Constraint dependency grammars are more expressive than context-free grammars (CFGs) and more tractable, but less expressive, than context-sensitive grammars, and they provide an extremely flexible framework for parsing natural language.

In Section 2 of this paper, the CDG parsing algorithm is described. In the remainder of the paper, the following three extensions to the CDG parsing algorithm are discussed:

1. Many words in the English language have lexical ambiguity (i.e., more than a single part of speech) which can cause correctness and efficiency problems for a parser [2]. A related problem is the processing of utterances with multiple word candidates over the same time interval. In Section 3.1, we extend the CDG parsing algorithm to handle lexical ambiguity and multiple word candidates in an integrated and efficient manner.
2. A word can also have ambiguity in the feature information associated with the word, like number, person, or case. For example, the noun *fish* can take a number/person value of third person singular or third person plural. Often feature information is useful for disambiguating parses for sentences or for eliminating impossible sentence hypotheses; hence, we have also added lexical feature analysis to the CDG parser, as described in Section 3.3.
3. Maruyama’s CDG parsing algorithm was designed to process single sentences. If a natural language processor is used in conjunction with a speech recognizer, the input to the natural language component would typically be a set of sentence hypotheses. However, processing all of the sentence hypotheses individually for an utterance provided by a speech recognizer is very inefficient since many hypotheses are similar. Furthermore, a list of sentence hypotheses is not the most compact representation to provide a natural language parser. A better representation is a graph of word candidates, which reduces the redundancy and compactly represents the set of sentence hypotheses. In Section 4, we extend CDG to operate on a word graph rather than single sentences. We also describe the implementation of our word graph constraint-based parser, PARSEC (Parallel ARCHitecture Sentence Constrainer), and the development of constraint-based grammars using PARSEC grammar development tools.

## 2 A Description of Maruyama’s CDG Parsing

We begin with the definition of Constraint Dependency Grammar (CDG), discuss Maruyama’s CDG parsing algorithm, and then relate the algorithm to other approaches.

## 2.1 Elements of CDG

Maruyama introduced CDG in [24, 25]. A sentence  $s = w_1w_2w_3\dots w_n$  is a string of finite length  $n$  and is an element of  $\Sigma^*$ , where  $\Sigma$  is a finite set of preterminal symbols or lexical categories.  $R = \{r_1, \dots, r_p\}$  is defined to be a finite set of *role-ids*. Each word  $i$  in a sentence  $s$  has  $p$  different roles,  $r_1(i), r_2(i), \dots, r_p(i)$ . Hence, every sentence contains  $n * p$  roles. A role can be thought of as a variable which is assigned a role value. A role value is a tuple  $\langle l, m \rangle$ , where  $l$  is an element from a finite set of labels  $L = \{l_1, \dots, l_q\}$ , and  $m$  is an element of the set of modifiees,  $\{1, 2, \dots, n, \text{nil}\}$ . A modifiee is a pointer to another word in the sentence (or to no word if nil). Role values will be denoted in the examples as *label-modifiee*. In CDG, an assignment  $\mathbf{A}$  for the sentence  $s$  is a function that assigns role values to roles given a set of constraints  $C$ .

A *constraint set* is a logical formula of the form<sup>1</sup>: (and  $P_1 P_2 \dots P_t$ ), where the definitions for the possible components of a subformula  $P_i$  are shown below:

- **Variables:**  $x_1, x_2, \dots, x_a$ .
- **Constants:** elements and subsets of  $\Sigma \cup L \cup R \cup \{\text{nil}, 1, 2, \dots, n\}$ , where  $n$  corresponds to the number of words in a sentence.
- **Access Functions:**
  - (**pos**  $x$ ) returns the position of the word for the role value  $x$ .
  - (**rid**  $x$ ) returns the role-id for the role value  $x$ .
  - (**lab**  $x$ ) returns the label for the role value  $x$ .
  - (**mod**  $x$ ) returns the position of the modifiee for the role value  $x$ .
  - (**cat**  $i$ ) returns the category (i.e., the element in  $\Sigma$ ) for the word <sup>2</sup> in position  $i$ .
- **Predicate symbols:**
  - (**eq**  $x$   $y$ ) returns true if  $x = y$ , false otherwise.
  - (**gt**  $x$   $y$ ) returns true if  $x > y$  and  $x, y \in \text{Integers}$ , false otherwise<sup>3</sup>.
  - (**lt**  $x$   $y$ ) returns true if  $x < y$  and  $x, y \in \text{Integers}$ , false otherwise.
  - (**elt**  $x$   $y$ ) returns true if  $x \in y$ , false otherwise.
- **Logical Connectives:**
  - (**and**  $p$   $q$ ) returns true if  $p$  and  $q$  are true, false otherwise.
  - (**or**  $p$   $q$ ) returns true if  $p$  or  $q$  is true, false otherwise.
  - (**not**  $p$ ) returns true if  $p$  is false, false otherwise.

---

<sup>1</sup>Maruyama uses an infix notation; whereas, we use a prefix notation throughout this paper.

<sup>2</sup>Maruyama uses the access function **word** rather than **cat**, though the function accesses the category of the word.

<sup>3</sup>For example, (gt 1 nil) is false because nil is not an integer.

Each  $P_i$  in  $C$  must be of the form (if *Antecedent Consequent*), where *Antecedent* and *Consequent* are predicates or predicates joined by the logical connectives. A subformula  $P_i$  is called a *unary constraint* if it contains one variable and a *binary constraint* if it contains two.

Hence, Maruyama defines a CDG as a 4-tuple,  $\langle \Sigma, R, L, C \rangle$ , where:

$\Sigma$  = a finite set of preterminal symbols or lexical categories.  
 $R$  = a finite set of uniquely named roles (or role-ids) =  $\{r_1, \dots, r_p\}$ .  
 $L$  = a finite set of labels =  $\{l_1, \dots, l_q\}$ .  
 $C$  = a constraint set that an assignment  $\mathbf{A}$  must satisfy.

A sentence  $s$  is said to be generated by a CDG grammar  $G$  if there exists an assignment  $\mathbf{A}$  such that the constraint set  $C$  is satisfied. There may be more than one assignment which satisfies  $C$ , in which case there is more than one parse for the sentence.  $L(G)$  is the language generated by grammar  $G$  if and only if  $L(G)$  is the set of all sentences generated by  $G$ . Note that the null string  $\epsilon$  has no roles and is always generated by any grammar according to definition.

A CDG grammar has two associated parameters, *degree* and *arity*. The degree of a grammar  $G$  is the size of  $R$ . The arity of the grammar corresponds to the maximum number of variables in the subformulas of  $C$ . To simplify the examples in this paper, we use grammars with a degree of one, that is, with a single role *governor*. The governor role indicates the function a word fills in a sentence when it is governed by its head word. In our implemented grammars (described in Section 4.4), we also use several needs roles (e.g, need1, need2) to make certain that a head word has all of the constituents it needs to be complete (e.g., a singular count noun needs a determiner to be a complete noun phrase). Maruyama has proven that a grammar with a degree and arity of two is required to be as expressive as a CFG.

To illustrate the use of CDG grammars, consider a very simple example grammar,  $G_1 = \langle \Sigma_1, R_1, L_1, C_1 \rangle$  in Figure 1, which has a degree of one and an arity of two<sup>4</sup>. Note that U-1, U-2, and U-3 are unary constraints because they contain a single variable, and B-1 is a binary constraint because it contains two variables.

For  $G_1$  to generate the sentence *The program runs*, there must be an assignment of a role value to the governor role of each word, and that assignment must simultaneously satisfy each of the subformulas in  $C_1$ . Note that each word is assumed to have a single lexical category, which is determined by dictionary lookup. Table 1 depicts an assignment for the sentence which satisfies  $C_1$ . This assignment can be interpreted as the parse graph shown in Figure 10.

---

<sup>4</sup>The constraints in this grammar were chosen for simplicity, not to exemplify constraints for a wide coverage grammar.

```

Σ1 = {det, noun, verb}
R1 = {governor}
L1 = {det, root, subj}
C1 = ∀ x y (and
  ;; [U-1] A det receives the label det and modifies a word to its right.
  (if (eq (cat (pos x)) det)
      (and (eq (lab x) det)
            (lt (pos x) (mod x))))
  ;; [U-2] A noun receives the label subj and modifies a word to its right.
  (if (eq (cat (pos x)) noun)
      (and (eq (lab x) subj)
            (lt (pos x) (mod x))))
  ;; [U-3] A verb receives the label root and modifies no word.
  (if (eq (cat (pos x)) verb)
      (and (eq (lab x) root)
            (eq (mod x) nil)))
  ;; [B-1] A det is governed by a noun.
  (if (and (eq (lab x) det)
           (eq (mod x) (pos y)))
      (eq (cat (pos y)) noun)))

```

Figure 1:  $G_1 = \langle \Sigma_1, R_1, L_1, C_1 \rangle$ .

pos	word	cat	governor role's value
1	the	det	det-2
2	program	noun	subj-3
3	runs	verb	root-nil

Table 1: An assignment for *The program runs*.

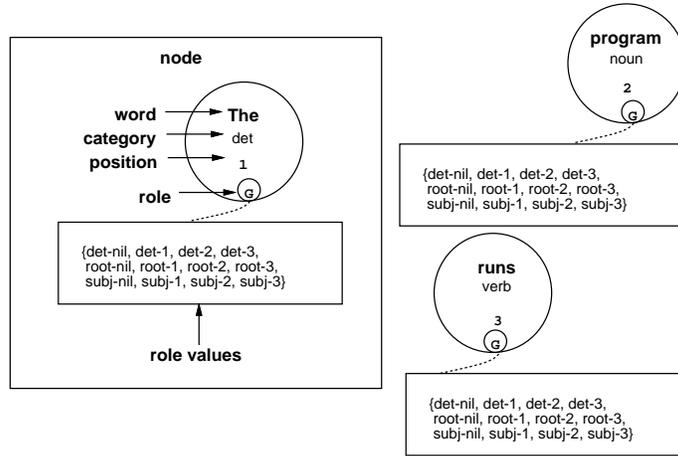


Figure 2: Initialization of roles for the sentence *The program runs*.

## 2.2 CDG Parsing

To determine whether a sentence is generated by a grammar, a CDG parser must be able to assign at least one role value which satisfies the grammar constraints to each of the  $n * p$  roles, where  $n$  is sentence length, and  $p$  is the number of role-ids. Because the role values for the role are selected from the finite set  $L_1 \times \{\text{nil}, 1, 2, \dots, n\}$ , CDG parsing can be viewed as a constraint satisfaction problem over a finite domain. Hence, constraint propagation [19, 29, 50] can be used to develop the parse of a sentence. Enumeration of the individual parses for a highly ambiguous sentence is intractable. Therefore, a CDG parser generates all parses for a sentence in a compact form. The steps required for parsing the sentence *The program runs* are provided to illustrate both the process of parsing with constraint propagation and the running time of the algorithm.

To develop a syntactic analysis for a sentence using CDG, a *constraint network* (CN) of words is created. Each of the  $n$  words in a sentence is represented as a node in a CN. Figure 2 illustrates the initial configuration of nodes in the CN for *The program runs* example. Notice that associated with each node is its word, category, sentence position, and roles (only one for this example). Each of the roles is initialized to the set of all possible role values (i.e., the domain). Given  $G_1$ , the domain for the example is  $L_1 \times \{\text{nil}, 1, 2, 3\} = \{\text{det-nil}, \text{det-1}, \text{det-2}, \text{det-3}, \text{root-nil}, \text{root-1}, \text{root-2}, \text{root-3}, \text{subj-nil}, \text{subj-1}, \text{subj-2}, \text{subj-3}\}$ . Since there are  $q * (n + 1) = O(n)$  possible role values for each of the  $n * p$  roles for a sentence (where  $p$ , the number of roles per word, and  $q$ , the number of different labels, are grammatical constants, and  $n$  is the number of words in the sentence), there are  $n * p * q * (n + 1) = O(n^2)$  role values which must be initially generated for the CN, requiring  $O(n^2)$  time. Note that each role value has direct access to its role-id, label, modifiee, and the position of its word. A role value must access the word node to determine its lexical category.

To parse the sentence using  $G_1$ , the unary and binary constraints in  $C_1$  are applied to the CN

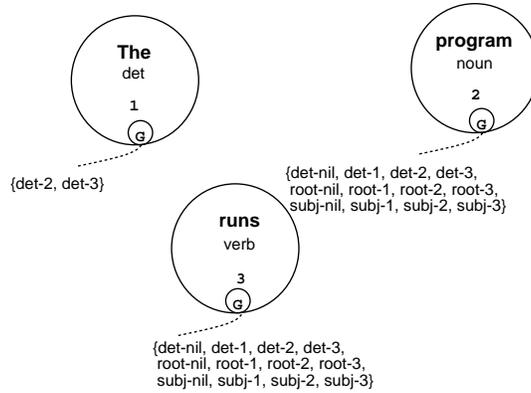


Figure 3: The CN after the propagation of U-1 for the sentence *The program runs*.

to eliminate the role values from the roles of each word which are incompatible with  $C_1$ . For a sentence to be grammatical, each role in each word node must contain at least one role value after constraint propagation.

The unary constraints are applied to each of the roles in the sentence to eliminate the role values incompatible with each word’s role in isolation. To apply the first unary constraint (i.e., U-1, shown below) to the network in Figure 2, each role value for every role is examined to ensure that it obeys the constraint.

```

;; [U-1] A det receives the label det
;; and modifies a word to its right.
(if (eq (cat (pos x)) det)
    (and (eq (lab x) det)
         (lt (pos x) (mod x))))

```

If a role value causes the antecedent of the constraint to evaluate to TRUE and the consequent to evaluate to FALSE, then that role value is eliminated. Figure 3 shows the remaining role values after U-1 has been applied to the CN in Figure 2.

Maruyama uses only atomic subformulas in a constraint set which can be evaluated in constant time. Because of this restriction, each constraint can only contain access functions and predicates that operate in constant time (e.g., access functions and predicates like those defined in Section 2.1). So when the unary constraint U-1 is applied to  $O(n^2)$  role values, it requires  $O(n^2)$  time.

To further eliminate role values which are incompatible with the word categories in the example, the remaining unary constraints (i.e., U-2 and U-3) are applied to the CN in Figure 3, producing the network in Figure 4. Given that the time to apply the unary constraints to a single role value is a grammatical constant denoted as  $k_u$ , the time required to apply the unary constraints in a grammar to all role values is  $O(k_u * n^2)$ .

The binary constraints determine which pairs of role values can legally coexist. To keep track of pairs of role values, *arcs* connect each role to all other roles in the network, and each arc has an

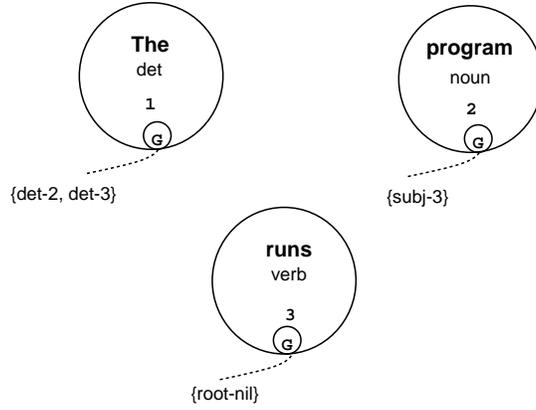


Figure 4: The CN after the propagation of all the unary constraints.

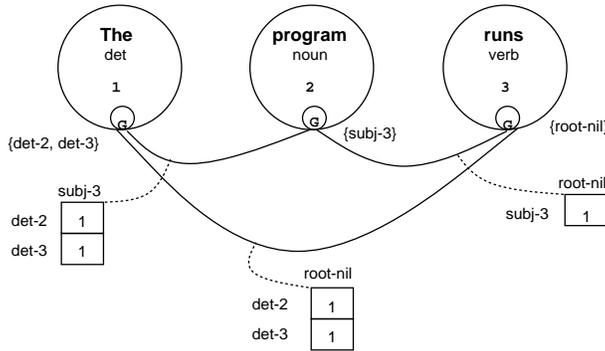


Figure 5: The CN after unary constraint propagation and before binary constraint propagation.

associated *arc matrix*, whose row and column indices are the role values associated with the two roles. The elements of an arc matrix can either be a **1** (indicating that the two role values which index the element are compatible) or a **0** (indicating that the role values cannot simultaneously exist). Initially, all elements in each matrix are set to **1**, indicating that the two role values are initially compatible. Since there are  $\binom{n+p}{2} = O(n^2)$  arcs required in the CN, and each arc contains a matrix with  $(q*(n+1))^2 = O(n^2)$  elements, the time to construct the arcs and initialize the matrices is  $O(n^4)$ . Figure 5 shows the matrices associated with the arcs before any binary constraints are propagated. Unary constraints are usually propagated before preparing the CN for binary constraints because they eliminate impossible role values, and hence reduce the dimensions of the arc matrices.

Binary constraints are applied to the pairs of role values indexing each of the arc matrix elements. When a binary constraint is violated by a pair of role values, the entry in the matrix indexed by those role values is set to zero. The binary constraint, B-1, ensures that a det is governed by a noun.

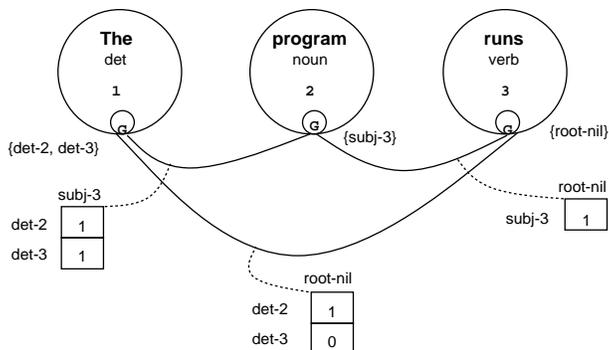


Figure 6: The CN after B-1 is propagated.

```

;; [B-1] A det is governed by a noun.
(if (and (eq (lab x) det)
         (eq (mod x) (pos y)))
    (eq (cat (pos y)) noun))

```

After the application of this constraint to the network in Figure 5, the element indexed by the role values<sup>5</sup>  $x=\text{det-3}$  and  $y=\text{root-nil}$  for the matrix on the arc connecting the governor roles for *the* and *runs* is set to zero, as shown in Figure 6. This is because *the* must be governed by a noun, not a verb. Since the constraint must be applied to  $O(n^4)$  pairs of role values, the time to apply the constraint is  $O(n^4)$ . Given that the time to apply the binary constraints to a pair of role values is a grammatical constant denoted as  $k_b$ , the time required to apply the binary constraints in a grammar to all pairs of role values is  $O(k_b * n^4)$ .

Following the propagation of binary constraints, the roles of the CN could still contain role values which are incompatible with the parse for the sentence. To determine whether a role value is still supported for a role, each of the matrices on the arcs incident to the role must be checked to ensure that the row (or column) indexed by the role value contains at least a single **1**. If any arc matrix contains a row (or column) of **0**s, then the corresponding role value cannot coexist with any of the role values for the second role and so is removed from the list of legal role values for the first role. Additionally, the rows (or columns) associated with the eliminated role value can be removed from the arc matrices attached to the role. The process of removing any rows or columns containing all zeros from arc matrices and eliminating the associated role values from their roles is called *filtering*. Following binary constraint propagation, any of the  $O(n^2)$  role values may be filtered immediately. However, filtering must also be applied iteratively since the elimination of one role value could lead to the elimination of another role value.

Filtering a constraint network is known as arc consistency to constraint satisfaction researchers. An optimal version of the algorithm, AC-4, was developed by Mohr and Henderson [28]. AC-4

<sup>5</sup>It might be useful to think of a particular instance of  $\text{det-3}$  assigned to a role as a name for a structure containing its role id, label, modifiee, and position.

Notation	Meaning
$(i, j)$	An ordered pair of roles.
$N$	$\{i, j, \dots\}$ is the set of all roles, with $ N  = n * p$ .
$L$	$\{a, b, \dots\}$ is the set of <i>role values</i> , with $ L  = q * (n + 1)$ .
$L_i$	$\{a   a \in L \text{ and } (i, a) \text{ is permitted by the constraints (i.e., admissible)}\}$
$R\mathcal{2}(i, a, j, b)$	$R\mathcal{2}(i, a, j, b) = 1$ indicates the admissibility $a \in L_i$ and $b \in L_j$ after binary constraint propagation. This corresponds to the element indexed by $[a, b]$ in the matrix for arc $(i, j)$ in our algorithm.
$(i, a)$	An ordered pair of role $i$ and role value $a \in L_i$ .
$M[i, a]$	$M[i, a] = 1$ indicates that the role value $a$ is not admissible for (and has already been eliminated from) role $i$ .
$E$	All role pairs $(i, j)$ . If $(i, j) \in E$ , then $(j, i) \in E$ .
$S[i, a]$	$(j, b) \in S[i, a]$ means that role value $a \in L_i$ and $b \in L_j$ are simultaneously admissible. This implies that $a$ supports $b$ .
Counter $[(i, j), a]$	The number of role values in $L_j$ which are compatible with $a \in L_i$ .
List	A queue of arc support to be deleted.

Table 2: Data structures and notation for the CN arc consistency algorithm (i.e., AC-4).

builds and maintains several data structures, described in Table 2, to allow it to efficiently perform this operation. Figure 7 shows the code for initializing the data structures, and Figure 8 contains the algorithm for eliminating inconsistent role values from the domains. This filtering algorithm requires  $O(el^2)$ , where  $e$  is the number of arcs, and  $l$  is the size of the domain [28]. In the case of CDG parsing,  $e = \binom{n * p}{2}$ , and the domain size is  $n * q$ , so the running time of the filtering step is  $O(n^4)$  [24, 25].

If the role value  $a$  is a member of role  $i$ 's domain, that is  $a \in L_i$ , and is compatible with  $b \in L_j$ , then  $a$  *supports*  $b$  (and vice versa). To keep track of how much support each role value  $a$  has, the number of role values in  $L_j$  which are compatible with  $a$  in  $L_i$  are counted, and the total is stored in Counter $[(i, j), a]$ . The algorithm also keeps track of which role values that role value  $a$  supports by using  $S[i, a]$ , which is a set of arc and role value pairs. For example,  $S[i, a] = \{(j, b), (j, c)\}$  means that  $a$  in  $L_i$  supports  $b$  and  $c$  in  $L_j$ . If  $a$  is ever removed from  $L_i$ , then  $b$  and  $c$  will lose some of their support. This is accomplished by decrementing Counter $[(j, i), b]$  and Counter $[(j, i), c]$ . For arc consistency, if Counter $[(i, j), a]$  becomes zero,  $a$  is automatically removed from  $L_i$ , because that would mean that  $a$  is impossible in any sentence parse. When a role value  $a \in L_i$  is found to be unsupported, the algorithm places the ordered pair  $(i, a)$  on *List*. When  $(i, a)$  is popped off *List* by the procedure in Figure 8, more role values may lose support and be placed on *List*.

```

1. List:= $\phi$ ;
2. for  $i \in N$  do
3.   for  $a \in L_i$  do
4.     begin
5.        $M[i, a] := 0$ ;
6.        $S[i, a] := \phi$ ;
7.     end
8.   for  $(i, j) \in E$  do
9.     for  $a \in L_i$  do
10.      begin
11.        Total := 0;
12.        for  $b \in L_j$  do
13.          if  $R\mathcal{L}(i, a, j, b)$  then
14.            begin
15.              Total := Total+1;
16.               $S[j, b] := S[j, b] + \{(i, a)\}$ ;
17.            end
18.          if Total=0 then
19.            begin
20.               $M[i, a] := 1$ ;
21.              List := List  $\cup \{(i, a)\}$ ;
22.               $L_i := L_i - \{a\}$ ;
23.            end
24.          Counter $[(i, j), a] :=$ Total;
25.        end

```

Figure 7: Construction of data structures for CN arc consistency (i.e., AC-4).

Consider how filtering is applied to the CN in Figure 6. The matrix associated with the arc connecting the governor roles of *the* and *runs* contains a row with a single element which is a zero. Because det-3 cannot coexist with the only possible role value for the governor role of *runs*, it cannot be a legal member of the governor role of *the*, and so (1, det-3) is placed on  $List^6$ , and det-3 is eliminated as a role value for node 1's governor role. When the role value is eliminated from all arcs associated with the role, filtering is complete. The resulting CN is depicted in Figure 9.

After all the constraints are propagated across the CN and filtering is completed, the CN provides a compact representation for all possible parses. Syntactic ambiguity is easy to spot in the CN since some of the roles in an ambiguous sentence contain more than a single role value. If multiple parses exist, we can propagate additional constraints to further refine the analysis of the ambiguous sentence, or we could just enumerate the parses contained in the CN by using backtracking search. For highly ambiguous grammars, the process of enumerating all possible parses is intractable, making incremental disambiguation a more attractive option. The parse trees in a CN are precedence graphs, which we call *parse graphs*, and they consist of a compatible set of role values for each of the roles in the CN. The modifiers of the role values, which point to the words they modify, form the edges of the parse graph. Our example sentence has an unambiguous parse graph given  $G_1$ , shown in Figure 10.

---

<sup>6</sup>For this example, we use the node number to denote the role number since there is only a single role. In the case where there are two or more roles per node, each role would need a unique number to identify it for filtering.

```

1. while List not empty do
2.   begin
3.     pop (j, b) from List;
4.     for (i, a) ∈ S[j, b] do
5.       begin
6.         Counter[(i, j), a] := Counter[(i, j), a] - 1;
7.         if Counter[(i, j), a] = 0 ∧ M[i, a] = 0 then
8.           begin
9.             List := List ∪ {(i, a)};
10.            M[i, a] := 1;
11.            Li := Li - {a}
12.          end
13.        end
14.      end

```

Figure 8: Algorithm to enforce CN arc consistency (i.e., AC-4).

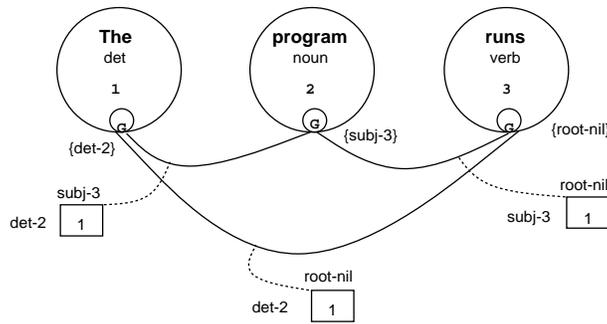


Figure 9: The CN after filtering.

Below we list the steps in the CDG parsing algorithm and their associated running times:

1. Constraint network construction prior to unary constraint propagation:  $O(n^2)$
2. Unary constraint propagation:  $O(k_u * n^2)$
3. Constraint network construction prior to binary constraint propagation:  $O(n^4)$
4. Binary constraint propagation:  $O(k_b * n^4)$
5. Filtering (arc consistency):  $O(n^4)$

Notice that the time required to propagate binary constraints is the slowest part of the algorithm.

### 2.3 CDG Parsing Compared with Other Parsing Approaches

Researchers have developed and used a variety of parsing paradigms which are based on context-free grammars (CFGs) or are equivalent in expressivity to CFGs. A complete survey of CFG parsers is beyond the scope of this paper; however, a comparison of the CDG parser with several representative CFG parsers will illustrate their similarities and differences.

Most of the CFG parsers that are used in computer language applications (e.g., LL and LR parsers) compile grammar rules into a table which is then used by a parser driver to deterministically parse programs. Though these parsers have been used by the natural language processing

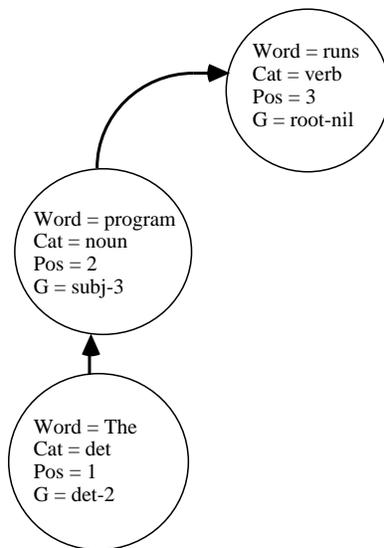


Figure 10: The parse graph for the CN in Figure 11.

community, the ambiguity that is common in natural language has caused researchers to seek alternative methods. A recursive transition network parser [53], which encodes a grammar in a network and then searches the network in a top-down fashion, is able to produce all possible parses for an ambiguous sentence. However, if the parser does not cache subresults during that search it can have an exponential running time. Chart-based parsers [17] use a chart to record how various portions of an input string can match the categorial sequences specified by different rules. A parse-forest can be created for a chart by indicating combinations of active and inactive edges that the constituent could come from. A chart parser using only phrase structure constraints is guaranteed to operate in polynomial time with polynomial space. Definite Clause Grammar parsers [33] use unification and search to determine the parses for a sentence where the algorithm can be organized in a variety of ways including as a chart parser. Researchers have developed a wide variety of CFG parsing algorithms in an attempt to more efficiently process languages [46, 3, 38, 6]. The best serial running time for a CFG parser operating with an ambiguous grammar is  $O(G * n^3)$ , where  $n$  is the number of words in a sentence and  $G$  is the size of the CFG grammar; however, there are useful parsers that exceed this time bound [47, 38].

Below we enumerate the differences between CFG parsers and the CDG parser:

1. CFG parsers use production rules (or equivalently, recursive transition networks) in a grammar to determine whether or not a string of terminals is in the language. The CDG parser, on the other hand, uses sets of constraints. Any type of constraint that can be formulated as an if-then rule containing one or two role value variables can be used to constrain a CN.
2. The best serial running time for a CFG parser operating with an ambiguous grammar is

$O(G * n^3)$ , where  $n$  is the number of words in a sentence and  $G$  is the size of the CFG grammar. On the other hand, the serial running time for the CDG parser of single sentences is  $O(k * n^4)$ , where  $k$  is the number of constraints in grammar. In practice, we have found that  $k$  is comparable in size to  $G$  for grammars with the same coverage.

3. CFG parsers often construct a parse forest (or similar structure) to avoid enumerating parse trees. The CN constructed by the CDG parser is a forest of parse graphs which is pruned during parsing. A CN differs from a CFG parse forest in that there are no non-terminals in the graph, only links between terminals, which are assigned sets of labels. A packed parse forest generated by a CFG parsing algorithm can be mapped to a syntactic graph [44], which like a CN, compactly encodes all parses of ambiguous sentences by using modifier links between a head word and its modifiers. Exclusion matrices are associated with the syntactic graph, which, like CDG arc matrices, prevent impossible parses from being selected from the graph. However, unlike a CN, a syntactic graph is generated by a grammar which is context-free.
4. The smallest known size for a CFG parse forest is  $O(G * n^2)$  [3, 38], regardless of how many parses there are for an ambiguous sentence. In contrast, a CN has the size of  $O(n^4)$ .
5. When a CFG parser generates a set of ambiguous parses for a sentence, it cannot invoke additional production rules to further prune the analyses. In contrast, in CDG parsing, the presence of ambiguity can trigger the propagation of additional constraints to further refine the parse for a sentence. A core set of constraints that hold universally can be propagated first, and then if ambiguity remains, additional, possibly context dependent, constraints can be used. This type of flexibility is easy to achieve with CDG. Furthermore, it is possible to withhold constraints that are not applicable to certain types of speakers. For example, it is not uncommon for non-native speakers of English to drop determiners out of noun phrases. We can easily develop a control scheme to handle different types of dialectical variations in the language. We could use similar strategies to handle classes of disfluencies.
6. CFG parsing has been parallelized by several researchers. For example, Kosaraju’s method [18] using cellular automata can parse CFGs in  $O(n)$  time using  $O(n^2)$  processors. However, achieving CFG parsing times of less than  $O(n)$  has required more powerful and less implementable models of parallel computation, as well as significantly more processors. Ruzzo’s method [36] has a running time of  $O(\log^2(n))$  using a CREW P-RAM model (Concurrent Read, Exclusive Write, Parallel Random Access Machine), but requires  $O(n^6)$  processors to achieve that time bound. In contrast, we have devised a parallelization for the single sentence CDG parser [12, 11] which uses  $O(n^4)$  processors to parse in  $O(k)$  time for a CRCW P-RAM

pos	word	governor role's value
1	a	A-7
2	a	A-8
3	a	A-9
4	b	B-1
5	b	B-2
6	b	B-3
7	c	C-4
8	c	C-5
9	c	C-6

Table 3: An assignment for *aaabbbccc*.

model (Concurrent Read, Concurrent Write, Parallel Random Access Machine), where  $n$  is the number of words in the sentence and  $k$ , the number of constraints, is a grammatical constant. Furthermore, this algorithm has been simulated on the MasPar MP-1<sup>7</sup>, using the special features of the machine and  $O(n^4)$  processors to obtain an  $O(k + \log(n))$  running time.

7. To parse a free-order language like Latin, CFGs require that additional rules containing the permutations of the right-hand side of a production be explicitly included in the grammar [30]. Unordered CFGs do not have this combinatorial explosion of rules, though the universal recognition problem for this class of grammars is NP-complete for an  $n$ -character alphabet. A free-order language can be handled by a CDG parser without enumerating all permutations because order among constituents is not a requirement of the grammatical formalism.
8. The set of languages accepted by a CDG grammar is a superset of the set of languages which can be accepted by CFGs. In fact, Maruyama [24, 25] is able to construct CDG grammars with two roles (degree = 2) and two variable constraints (arity = 2) which accept the same language as an arbitrary CFG converted to Griebach Normal form. We have also devised an algorithm to map a set of CFG production rules into a CDG grammar. This algorithm does not assume that the rules are in normal form, and the number of constraints created is  $O(G)$ . In addition, CDG can accept languages that CFGs cannot, for example,  $a^n b^n c^n$  and  $ww$ , (where  $w$  is some string of terminal symbols). To illustrate the ease of writing such grammars, we have created a grammar which accepts the language  $a^n b^n c^n$ ,  $n \geq 0$ , shown in Figure 11. This grammar determines whether a string is acceptable by ensuring that there is a one-to-one correspondence between each  $b$  and  $a$ , each  $c$  and  $b$ , and each  $a$  and  $c$ , and that all  $a$ 's occur before all  $b$ 's and all  $b$ 's occur before all  $c$ 's. An assignment for the string *aaabbbccc* given  $G_2$  is shown in Table 3.

---

<sup>7</sup>The MasPar MP-1 is a massively parallel SIMD computer, which supports up to 16K 4-bit processing elements, each with 16KB of local memory.

$\Sigma_2 = \{a, b, c\}$ .  
 $R_2 = \{\text{governor}\}$   
 $L_2 = \{A, B, C\}$   
 $C_2 = \forall x y \text{ (and$

```

;;; [U-1] An a receives the label A and modifies an item to its right.
(if (eq (cat (pos x)) a)
    (and (eq (lab x) A)
         (gt (mod x) (pos x))))
;;; [U-2] A b receives the label B and modifies an item to its left.
(if (eq (cat (pos x)) b)
    (and (eq (lab x) B)
         (lt (mod x) (pos x))))
;;; [U-3] A c receives the label C and modifies an item to its left.
(if (eq (cat (pos x)) c)
    (and (eq (lab x) C)
         (lt (mod x) (pos x))))
;;; [B-1] Every A precedes every B.
(if (and (eq (lab x) A)
         (eq (lab y) B))
    (lt (pos x) (pos y)))
;;; [B-2] Every B precedes every C.
(if (and (eq (lab x) B)
         (eq (lab y) C))
    (lt (pos x) (pos y)))
;;; [B-3] If an A occurs after another A, then it must
;;; modify something after that A's modifiee.
(if (and (eq (lab x) A)
         (eq (lab y) A)
         (gt (pos x) (pos y)))
    (gt (mod x) (mod y)))
;;; [B-4] An A must modify a C.
(if (and (eq (lab x) A)
         (eq (mod x) (pos y))
         (eq (rid y) governor))
    (eq (lab y) C))
;;; [B-5] If a B occurs after another B, then it must
;;; modify something after that B's modifiee.
(if (and (eq (lab x) B)
         (eq (lab y) B)
         (gt (pos x) (pos y)))
    (gt (mod x) (mod y)))
;;; [B-6] A B must modify an A.
(if (and (eq (lab x) B)
         (eq (mod x) (pos y))
         (eq (rid y) governor))
    (eq (lab y) A))
;;; [B-7] If a C occurs after another C, then it must
;;; modify something after that C's modifiee.
(if (and (eq (lab x) C)
         (eq (lab y) C)
         (gt (pos x) (pos y)))
    (gt (mod x) (mod y)))
;;; [B-8] A C must modify a B.
(if (and (eq (lab x) C)
         (eq (mod x) (pos y))
         (eq (rid y) governor))
    (eq (lab y) B))

```

Figure 11:  $G_2 = \langle \Sigma_2, R_2, L_2, C_2 \rangle$  accepts the language  $a^n b^n c^n$ ,  $n \geq 0$ .

Constraint-based parsing has received considerable interest in computational linguistics in recent years. For example, Covington [1] outlines a constraint-based parser that uses dependency rules to set up modifiee links between terminals, as in CDG. Covington’s parser differs from CDG in that it uses search and unification to provide dependency graphs for sentences, while CDG uses constraint propagation. However, because both approaches use rules limiting dependency links between terminals, the dependency rules of Covington’s parser should easily map into CDG constraints. Additionally, both share a capability for handling free-order languages. Shieber [45] develops a constraint-based approach to parsing which also uses unification. Shieber’s rules consist of two parts, a CFG phrase-structure portion and a feature analysis portion. The strength of Shieber’s approach is in his well-defined semantics of feature constraints. CDG differs from Shieber’s approach in several ways. First, in CDG, all rules are specified as constraints; there is no explicit separation between phrase-structure rules and other types of constraints. Second, CDG is not limited to the use of CFG phrase structure rules.

There has also been considerable interest in the development of parsers for grammars that are more expressive than the class of context-free grammars, but less expressive than context-sensitive grammars [16, 48, 49]. The parser for tree adjoining grammars (TAG) has a running time<sup>8</sup> of  $O(n^6)$ . CDG is a weaker formalism than TAG; however, its lower order running time could make it more attractive given current technology.

In summary, CDG is more expressive and flexible than CFGs, making it an attractive alternative to traditional parsers. It is able to utilize a variety of different knowledge sources in a uniform framework to incrementally disambiguate a sentence’s parse. The algorithm also has the advantage that it is efficiently parallelizable. Because CDG parsing differs from traditional parsers in its use of constraint propagation, further study of the relationship between CDG and other parsing approaches could lead to some useful insights about parsing algorithms in general.

### 3 Enhancements of CDG Text-Based Parsing

This section describes a number of enhancements we have made to the CDG parsing algorithm for single sentences to increase its usefulness for both text-based and spoken natural language processing. First, the algorithm is modified to parse sentences with lexically ambiguous words. Next, a table is introduced to lexically restrict the possible labels for a word during CN construction. Finally, the parsing algorithm is modified to allow constraints to test for features such as number or person.

---

<sup>8</sup>This algorithm has also been parallelized, and operates in linear time with  $O(n^5)$  processors [31].

### 3.1 Lexical Ambiguity

Many words in the English language have more than a single part of speech. For example, the word *program* in *The program runs* can be either a noun or a verb. Maruyama's algorithm requires that a word have a single part of speech, which is determined by dictionary lookup prior to the application of the parsing algorithm. Since parsing can be used to lexically disambiguate a sentence, ideally, a parsing algorithm should not require that a part of speech be known prior to parsing. In addition, lexical ambiguity, if not handled in a reasonable manner, can cause correctness and/or efficiency problems for a parser [2].

To handle lexically ambiguous words, we allow role values within the same node to have their own parts of speech as shown in the top CN in Figure 14<sup>9</sup>. When the CN is constructed, the parts of speech for each word are determined by looking the word up in the dictionary. If a word is lexically ambiguous then for each part of speech, a set of role values in the domain is created and assigned that part of speech. Note that the maximum size of this modified constraint network is  $\binom{p^n}{2} (wqn)^2$ , given that each word can have  $w$  parts of speech. Because each role value has its own part of speech, the constraints in  $G_1$  in Figure 1 are rewritten so that the access function `cat` operates on a role value rather than on a word node addressed by its position. The constraints in  $G_1$  are rewritten as follows:

```

;; [U-1] A det receives the label det
;; and modifies a word to its right.
(if (eq (cat x) det)
    (and (eq (lab x) det)
         (lt (pos x) (mod x))))

;; [U-2] A noun receives the label subj
;; and modifies a word to its right.
(if (eq (cat x) noun)
    (and (eq (lab x) subj)
         (lt (pos x) (mod x))))

;; [U-3] A verb receives the label root
;; and modifies no word.
(if (eq (cat x) verb)
    (and (eq (lab x) root)
         (eq (mod x) nil)))

;; [B-1] A det is governed by a noun.
(if (and (eq (lab x) det)
         (eq (mod x) (pos y)))
    (eq (cat y) noun))

```

This solution is compatible with the semantics of a CDG CN, which is like an AND/OR tree such that the values assigned to the roles account for the only OR nodes in the tree, as shown in Figure 12. For a sentence to have a parse, every role in the CN must have a least one role value after

---

<sup>9</sup>Rather than show each role value with its corresponding part of speech in the figures, we show the set of all of the role values with a particular part of speech to save space.

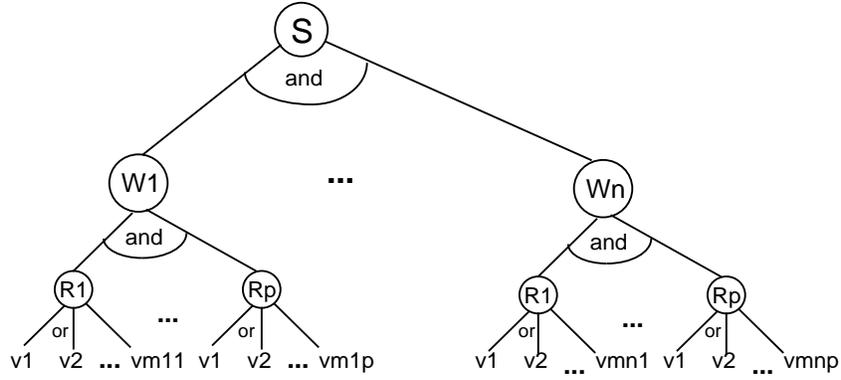


Figure 12: The AND/OR tree for the CDG parsing algorithm.

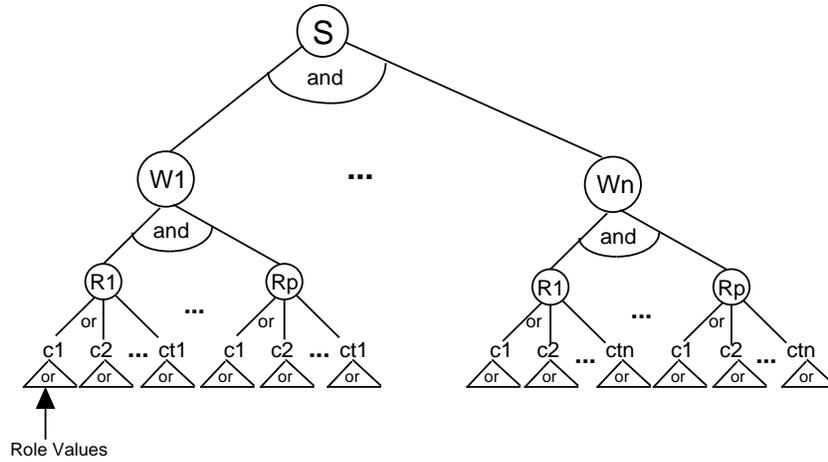


Figure 13: The AND/OR tree for the CDG parsing algorithm with lexical ambiguity.

filtering. Furthermore, the filtering algorithm requires that a role value in a role be supported by all of its arcs. One strategy for handling lexical ambiguity in a CN places the disjunction associated with the ambiguity at the level of the role, as shown in Figure 13. The categories are represented as the  $c_i$ s, which are ORed below the level of the role. Because of this, no modification of the CDG filtering algorithm is required, except that now it is possible for lexical categories of a word to loose support.

By using the modified CN and constraints, the CDG parsing algorithm, operating on the lexically ambiguous CN for *The program runs*, produces the same parse graph as the CN without lexical ambiguity for the same sentence from Section 2.2. This process is depicted in Figure 14. For grammars with  $p > 1$ , an additional constraint must be added to prevent role values associated with one lexical category from supporting role values for other categories of the same word.

Our approach to handling lexical ambiguity can easily be extended to handle multiple word candidates in the same time interval. In this case, each role value keeps track of its word candidate,

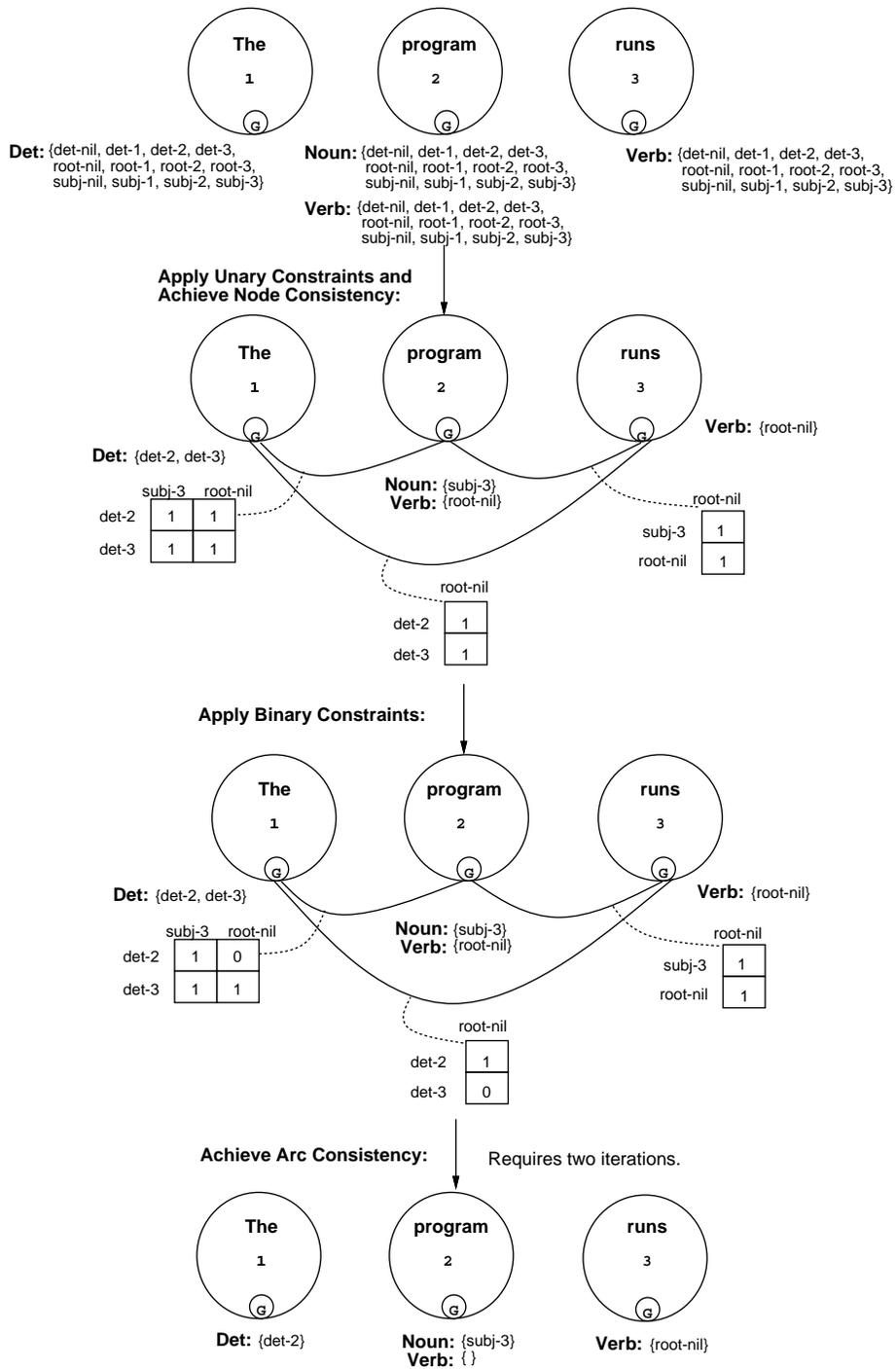


Figure 14: Parsing a lexically ambiguous CN with constraints.

		$L_1$ :		
		det	root	subj
$\Sigma_1$ :	det	1	0	0
	noun	0	0	1
	verb	0	1	0

Figure 15: A table of legal labels for word categories in the governor role for  $G_1$ .

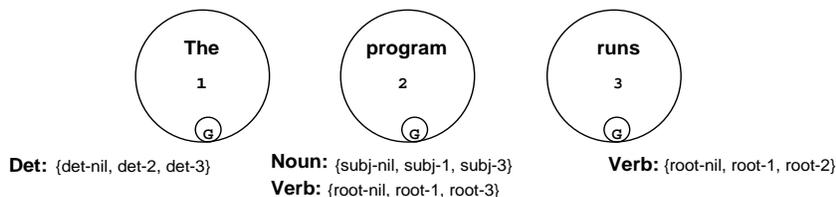


Figure 16: The initial CN given the table in Figure 18.

as well as its lexical category. In section 4, we will describe the final step needed to process general word graphs.

### 3.2 An Efficiency Issue: Label Pruning Using a Table

Given that the role values assigned to a role for a word in CDG are affected by both the role and the part of speech, it is possible, at network construction time, to restrict the role values assigned to a role for each part of speech for a word to those that are appropriate for the role and the lexical category under consideration. To do so, we add a fifth parameter to the CDG grammar tuple,  $T$ , where  $T$  is a *table* which restricts the possible labels for each role according to the category of the word and its role id. Now a CDG grammar consists of a quintuple,  $\langle \Sigma, R, L, C, T \rangle$ . Though  $T$  is not a necessary aspect of the grammar, it does make the analysis of a sentence more efficient because the roles are initialized to smaller domains, and many of the unary constraints (i.e., those which restrict labels of role values to lexically appropriate values) can be omitted. The table for augmenting grammar  $G_1$  is shown in Figure 15. It shows the legal labels for the governor role given the word categories in  $\Sigma_1$ . If this table is used during CN construction for the sentence *The program runs* along with the assumption that no word modifies itself, the resulting CN is depicted in Figure 16.

In practice, the table reduces the number of role values in the initial CN by a factor of five to seven and eliminates the need to propagate some unary constraints. Hence, it does affect the actual running time of the CDG algorithm, though it does not improve the asymptotic running time.

### 3.3 Lexical Features in CDG

Many times, even if a word is not lexically ambiguous, it can have ambiguity in the feature information associated with the word, like number, person, or case. For example, the noun *fish* can take the number/person feature value of third person singular or third person plural. Lexical features are used in many natural language parsers to enforce subject-verb agreement, determiner-head noun agreement, and case requirements for pronouns. This information can be very useful for disambiguating parses for sentences or for eliminating impossible sentence hypotheses, hence we have also added lexical feature analysis to the CDG parser.

The incorporation of feature tests into CDG parsing requires the same special care used to introduce lexical ambiguity into the parsing algorithm. Consider again the modified constraints of grammar G1.

```
;; [U-1] A det receives the label det
;; and modifies a word to its right.
(if (eq (cat x) det)
    (and (eq (lab x) det)
         (lt (pos x) (mod x))))

;; [U-2] A noun receives the label subj
;; and modifies a word to its right.
(if (eq (cat x) noun)
    (and (eq (lab x) subj)
         (lt (pos x) (mod x))))

;; [U-3] A verb receives the label root
;; and modifies no word.
(if (eq (cat x) verb)
    (and (eq (lab x) root)
         (eq (mod x) nil)))

;; [B-1] A det is governed by a noun.
(if (and (eq (lab x) det)
         (eq (mod x) (pos y)))
    (eq (cat y) noun))
```

We will consider how to add number/person feature tests to CDG to process the sentence *\*A fish swim*. The grammar should not generate this sentence because it is ungrammatical given appropriate number/person tests. To add number/person feature tests to CDG grammars, the algorithm for constructing the initial CN must be modified to look up number/person information for the word and store this information with the role values. The lexical entries for this example follow:

```
(a (category det (number 3s)))
(swim (category verb (number 1s 2s 1p 2p 3p)))
(fish (category noun (number 3s 3p))
      (category verb (number 1s 2s 1p 2p 3p)))
```

In addition to storing number/person information in the lexicon, two agreement constraints must

be added to  $G_1$  to ensure that the number of the determiner agrees with the number of the head noun and that the number of the subject agrees with the number of the root.

```

;; [B-2] A det which is governed by a noun
;; must agree in number with the noun's number.
(if (and (eq (lab x) det)
         (eq (cat y) noun)
         (eq (mod x) (pos y)))
    (agree (number x) (number y)))

;; [B-3] A subj which is governed by a root
;; must agree in number with the verb's number.
(if (and (eq (lab x) subj)
         (eq (lab y) root)
         (eq (mod x) (pos y)))
    (agree (number x) (number y)))

```

These constraints require the addition of one access function **number** and a predicate **agree**. The function (**number x**) returns the number/person information associated with the role value, and the predicate (**agree (number x) (number y)**) returns true only if its two number/person arguments agree. We consider two ways to store number/person information with a role value. One way is to store the entire set of features with each role value. In this case, agree returns true if the intersection of the two number sets is non-empty. The second approach is to store one number/person feature value per role value, and the agree predicate becomes equivalent to an equality test.

If the CDG parsing algorithm stores the set of number/person feature values with each role value, the CN depicted in Figure 17 for the sentence *\*A fish swim* is constructed using the table in Figure 15. This figure depicts the propagation of all of the phrasal constraints (i.e., U-1, U-2, U-3, and B-1).

Now consider the impact of constraints B-2 and B-3 on the network, and notice that the constraints succeed for the CN, despite the fact that the sentence is ungrammatical. This occurs because the words are checked pairwise for agreement. The word *a* agrees with *fish*, and the word *fish* agrees with *swim*, but the numbers that cause agreement on the two arcs are incompatible with each other. Using this approach, the only way to ensure that sets of numbers jointly agree for the determiner, subject, and verb is by propagating an agreement constraint over the three role values. This constraint would contain three variables as shown below:

```

;; A det that is governed by a subj, which is
;; governed by a root must also agree with the root.
(if (and (eq (lab x) det)
         (eq (lab y) subj)
         (eq (lab z) root)
         (eq (mod x) (pos y))
         (eq (mod y) (pos z)))
    (agree (number x) (number z)))

```

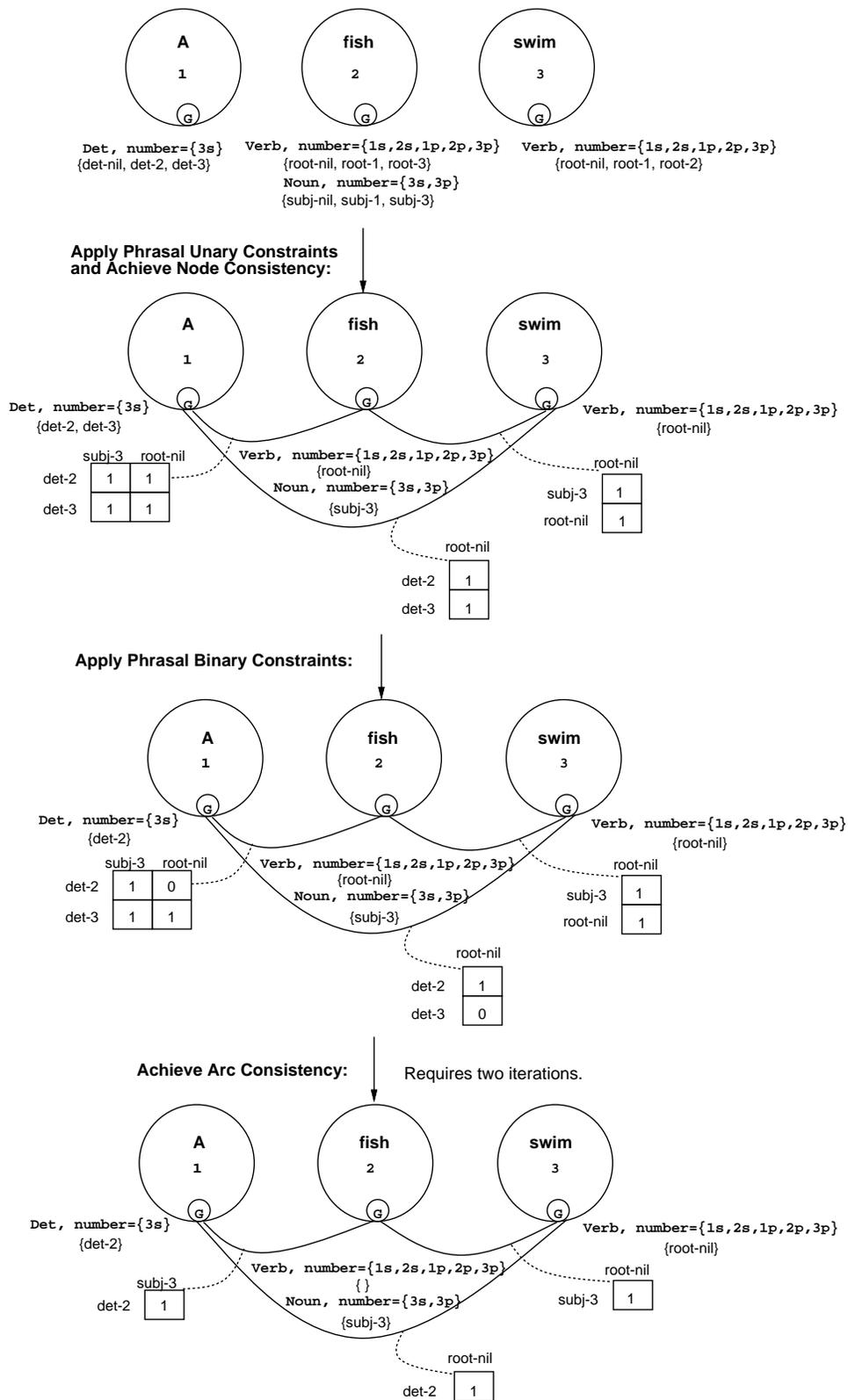


Figure 17: Parsing a CN with ambiguous features using phrasal constraints.

To propagate this constraint requires the addition of arcs linking triples of roles in the sentence and the use of three dimensional arc matrices. Because there are  $\binom{n+p}{3} = O(n^3)$  arcs required in a CN with 3-variable constraints, and each arc contains a matrix with  $(q * n)^3 = O(n^3)$  elements, the time to construct the arcs and initialize the matrices is  $O(n^6)$ , and the time to propagate a three variable constraint is  $O(n^6)$ . This constraint will work for the current example, but to handle four-way agreement for sentences like *\*The fish which are eating swims* would require constraints with an arity of four. Because of cases like these, we have developed another approach to feature testing.

To correctly utilize number and person features in agreement tests for CDG parsing without resorting to greater than two-variable constraints, each role value must be assigned a **single** feature value, not a set of values, at the point when a feature constraint is applied. If there are two feature types (say number/person and case) to be used in constraints for a grammar, then the role values will have to be duplicated and assigned feature values from the cross product of the features' values. This could easily lead to a combinatorial explosion of role values. Fortunately, there is an excellent strategy for limiting the number of role values. The basic idea is to store the sets of feature values with a single role value and to duplicate the role values only on demand, when a particular feature type is being tested by a constraint.

Consider how the sentence *\*A fish swim* is processed given this strategy. Assume that the phrasal constraints are propagated first, and arc consistency is achieved as shown in Figure 17. Note that many of the role values have been eliminated by the phrasal constraints before the feature constraints are propagated. Now in preparation for the propagation of constraints using the number/person feature, the role values in the CN at the bottom of Figure 17 must be duplicated for each number/person feature value, giving the CN at the top of Figure 18. This figure also depicts the propagation of the feature constraints and filtering, after which no role values remain.

A grammar writer can order constraints in a constraint file in such a way that role values are reduced by pure phrase structure constraints prior to the feature constraints. Also, feature constraints can be ordered to minimize useless role value duplication. When the parser is preparing to propagate a constraint with a particular feature test, each of the role values having multiple values for that feature is duplicated and assigned one of the feature values. The corresponding feature constraints should then eliminate many of the duplicated role values before other types of feature constraints are propagated.

A wide variety of grammatical formalisms explicitly or implicitly divide linguistic constraints into phrasal and attribute-value feature constraints, the reason being that context-free phrase structure constraints can be solved in polynomial time in the length of the sentence, but known algorithms for solving Boolean combinations of equality constraints can be worst case exponential

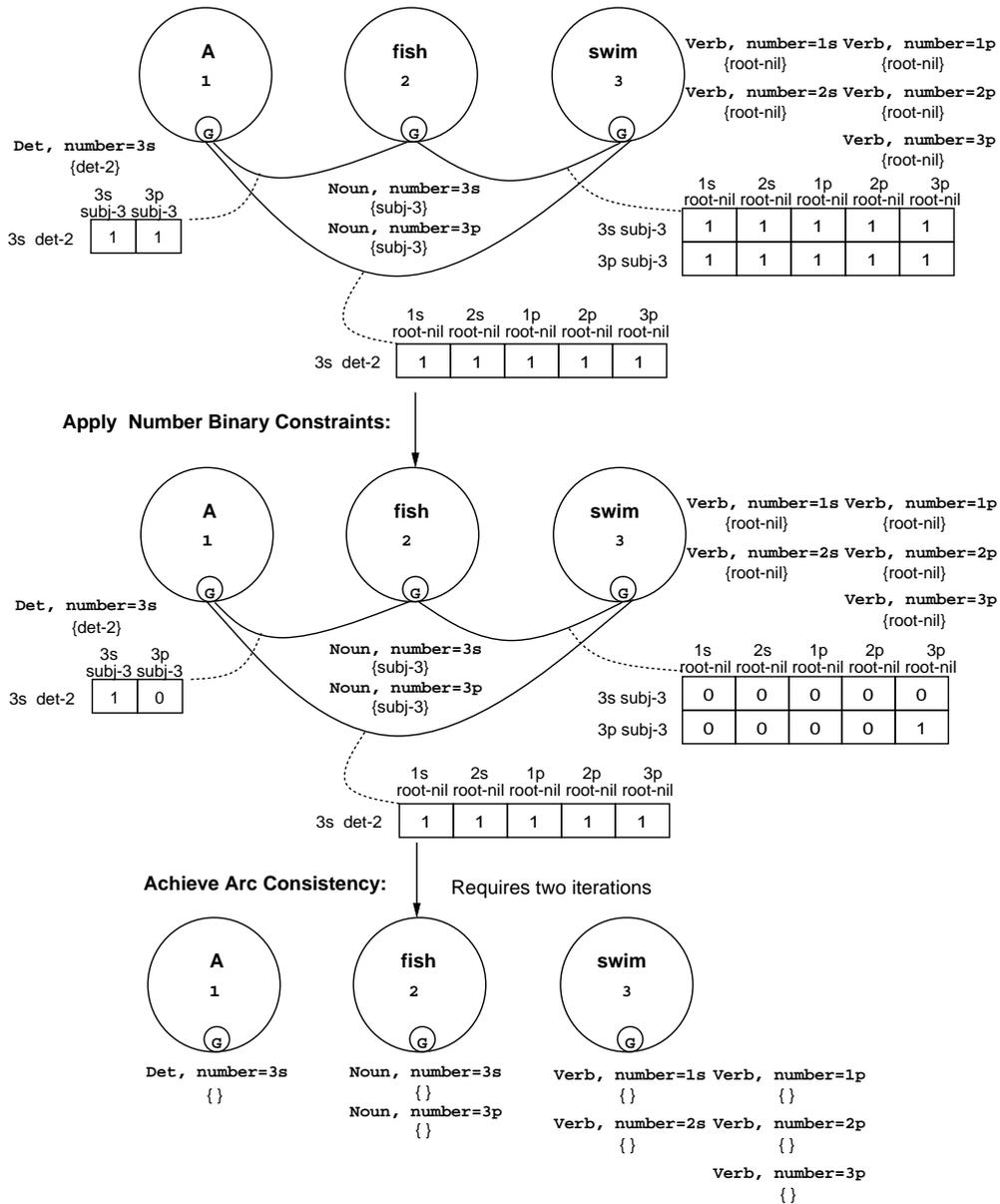


Figure 18: Parsing a CN with ambiguous features using number constraints.

[27]. Hence, we believe that our solution is similar to other approaches, and in general, feature constraints are tight enough that they decrease the number of role values after arc consistency is applied. There is some flexibility to apply feature constraints prior to phrase structure constraints, but typically phrase structure constraints prune more than feature constraints by themselves.

A grammar is lexicalized [37] if each of the basic structures it manipulates is associated with the lexical item. The benefits of using a lexicalized grammar include the fact that they are finitely ambiguous and have decidable recognition algorithms. In addition, much of the linguistic detail of a grammar is encoded in the lexicon (e.g., subcategorization). Our modification of CDG is lexicalized in this sense, though the original CDG grammar was not.

## 4 Spoken Language Modification

CDG has several properties which make it attractive for use in spoken language understanding systems. First, it uses a single representation, the constraint, to apply different knowledge sources to a single data structure, the constraint network. This uniformity is especially compelling for speech understanding because such a system could potentially use lexical, syntactic, semantic, prosodic, and contextual rules. Second, parsing can proceed incrementally. Additional knowledge sources can be used if the degree of ambiguity remains high. Third, it is able to handle grammars that are beyond context-free. Fourth, the flexibility of incremental constraint-based parsing should be less sensitive than CFG parsers to the syntactic irregularities common in spontaneous speech. It should be possible to apply grammars differently when handling speakers with various types of disfluencies. Fifth, CDG is able to support the use of context when determining the meaning of a sentence (especially to reduce ambiguity). Finally, the algorithm is amenable to effective parallel implementation.

One drawback of the CDG parser as defined by Maruyama is that it is only able to process one sentence at a time. However, since a speech recognizer can generate multiple sentence hypotheses for a given utterance, a one-sentence-at-a-time parser would be very inefficient. Hence, in this section, we extend the CDG parser to process word graphs containing multiple sentence hypotheses.

In the Section 4.1, we briefly describe current spoken language approaches and motivate the use of word graphs for processing the multiple sentence hypotheses provided by a speech recognizer. In Section 4.2, we adapt the CDG parsing algorithm to operate directly on a word graph. In Section 4.3, we describe how the filtering algorithm must be modified to simultaneously process multiple sentence hypotheses. In Section 4.4, we describe our implementation of the spoken language parser and the development of two constraint-based grammars. Finally, in Section 4.5, we describe how the parser can utilize word scores to focus on the most likely sentence hypotheses.

## 4.1 Current Approaches to Speech

Among the most successful current speech recognition systems which process continuous speech for a limited (1000 word) vocabulary are those which utilize hidden Markov models (HMM). Hidden Markov modeling has been one of the most successful strategies for acoustic pattern matching [20, 22], but this method is generally difficult to integrate with adequate language models.

State of the art speech recognition systems achieve high recognition accuracies on tasks that have low perplexities. The perplexity of a task is, roughly speaking, the average number of choices at any decision point. Most systems using HMMs (e.g., [21, 40]) have reduced recognition errors by incorporating some language model into their system to reduce perplexity. The trend in recent systems has been to use stochastic language models [32, 55]. However, this approach is limited to relatively simple cases (e.g., bigram or trigram) in order to control network size and complexity of training. Furthermore, since the goal of these systems is recognition, not understanding, no structural analysis of the utterance is performed. Instead, the output of such systems is often an ordered list of the  $N$  most likely sentence hypotheses (where  $N$  is a constant usually less than 100) [39, 41].

The question of how to integrate language models with speech recognition systems is becoming more important as speech understanding becomes the focus. Early systems [23, 52] grappled with knowledge source interaction and flow of control. Approaches that jointly model the grammar and the acoustic signal have been applied to small problems successfully. Widespread use of these strategies for larger problems has been limited due to computational costs, insufficient training data, or an inadequate language model. By separating the language model from the acoustic model, it is possible to use a more accurate language model without increasing computational costs or the amount of training data required. Also a variety of language models can be tried with a single acoustic model. Systems that are attempting to integrate speech recognition processing with more traditional natural language processing techniques include CMU's Phoenix, using frame based parsing and semantic phrase grammars [51]; CSELT's system, based on finite-state language models [5]; MIT's Voyager, using LR parsing [57]; and Seneff's robust parsing [42, 43]. See [9] for discussion of various approaches to integrating speech recognition with natural language processing techniques.

To construct a speech understanding system which builds on current recognizers, a researcher might pass the  $N$ -best sentence hypotheses generated by a recognizer through a natural language parser as a first step toward producing meaning representations. However, processing each sentence hypothesis provided by a speech recognizer individually is inefficient since the sentence hypotheses often differ only slightly from each other. Furthermore, a list of sentence hypotheses is not the most compact representation to provide a natural language parser. A better representation for

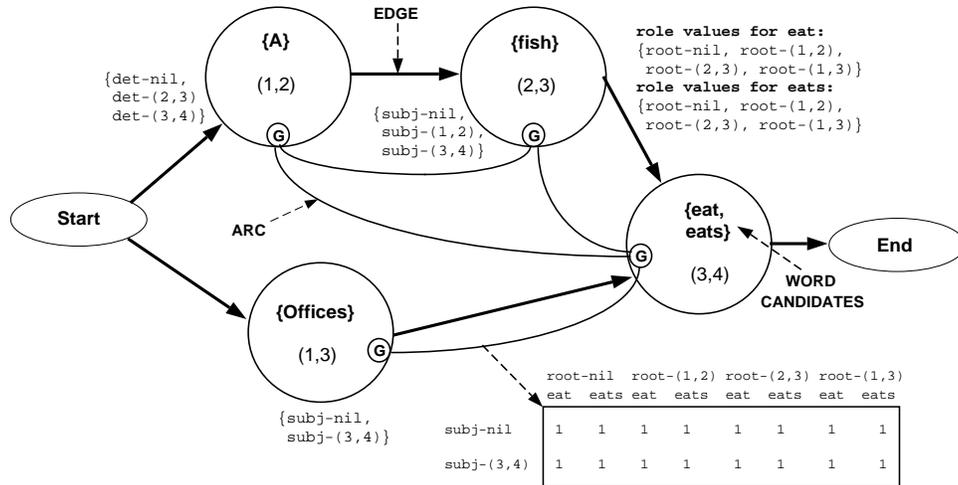


Figure 19: Example of a spoken language constraint network constructed from a word graph.

the sentence hypotheses is a word graph of word candidates which contains information on the approximate beginning and end point of each word’s utterance.

Even though a word graph is a compact representation for the output of a speech recognition system, most current systems do not provide this type of representation. However, parsers that can process the graph representation should more efficiently process the sentence hypotheses. Tomita [47] has developed an LR parsing algorithm capable of processing a word graph. Chart parsers can also process a word graph by storing the words in its chart. The CDG approach can also be extended to operate on a word graph, as we show in the next section. This extension makes the power and flexibility of CDG applicable to the problem of eliminating spurious sentence hypotheses.

## 4.2 Parsing Word Graphs with Constraints

We have adapted the CDG constraint network to handle the multiple sentence hypotheses stored in a word graph, calling it a Spoken Language Constraint Network (SLCN). Figure 19 depicts an SLCN derived from a word graph constructed for the sentence hypotheses: *\*A fish eat* and *\*Offices eats*. By representing these hypotheses in a word graph, we are able to process additional sentences (i.e., *A fish eats* and *Offices eat*) not present in the list of hypotheses, one of which might represent the intended utterance. Notice that word nodes contain a list of all word candidates with the same beginning and end points, and *edges* join word nodes that can be adjacent in a sentence hypothesis (see Figure 19). A sentence hypothesis must include one word node from the beginning of the utterance, one word node from the end of the utterance, and these two word nodes must be connected by a path of edges. The word nodes along a path can contain multiple word candidates, so the number of sentence hypotheses for a particular path of edges can be quite large. In the SLCN of Figure 19, each word node contains information on the beginning and end point of the word’s

utterance, represented as an integer tuple  $(b, e)$ , with  $b < e$ . The tuple is more expressive than the point scheme used for CNs and requires modification of some of the access functions and predicates defined for the CN scheme. The access functions  $(\text{pos } x)$  and  $(\text{mod } x)$  now return a tuple  $(b, e)$  which describes the position of the word associated with the role value  $x$ . The equality predicate must be extended to test for equality of intervals (e.g.,  $(\text{eq } (1,2) (1,2))$  should return true). The less-than predicate,  $(\text{lt } (b1, e1) (b2, e2))$ , returns true if  $e1 < b2$ , and the greater than predicate,  $(\text{gt } (b1, e1) (b2, e2))$ , returns true if  $b1 > e2$ .

To parse an SLCN, each word candidate contained in a word node is assigned a set of role values for each role, requiring  $O(n^2)$  time, where  $n$  is the number of word candidates in the graph. Unary constraints are applied to each of the role values in the network, and like CNs, require  $O(k_u * n^2)$  time.

The preparation of the SLCN for the propagation of binary constraints is similar to that for a CN. All roles within the same word node are joined with an arc as in a CN; however, roles in different word nodes are joined with an arc if and only if they can be members of at least one common sentence hypothesis (i.e., they are connected by a path of directed edges). To construct the arcs and arc matrices for an SLCN, it suffices to traverse the graph from beginning to end and string arcs from each of the current word node's roles to each of the preceding word node's roles (where a node precedes a node if there is a directed edge from the preceding to the current node) and to each of the roles that the preceding word nodes' roles have arcs to. For example, there should be an arc between the roles for *a* and *fish* in Figure 19 because they are located on a path from the beginning to the end of the sentence *a fish {eats, eat}*. However, there should not be an arc between the roles for *a* and *offices* since they are not found in any of the same sentence hypotheses. After the arcs for the SLCN are constructed, the arc matrices are constructed in the same manner as for a CN. The time required to construct the SLCN network in preparation for binary constraint propagation is  $O(n^4)$  because there may be up to  $O(n^2)$  arcs constructed, each requiring the creation of a matrix with  $O(n^2)$  elements. Once the SLCN is constructed, binary constraints are applied to pairs of role values associated with arc matrix entries (in the same manner as for the CN), requiring  $O(k_b * n^4)$  time, where  $n$  is the number of word candidates.

Filtering in an SLCN is complicated by the fact that the limitation of one word's function in one sentence hypothesis should not necessarily limit that word's function in another sentence hypothesis. For example, consider the SLCN depicted in Figure 20. Even though all the role values for *are* would be disallowed by the singular subject *empath*, those role values cannot be eliminated since they are supported by *paths*, the subject in a different hypothesis. The SLCN filtering algorithm cannot disallow role values that are allowed by at least one sentence in the network, in contrast to the CN filtering algorithm. Hence, we must modify the CN filtering algorithm to accommodate

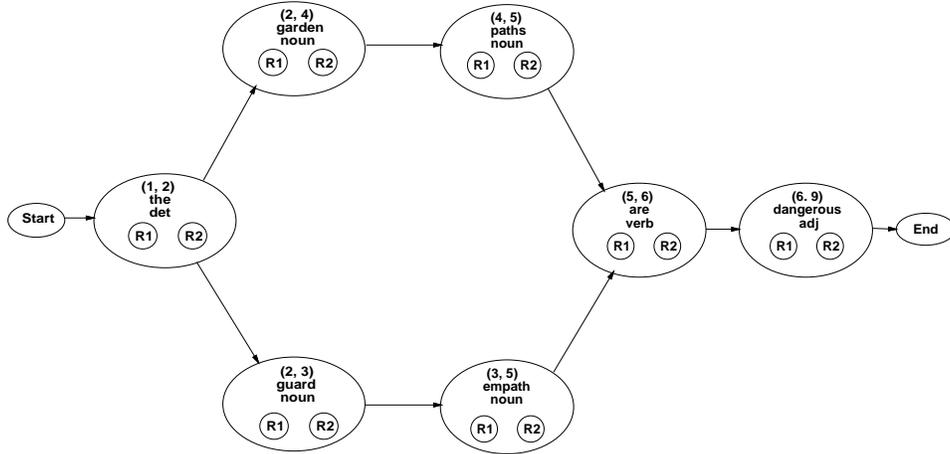


Figure 20: A simple SLCN.

word graphs. We have developed an algorithm to achieve arc consistency in an SLCN by using the properties of the directed acyclic graph representing the word network to filter role values that can never appear in any parse [13]. This algorithm, described in the next section, can process single sentences or word graphs.

### 4.3 SLCN Arc Consistency

When we create a constraint network representing multiple alternative sentence hypotheses, we have changed the logical meaning of the constraint network. A CN can be thought of as an AND/OR graph such that the values assigned to the roles of a word account for the only OR nodes in the graph, as shown in Figure 12. Hence, for a sentence to have a parse, every role in the CN must have a least one role value after filtering. A CN is said to be *arc consistent* if and only if for every pair of roles  $i$  and  $j$ , each role value in the domain of  $i$  has at least one role value in the domain of  $j$  for which they both satisfy the binary constraints.

On the other hand, an SLCN is constructed from a parse graph containing multiple sentence candidates, some with shared word nodes. Figure 20 depicts a simple SLCN with two roles constructed from a word graph. Figure 21 depicts the logical meaning of this SLCN. An OR node is required at the top level of the AND/OR graph to represent the contribution of various word nodes to the different sentence hypotheses. The SLCN arc consistency algorithm must be modified to handle the presence of this OR node.

An instance of an SLCN is said to be *SLCN arc consistent* if and only if for every role value  $a$  in the domain of each role, there is at least one sentence whose roles' domains contain at least one role value  $b$  which supports that value. Hence, even though a binary constraint might disallow a role value in one sentence, it might allow it in another. When enforcing arc consistency in a CN,

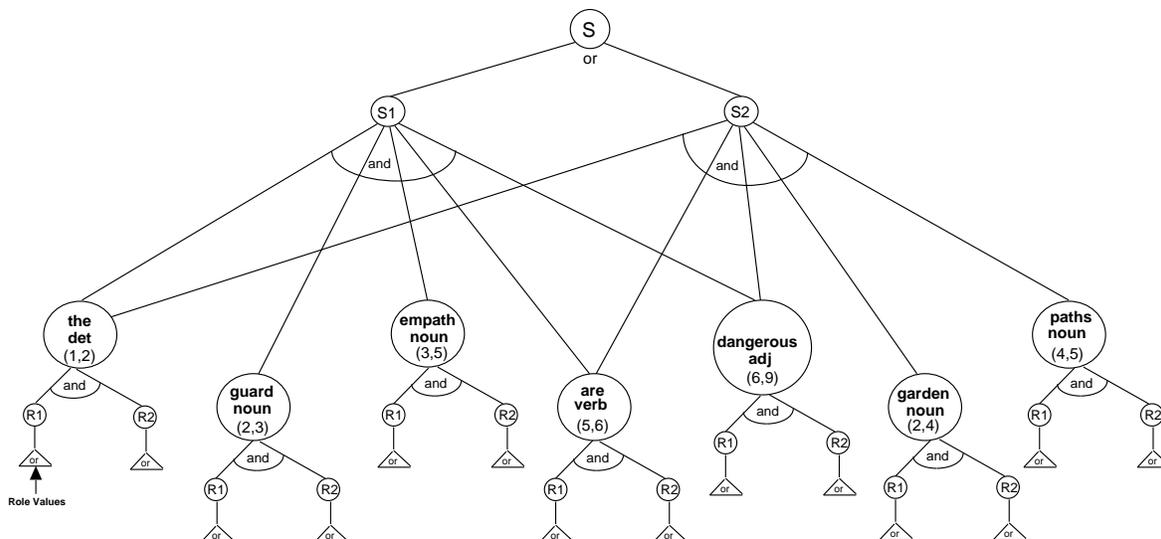


Figure 21: The AND/OR tree for the SLCN in Figure 23.

a role value  $a$  in the domain of  $i$  can be eliminated from role  $i$  whenever any other role has no role values which together with  $a$  satisfy the binary constraints. In an SLCN, however, a role value can be eliminated from a role only if it fails to satisfy the binary constraints in all the sentences in which it appears. Note that SLCN arc consistency reduces to CN arc consistency when the number of sentences is one.

The SLCN arc consistency algorithm builds and maintains several data structures, described in Table 4. Figure 24 shows the code for initializing the data structures, and Figure 25 contains the algorithm for eliminating inconsistent role values from the domains. The algorithm initially assumes that each word node has a single role. After we present the algorithm, we will discuss how multiple roles are supported.

The basic idea of the SLCN arc consistency algorithm is to remove role value  $a$  from role  $i$  if  $a$  has no support in any of the sentence hypotheses. If the role value  $a \in L_i$  is compatible with role value  $b \in L_j$ , then  $a$  supports  $b$ . To keep track of  $a$ 's support, the number of role values in  $L_j$  which are compatible with  $a$  in  $L_i$  are counted, and the total is stored in  $\text{Counter}[(i, j), a]$ . The algorithm must also keep track of which role values that  $a$  supports by using  $S[(i, j), a]$ , which is a set of arc and role value pairs. Note that for  $a \in L_i$  to support  $b \in L_j$ ,  $R2(i, a, j, b)$  must evaluate to **1** and one of the following conditions must hold:

1.  $\exists k \in N (k = (\text{role-of}(\text{mod } b)) \wedge (i, k) \in E)$ ,
2.  $(\text{mod } b) = \text{nil}$ ,
3.  $(\text{mod } b) = (\text{pos } a)$ .

These three conditions are used to ensure that  $b$  is a role value that can occur in at least one

Notation	Meaning
$(i, j)$	An ordered pair of roles.
$N$	$\{i, j, \dots\}$ is the set of all roles, with $ N  = n * p$ .
$L$	$\{a, b, \dots\}$ is the set of <i>role values</i> , with $ L  = q * (n + 1)$ .
$L_i$	The domain of role $i$ which is supported by the constraints.
$R2(i, a, j, b)$	$R2(i, a, j, b) = 1$ indicates the admissibility $a \in L_i$ and $b \in L_j$ after binary constraint propagation. This corresponds to the element indexed by $[a, b]$ in the matrix for arc $(i, j)$ in our algorithm.
$[(i, j), a]$	An ordered pair of a role pair $(i, j)$ and a role value $a \in L_i$ .
$M[(i, j), a]$	$M[(i, j), a] = 1$ indicates that the role value $a$ is not admissible for (and has already been eliminated from) all sentences containing $i$ and $j$ .
$E$	All role pairs $(i, j)$ such that there exists a sentence containing both $i$ and $j$ . If $(i, j) \in E$ , then $(j, i) \in E$ .
$S[(i, j), a]$	$[(j, i), b] \in S[(i, j), a]$ means that role values $a \in L_i$ and $b \in L_j$ are simultaneously admissible. This implies that $a$ supports $b$ .
Next-edge <sub><math>i</math></sub>	If a directed edge from $i$ to $j$ exists, then $(i, j) \in$ Next-edge <sub><math>i</math></sub> .
Prev-edge <sub><math>i</math></sub>	If a directed edge from $j$ to $i$ exists, then $(j, i) \in$ Prev-edge <sub><math>i</math></sub> .
Counter $[(i, j), a]$	The number of role values in $L_j$ compatible with $a \in L_i$ .
Prev-Support $[(i, j), a]$	$(i, k) \in$ Prev-Support $[(i, j), a]$ means that $a$ is admissible in every sentence which contains $i, j$ , and $k$ .
Next-Support $[(i, j), a]$	$(i, k) \in$ Next-Support $[(i, j), a]$ means that $a$ is admissible in every sentence which contains $i, j$ , and $k$ .
Local-Prev-Support $(i, a)$	A set of elements $(i, j)$ such that $(j, i) \in$ Prev-edge <sub><math>i</math></sub> and $a$ is compatible with at least one of $j$ 's role values.
Local-Next-Support $(i, a)$	A set of elements $(i, j)$ such that $(i, j) \in$ Next-edge <sub><math>i</math></sub> and $a$ is compatible with at least one of $j$ 's role values.
List	A queue of arc support to be deleted. Note that $[(i, j), a]$ is placed on List if $a$ is invalid in all sentences containing $i$ and $j$ .

Table 4: Data structures and notation for the SLCN arc consistency algorithm.

sentence hypothesis with  $a^{10}$ . For example,  $S[(i, j), a] = \{[(j, i), b], [(j, i), c]\}$  means that  $a$  in  $L_i$  supports  $b$  and  $c$  in  $L_j$ . If  $a$  is ever invalid for  $L_i$ , then  $b$  and  $c$  will lose some of their support. This is accomplished by decrementing  $\text{Counter}[(j, i), b]$  and  $\text{Counter}[(j, i), c]$ . If  $\text{Counter}[(i, j), a]$  becomes zero,  $[(i, j), a]$  would be placed on the *List* for further processing. Remember that for CN arc consistency, if  $\text{Counter}[(i, j), a]$  becomes zero,  $a$  would also be immediately removed from  $L_i$ , because it would be incompatible with every sentence parse. However, this is not necessarily the case for SLCNs because even though  $a$  does not participate in a parse for any of the sentences which contain  $i$  and  $j$ , there could be another sentence for which  $a$  is perfectly legal. A role value cannot become globally inadmissible until it is incompatible with every sentence.

Because an SLCN is represented as a directed acyclic graph (DAG), the algorithm is able to use the properties of DAGs to identify local (and hence efficiently computable) conditions under which role values become globally inadmissible. For the sake of discussion, we assume that each node contains a single role and the directed edges associated with the word node relate the roles in the SLCN. Consider Figure 22, which shows the roles that are adjacent to role  $i$  in an SLCN. Because every sentence in the SLCN which contains role  $i$  is represented as a path going through role  $i$ , either role  $j$  or role  $k$  must be in every sentence containing  $i$ . Hence, if the role value  $a$  is to remain in  $L_i$ , it must be compatible with at least one role value in either  $L_j$  or  $L_k$ . Also, because either  $n$  or  $m$  must be contained in every sentence containing  $i$ , if  $a$  is to remain in  $L_i$ , it must also be compatible with at least one role value in either  $L_n$  or  $L_m$ .

In order to track this dependency, two sets are maintained for each role value  $a \in L_i$ :  $\text{Local-Next-Support}(i, a)$  and  $\text{Local-Prev-Support}(i, a)$ .  $\text{Local-Next-Support}(i, a)$  is a set of ordered role pairs  $(i, j)$  such that  $(i, j) \in \text{Next-edge}_i$ , and there is at least one role value  $b \in L_j$  which is compatible with  $a$ .  $\text{Local-Prev-Support}(i, a)$  is a set of ordered pairs  $(i, j)$  such that  $(j, i) \in \text{Prev-edge}_i$ , and there is at least one role value  $b \in L_j$  which is compatible with  $a$ . Whenever one of  $i$ 's adjacent roles,  $j$ , no longer has any role values  $b$  in its domain which are compatible with  $a$ , then  $(i, j)$  should be removed from  $\text{Local-Prev-Support}(i, a)$  or  $\text{Local-Next-Support}(i, a)$ , depending on whether the edge is from  $j$  to  $i$  or from  $i$  to  $j$ , respectively. If either  $\text{Local-Prev-Support}(i, a)$  or  $\text{Local-Next-Support}(i, a)$  becomes the empty set, then  $a$  is no longer a part of any parse, and may be eliminated from  $L_i$ . In Figure 22, the role value  $a$  is admissible for the sentence containing  $i$  and  $j$ , but not for the sentence containing  $i$  and  $k$ . The sets in the figure indicate that the role value  $a$  is allowed for every sentence which contains  $n$  or  $m$  and  $j$ , but is disallowed for every sentence which contains  $k$ . The solid directed lines in the figure represent the SLCN edges, and the dotted directed lines represent the arcs. We use the directionality of the arcs to represent the fact that an

---

<sup>10</sup>The only condition for  $a$  supporting  $b$  in the general MUSE CSP algorithm [13] is  $R2(i, a, j, b)$  because not all values assigned to variables in CSP have modifiers.

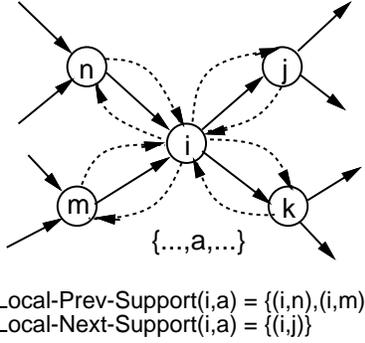


Figure 22: Local-Prev-Support and Local-Next-Support for an example SLCN.

arc matrix associated with an arc is used in two ways. For example,  $n$ 's role values support  $i$ 's role values, but also  $i$ 's role values support  $n$ 's. If because of additional constraints, the role values in  $j$  become inconsistent with  $a$  in  $L_i$ ,  $(i, j)$  would be eliminated from  $\text{Local-Next-Support}(i, a)$ , leaving an empty set. In that case,  $a$  would no longer be supported by any sentence.

**Theorem 1 (Local-Prev-Support)** *If  $\text{Local-Prev-Support}(i, a)$  associated with the role value  $a \in L_i$  becomes empty then that role value can never be a member of a parse for the sentence, and so can be deleted.*

**Proof:** Every role value  $a \in L_i$  must be allowed by at least one of the roles that precedes  $i$  to be a member of a parse for a sentence. An element  $(i, j)$  is deleted from  $\text{Local-Prev-Support}(i, a)$  because  $j$ , a role that immediately precedes  $i$ , has no role values that are compatible with  $a$ . If all elements are deleted from  $\text{Local-Prev-Support}(i, a)$ , then no adjacent preceding role supports  $a$ , so it can be deleted from the role  $i$ .

**Theorem 2 (Local-Next-Support)** *If  $\text{Local-Next-Support}(i, a)$  associated with the role value  $a \in L_i$  becomes empty then that role value can never be a member of a parse for the sentence, and so can be deleted.*

**Proof:** Every role value  $a \in L_i$  must be allowed by at least one of the roles that follows  $i$  to be a member of a parse for a sentence. An element  $(i, j)$  is deleted from  $\text{Local-Next-Support}(i, a)$  because  $j$ , a role that immediately follows  $i$ , has no role values that are compatible with  $a$ . If all elements are deleted from  $\text{Local-Next-Support}(i, a)$ , then no adjacent following role supports  $a$ , so it can be deleted from the role  $i$ .

**Theorem 3 (Local-Prev-Support and Local-Next-Support)** *If  $\text{Local-Next-Support}(i, a)$  or  $\text{Local-Prev-Support}(i, a)$  associated with the role value  $a \in L_i$  becomes empty then that role value can never be a member of a parse for the sentence, and so can be deleted.*

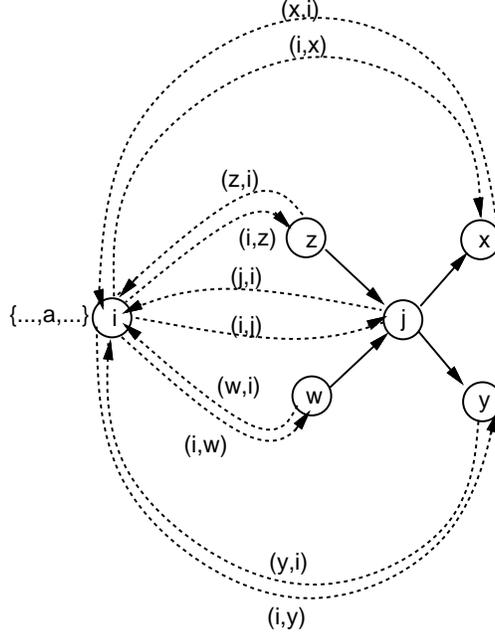


Figure 23: If  $\text{Next-edge}_j = \{(j, x), (j, y)\}$ ,  $S[(i, x), a] = \phi$ , and  $S[(i, y), a] = \phi$ , then  $a$  is inadmissible in every sentence containing both  $i$  and  $j$ .

**Proof:** This follows directly from Theorems 1 and 2.

The algorithm can utilize similar conditions for roles which may not be directly connected to  $i$  by  $\text{Next-edge}_i$  or  $\text{Prev-edge}_i$ . Consider Figure 23. Suppose that the role value  $a \in L_i$  is compatible with a role value in  $L_j$ , but it is incompatible with the role values in  $L_x$  and  $L_y$ , then it is reasonable to eliminate  $a$  for all sentences containing both  $i$  and  $j$ , because those sentences would have to include either role  $x$  or  $y$ . To determine whether a role value is admissible for a set of sentences containing  $i$  and  $j$ , we calculate  $\text{Prev-Support}[(i, j), a]$  and  $\text{Next-Support}[(i, j), a]$ .  $\text{Next-Support}[(i, j), a]$  includes all  $(i, k)$  arcs which support  $a$  in  $L_i$  given that there is a directed edge between  $j$  and  $k$ , and  $(i, j)$  supports  $a$ .  $\text{Prev-Support}[(i, j), a]$  includes all  $(i, k)$  arcs which support  $a$  in  $i$  given that there is a directed edge between  $k$  and  $j$ , and  $(i, j)$  supports  $a$ . Note that  $\text{Prev-Support}[(i, j), a]$  will contain an ordered pair  $(i, j)$  if  $(i, j) \in \text{Prev-edge}_j$ , and  $\text{Next-Support}[(i, j), a]$  will contain an ordered pair  $(i, j)$  if  $(j, i) \in \text{Next-edge}_j$ . These elements are included because the edge between roles  $i$  and  $j$  is sufficient to allow  $j$ 's role value to support  $a$  in the sentences containing  $i$  and  $j$ . Dummy ordered pairs are also created to handle cases where a role is at the beginning or end of a network: when  $(\mathbf{start}, j) \in \text{Prev-edge}_j$ ,  $(i, \mathbf{start})$  is added to  $\text{Prev-support}[(i, j), a]$ , and when  $(j, \mathbf{end}) \in \text{Next-edge}_j$ ,  $(i, \mathbf{end})$  is added to  $\text{Next-support}[(i, j), a]$ . This is to prevent a role value from being ruled out because no roles precede or follow it in the SLCN.

**Theorem 4 (Prev-Support)** *If  $\text{Prev-Support}[(i, j), a]$  associated with the role value  $a \in L_i$  becomes empty then that role value can never be a member of a parse for a sentence containing roles*

$i$  and  $j$ .

**Proof:** Every role value  $a$  associated with a role must be allowed by at least one of the roles that precedes it and one that follows it to be a member of a parse for a sentence. Assume that  $\text{Local-Prev-Support}(i, a)$  and  $\text{Local-Next-Support}(i, a)$  are non-empty, otherwise,  $a$  would no longer be valid in any sentence. An element  $(i, k)$  is deleted from  $\text{Prev-Support}[(i, j), a]$  because  $k$ , a role that immediately precedes  $j$ , has no role values that are compatible with  $a$ . If all elements of the set are deleted, then no adjacent role preceding  $j$  supports the value  $a$ . Hence, if  $\text{Next-Support}[(i, j), a]$  becomes empty,  $a$  cannot appear in a parse for any sentences involving roles  $i$  and  $j$ .

**Theorem 5 (Next-Support)** *If  $\text{Next-Support}[(i, j), a]$  associated with the role value  $a \in L_i$  becomes empty then that role value can never be a member of a parse for a sentence containing roles  $i$  and  $j$ .*

**Proof:** Every role value  $a$  associated with a role must be allowed by at least one of the roles that precedes it and one that follows it to be a member of a parse for a sentence. Assume that  $\text{Local-Prev-Support}(i, a)$  and  $\text{Local-Next-Support}(i, a)$  are non-empty, otherwise,  $a$  would no longer be valid in any sentence. An element  $(i, k)$  is deleted from  $\text{Next-Support}[(i, j), a]$  because  $k$ , a role that immediately follows  $j$ , has no role values that are compatible with  $a$ . If all elements of the set are deleted, then no adjacent role following  $j$  supports the value  $a$ . Hence, if  $\text{Next-Support}[(i, j), a]$  becomes empty,  $a$  cannot appear in a parse for any sentences involving roles  $i$  and  $j$ .

**Theorem 6 (Local-Prev-Support and Local-Next-Support)** *If  $\text{Prev-Support}[(i, j), a]$  or  $\text{Next-Support}[(i, j), a]$  associated with the role value  $a \in L_i$  becomes empty then that role value can never be a member of a parse for a sentence containing roles  $i$  and  $j$ .*

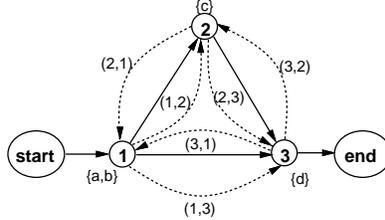
**Proof:** This follows directly from Theorems 4 and 5.

Figure 24 shows the  $\text{Prev-Support}$ ,  $\text{Next-Support}$ ,  $\text{Local-Next-Support}$ , and  $\text{Local-Prev-Support}$  sets that the initialization algorithm creates for some role values in a simple example SLCN. To illustrate how these data structures are used in SLCN arc consistency (see Figure 25), consider what happens if initially  $[(1, 3), a] \in \text{List}$  for the SLCN in Figure 24.  $[(1, 3), a]$  is placed on the list to indicate that the role value  $a$  in role 1 is not supported by any of the role values associated with role 3. When that value is popped off  $\text{List}$ , it is necessary to remove  $[(1, 3), a]$ 's support from all  $[(3, 1), x]$  such that  $(3, x) \in \text{S}[(1, 3), a]$  by decrementing for each  $x$ ,  $\text{Counter}[(3, 1), x]$  by one. If the counter for any  $[(3, 1), x]$  becomes 0, and the value has not already been placed on the  $\text{List}$ , then it is added for future processing. Once this is done, it is necessary to remove  $[(1, 3), a]$ 's influence on the SLCN. To handle this, we examine the two sets  $\text{Prev-Support}[(1, 3), a] =$

```

1. List:= $\phi$ ;
2.  $E := \{(i, j) | \exists \sigma \in \Sigma : i, j \in \sigma \wedge i \neq j \wedge i, j \in N\}$ ;
3. for  $(i, j) \in E$  do
4.   for  $a \in L_i$  do {
5.      $S[(i, j), a] := \phi$ ;
6.      $M[(i, j), a] := 0$ ;
7.     Local-Prev-Support( $i, a$ ) :=  $\phi$ ; Local-Next-Support( $i, a$ ) :=  $\phi$ ;
8.     Prev-Support $[(i, j), a] := \phi$ ; Next-Support $[(i, j), a] := \phi$ ; }
9. for  $(i, j) \in E$  do
10.  for  $a \in L_i$  do {
11.    Total:=0;
12.    for  $b \in L_j$  do
13.      if  $R \not\subseteq (i, a, j, b) \wedge$ 
14.         $[\exists k \in N (k = \text{role-of}(\text{mod } b) \wedge (i, k) \in E) \vee$ 
15.         $(\text{mod } b) = \text{nil} \vee (\text{mod } b) = (\text{pos } a)]$  then{
16.        Total:=Total+1;
17.         $S[(j, i), b] := S[(j, i), b] \cup \{(i, a)\}$ ; }
18.      if Total=0 then {
19.        List:=List  $\cup \{(i, j), a\}$ ;
20.         $M[(i, j), a] := 1$ ; }
21.      Counter $[(i, j), a] := \text{Total}$ ;
22.      Prev-Support $[(i, j), a] := \{(i, x) | (i, x) \in E \wedge (x, j) \in \text{Prev-Edge}_j\}$ 
23.         $\cup \{(i, j) | (i, j) \in \text{Prev-Edge}_j\}$ 
24.         $\cup \{(i, \text{start}) | (\text{start}, j) \in \text{Prev-Edge}_j\}$ ;
25.      Next-Support $[(i, j), a] := \{(i, x) | (i, x) \in E \wedge (j, x) \in \text{Next-Edge}_j\}$ 
26.         $\cup \{(i, j) | (j, i) \in \text{Next-Edge}_j\}$ 
27.         $\cup \{(i, \text{end}) | (j, \text{end}) \in \text{Next-Edge}_j\}$ ;
28.      Local-Prev-Support( $i, a$ ) :=  $\{(i, x) | (i, x) \in E \wedge (x, i) \in \text{Prev-Edge}_i\}$ 
29.         $\cup \{(i, \text{start}) | (\text{start}, i) \in \text{Prev-Edge}_i\}$ ;
30.      Local-Next-Support( $i, a$ ) :=  $\{(i, x) | (i, x) \in E \wedge (i, x) \in \text{Next-Edge}_i\}$ 
31.         $\cup \{(i, \text{end}) | (i, \text{end}) \in \text{Next-Edge}_i\}$ ; }

```



Prev-Support $[(1, 2), a] = \{(1, 2)\}$	Next-Support $[(1, 2), a] = \{(1, 3)\}$
Prev-Support $[(1, 3), a] = \{(1, 2), (1, 3)\}$	Next-Support $[(1, 3), a] = \{(1, \text{end})\}$
Prev-Support $[(1, 2), b] = \{(1, 2)\}$	Next-Support $[(1, 2), b] = \{(1, 3)\}$
Prev-Support $[(1, 3), b] = \{(1, 2), (1, 3)\}$	Next-Support $[(1, 3), b] = \{(1, \text{end})\}$
Prev-Support $[(2, 1), c] = \{(2, \text{start})\}$	Next-Support $[(2, 1), c] = \{(2, 1), (2, 3)\}$
Prev-Support $[(2, 3), c] = \{(2, 3), (2, 1)\}$	Next-Support $[(2, 3), c] = \{(2, \text{end})\}$
Prev-Support $[(3, 1), d] = \{(3, \text{start})\}$	Next-Support $[(3, 1), d] = \{(3, 1), (3, 2)\}$
Prev-Support $[(3, 2), d] = \{(3, 1)\}$	Next-Support $[(3, 2), d] = \{(3, 2)\}$
Local-Prev-Support( $1, a$ ) = $\{(1, \text{start})\}$	Local-Next-Support( $1, a$ ) = $\{(1, 2), (1, 3)\}$
Local-Prev-Support( $1, b$ ) = $\{(1, \text{start})\}$	Local-Next-Support( $1, b$ ) = $\{(1, 2), (1, 3)\}$
Local-Prev-Support( $2, c$ ) = $\{(2, 1)\}$	Local-Next-Support( $2, c$ ) = $\{(2, 3)\}$
Local-Prev-Support( $3, d$ ) = $\{(3, 1), (3, 2)\}$	Local-Next-Support( $3, d$ ) = $\{(3, \text{end})\}$

Figure 24: Algorithm for initializing the SLCN arc consistency data structures along with a simple example. The dotted lines are members of the set  $E$ .

```

1.  while List  $\neq \phi$  do {
2.      Pop [(j, i), b] from List;
3.      for (i, a)  $\in$  S[(j, i), b] do {
4.          Counter[(i, j), a] := Counter[(i, j), a] - 1;
5.          if Counter[(i, j), a] = 0  $\wedge$  M[(i, j), a] = 0 then {
6.              List := List  $\cup$  {(i, j), a};
7.              M[(i, j), a] := 1; } }
8.      for (j, x)  $\in$  Prev-Support[(j, i), b]  $\wedge$  x  $\neq$  i  $\wedge$  x  $\neq$  start do {
9.          Remove (j, x) from Prev-Support[(j, i), b];
10.         Next-Support[(j, x), b] := Next-Support[(j, x), b] - {(j, i)};
11.         if Next-Support[(j, x), b] =  $\phi$   $\wedge$  M[(j, x), b] = 0 then {
12.             List := List  $\cup$  {(j, x), b};
13.             M[(j, x), b] := 1; } }
14.         for (j, x)  $\in$  Next-Support[(j, i), b]  $\wedge$  x  $\neq$  i  $\wedge$  x  $\neq$  end do {
15.             Remove (j, x) from Next-Support[(j, i), b];
16.             Prev-Support[(j, x), b] := Prev-Support[(j, x), b] - {(j, i)};
17.             if Prev-Support[(j, x), b] =  $\phi$   $\wedge$  M[(j, x), b] = 0 then {
18.                 List := List  $\cup$  {(j, x), b};
19.                 M[(j, x), b] := 1; } }
20.         if (i, j)  $\in$  Prev-Edgej then
21.             Local-Prev-Support(j, b) := Local-Prev-Support(j, b) - {(j, i)};
22.         if Local-Prev-Support(j, b) =  $\phi$  then {
23.             Lj := Lj - {b};
24.             for (j, x)  $\in$  Local-Next-Support(j, b)  $\wedge$  x  $\neq$  i  $\wedge$  x  $\neq$  end do {
25.                 Remove (j, x) from Local-Next-Support(j, b);
26.                 if M[(j, x), b] = 0 then {
27.                     List := List  $\cup$  {(j, x), b};
28.                     M[(j, x), b] := 1; } } }
29.         if (j, i)  $\in$  Next-Edgej then
30.             Local-Next-Support(j, b) := Local-Next-Support(j, b) - {(j, i)};
31.         if Local-Next-Support(j, b) =  $\phi$  then {
32.             Lj := Lj - {b};
33.             for (j, x)  $\in$  Local-Prev-Support(j, b)  $\wedge$  x  $\neq$  i  $\wedge$  x  $\neq$  start do {
34.                 Remove (j, x) from Local-Prev-Support(j, b);
35.                 if M[(j, x), b] = 0 then {
36.                     List := List  $\cup$  {(j, x), b};
37.                     M[(j, x), b] := 1; } } } }

```

Figure 25: Algorithm to enforce SLCN arc consistency.

$\{(1,2), (1,3)\}$  and  $\text{Next-Support}[(1,3), a] = \{(1, \mathbf{end})\}$ . Note that the value  $(1, \mathbf{end})$  in  $\text{Next-Support}[(1,3), a]$  and the value  $(1,3)$  in  $\text{Prev-Support}[(1,3), a]$ , once eliminated from those sets, require no further action because they are dummy values. However, the value  $(1,2)$  in  $\text{Prev-Support}[(1,3), a]$  indicates that  $(1,3)$  is a member of  $\text{Next-Support}[(1,2), a]$ , and since  $a$  is not admissible for  $(1,3)$ ,  $(1,3)$  should be removed from  $\text{Next-Support}[(1,2), a]$ , leaving an empty set. Note that because  $\text{Next-Support}[(1,2), a]$  is empty, and assuming that  $M[(1,2), a] = 0$ ,  $[(1,2), a]$  is added to *List* for further processing. Next,  $(1,3)$  is removed from  $\text{Local-Next-Support}(1, a)$ , but that set is non-empty. During the next iteration of the while loop,  $[(1,2), a]$  is popped from *List*. When  $\text{Prev-Support}[(1,2), a]$  and  $\text{Next-Support}[(1,2), a]$  are processed,  $\text{Next-Support}[(1,2), a] = \phi$  and  $\text{Prev-Support}[(1,2), a]$  contains only a dummy, which is removed. When  $(1,2)$  is removed from  $\text{Local-Next-Support}(1, a)$ , the set becomes empty, so  $a$  is no longer compatible with any sentence containing role 1 and can be eliminated from further consideration as a possible role value for role 1. Once  $a$  is eliminated from role 1, it is also necessary to remove the support of  $a \in L_1$  from all role values on roles that precede role 1, that is for all roles  $x$  such that  $(1, x) \in \text{Local-Prev-Support}(1, a)$ . Since  $\text{Local-Prev-Support}(1, a) = \{(1, \mathbf{start})\}$ , and  $\mathbf{start}$  is a dummy role, there is no more work to be done.

In contrast, consider what happens if initially  $[(1,2), a] \in \text{List}$  for the SLCN in Figure 24. In this case,  $\text{Prev-Support}[(1,2), a]$  contains  $(1,2)$  which requires no additional work; whereas,  $\text{Next-Support}[(1,2), a]$  contains  $(1,3)$ , indicating that  $(1,2)$  must be removed from  $\text{Prev-Support}[(1,3), a]$ 's set. After the removal,  $\text{Prev-Support}[(1,3), a]$  is non-empty, so the sentence containing roles 1 and 3 still supports the role value  $a$  in  $L_1$ . The reason that these two cases provide different results is that roles 1 and 3 are in every sentence; whereas, roles 1 and 2 are in only one of them.

### 4.3.1 The Running Time of the SLCN Filtering Algorithm

The running time of the routine to initialize the filtering data structures (in Figure 24) is  $O(n^2l^2 + n^3l)$ , where  $n$  is the total number of roles in an SLCN and  $l$  is the number of role values assigned to each role. Given that the number of  $(i, j)$  elements in  $E$  is  $O(n^2)$  and the domain size is  $O(l)$ , there are  $O(n^2l)$  counters and S sets for which to calculate values. To determine the number of supporters for a given arc-role value pair requires  $O(l)$  work; hence, initializing all of the counters and S sets requires  $O(n^2l^2)$  time. However, there are  $O(n^2l)$   $\text{Prev-Support}$  and  $\text{Next-Support}$  sets, where each  $\text{Prev-Support}[(i, j), a]$  and  $\text{Next-Support}[(i, j), a]$  requires  $O(n)$  time to compute, so the time to calculate all  $\text{Prev-Support}$  and  $\text{Next-Support}$  sets is  $O(n^3l)$ . Finally, the time needed to calculate all  $\text{Local-Next-Support}$  and  $\text{Local-Prev-Support}$  sets is  $O(n^2l)$  because there are  $O(nl)$  sets with up to  $O(n)$  elements per set.

The running time for the algorithm which prunes role values that are not SLCN arc consistent

(in Figure 25) also operates in  $O(n^2l^2 + n^3l)$  time. Clearly there are only  $O(n^2l)$  counters (and similarly S sets) to keep track of in the algorithm. Each counter can be at most  $l$  in magnitude, and, it can never become negative, so the maximum running time for line 4 in Figure 25 (given that elements appear on the list only once because of M) is  $O(n^2l^2)$ . Because there are  $O(n^2l)$  Next-Support and Prev-Support Lists, each up to  $O(n)$  in size, the running time required for lines 10 and 16 in Figure 25 is  $O(n^3l)$ . Finally, since there are  $O(nl)$  Local-Prev-Support and Local-Next-Support sets from which to eliminate  $O(n)$  elements, the running time of lines 21 and 30 in Figure 25 is  $O(n^2l)$ . Hence, the running time of the SLCN arc consistency algorithm is  $O(n^2l^2 + n^3l)$ . By comparison, the running time for CDG arc consistency is  $(n^2l^2)$ . Note that for this application  $l = n$ , so the running times of the algorithms are the same order.

### 4.3.2 Correctness of the SLCN Arc Consistency Algorithm

Next we prove the correctness of the SLCN arc consistency algorithm. A role value is eliminated from a role's domain by our algorithm (see lines 23 and 32 in Figure 25) if and only if its Local-Prev-Support set or its Local-Next-Support set becomes empty (see lines 22 and 31 in Figure 25). In either case, the role value can never appear in any valid parse for a sentence. We prove that if a role value's local support sets become empty, that role value cannot participate in any valid parse for the sentence. This is proven for Local-Next-Support (Local-Prev-Support follows by symmetry.) Observe that if  $a \in L_i$ , and it is incompatible with all of the roles which immediately follow  $L_i$  in the DAG, then it cannot participate in any valid parse. In line 30 of Figure 25,  $(i, j)$  is removed from Local-Next-Support( $i, a$ ) set if and only if  $[(i, j), a]$  has been popped off *List*. The removal of  $(i, j)$  from Local-Next-Support( $i, a$ ) indicates that the sentences containing  $i$  and  $j$  do not support  $a \in L_i$ . It remains to be shown that  $[(i, j), a]$  is put on *List* if only if  $a \in L_i$  is incompatible with every sentence which contains  $i$  and  $j$ . This is proven by induction on the number of iterations of the while loop in Figure 25.

**Base case:** The initialization routine only puts  $[(i, j), a]$  on *List* if  $a \in L_i$  is incompatible with every role value in  $L_j$  (line 17 of Figure 24). Therefore,  $a \in L_i$  is in no parse for any sentences which contain  $i$  and  $j$ .

**Induction step:** Assume that at the start of the  $k$ th iteration of the while loop all  $[(x, y), c]$  which have ever been put on *List* indicate that  $c \in L_x$  is incompatible with every sentence which contains  $x$  and  $y$ . It remains to show that during the  $k$ th iteration, if  $[(i, j), a]$  is put on *List*, then  $a \in L_i$  is incompatible with every sentence which contains  $i$  and  $j$ . There are four ways in which a new  $[(i, j), a]$  can be put on *List*:

1. All role values in  $L_j$  which were once compatible with  $a \in L_i$  have been eliminated. This item could have been placed on *List* either during initialization (see line 17 in Figure 24) or

during a previous iteration of the while loop (see line 6 in Figure 25)), just as in the CSP AC-4 algorithm. It is obvious that, in this case,  $a \in L_i$  is incompatible with every sentence containing  $i$  and  $j$ .

2.  $\text{Prev-Support}[(i, j), a] = \phi$  (see line 17 in Figure 25) indicating that  $a \in L_i$  is incompatible with all roles  $k$  for  $(k, j) \in \text{Prev-Edge}_j$ . The only way for  $[(i, j), a]$  to be placed on *List* for this reason (at line 18) is because all tuples of the form  $[(i, k), a]$  (where  $(k, j) \in \text{Prev-Edge}_j$ ) were already put on *List*. By the induction hypothesis, these  $[(i, k), a]$  items were placed on the *List* because  $a \in L_i$  is incompatible with all sentences containing  $i$  and  $k$  in the DAG. But if  $a$  is incompatible with every role which immediately precedes  $j$  in the DAG, then  $a$  is incompatible with every sentence which contains  $j$ . Therefore, it is correct to put  $[(i, j), a]$  on *List*.
3.  $\text{Next-Support}[(i, j), a] = \phi$  (see line 11 in Figure 25) indicating that  $a \in L_i$  is incompatible with all roles  $k$  for  $(j, k) \in \text{Next-Edge}_j$ . The only way for  $[(i, j), a]$  to be placed on *List* (at line 12) for this reason is because all tuples of the form  $[(i, k), a]$  (where  $(j, k) \in \text{Next-Edge}_j$ ) were already put on *List*. By the induction hypothesis, these  $[(i, k), a]$  items were placed on the *List* because  $a \in L_i$  is incompatible with all sentences containing  $i$  and  $k$  in the DAG. But if  $a$  is incompatible with every role value in every role which immediately follows  $j$  in the DAG, then  $a$  is incompatible with every segment which contains  $j$ . Therefore, it is correct to put  $[(i, j), a]$  on *List*.
4.  $\text{Local-Next-Support}(i, a) = \phi$  (see line 31 in Figure 25) indicating that  $a \in L_i$  is incompatible with all roles  $k$  such that  $(i, k) \in \text{Next-Edge}_i$ . The only way for  $[(i, j), a]$  to be placed on *List* (at line 36) for this reason is because no role which follows  $i$  in the DAG supports  $a$ , and so all pairs  $(i, k)$  have been legally removed from  $\text{Local-Next-Support}(i, a)$  during previous iterations. Because there is no sentence containing  $i$  which supports  $a$ , it follows that no sentence containing  $i$  and  $j$  supports that role value.

At the beginning of the  $(k + 1)$ th iteration of the while loop, every  $[(x, y), c]$  on *List* implies that  $c$  is incompatible with every sentence which contains  $x$  and  $y$ . Therefore, by induction, it is true for all iterations of the while loop in Figure 25. Hence, when a role value's local support sets become empty, that label cannot participate in a valid parse for the sentence.

The result of our algorithm is SLCN arc consistent. That is for all edges  $(i, j)$ , for all role values  $a \in L_i$ , if  $\text{Counter}[(i, j), a] > 0$ , and  $\text{Prev-support}[(i, j), a] \neq \phi$ ,  $\text{Next-support}[(i, j), a] \neq \phi$ ,  $\text{Local-Prev-support}(i, a) \neq \phi$  and  $\text{Local-Next-support}(i, a) \neq \phi$ , then  $a$  is supported by some role value in  $j$ , and is SLCN arc consistent in at least one sentence containing  $(i, j)$ . From the previous

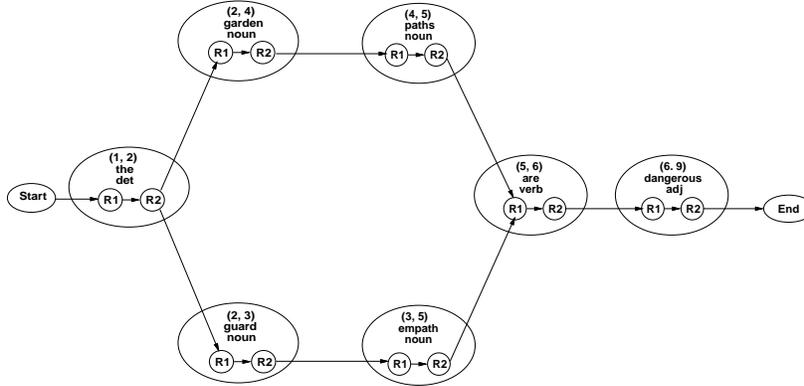


Figure 26: An SLCN with multiple roles compatible with our new algorithm.

induction proof and this, we may conclude that our algorithm builds the largest SLCN arc consistent structure.

### 4.3.3 Multiple Roles

As shown in Figure 20, an SLCN can have more than a single role. In our initial development of the SLCN arc consistency algorithm discussed in [7], we distinguished between two types of arcs: *intra-arcs*, which are arcs joining roles within the same word node, and *inter-arcs*, which are arcs joining roles across word nodes. The inter-arcs were processed in the same way as in the algorithm in Figures 24 and 25, but the intra-arcs were handled differently. They require special handling because if a role value is disallowed for role  $x$  by a role  $y$  within the same word node as  $x$ , then the role value must be eliminated from  $x$ . In this case, the elimination of the role value from  $x$  is correct because every sentence containing the word node disallows the role value.

Special handling of this case complicates the arc consistency algorithm; however, there is a simpler way to accomplish precisely the same effect as the special case while using the algorithm shown in Figures 24 and 25. It simply involves setting up the SLCN in a slightly different way than in Figure 20. The edges in the SLCN in Figure 20 relate the roles across nodes but not roles within the same word node. If we assume that the roles within a word node are connected by directed edges as in Figure 26, then the arc consistency algorithm in Figure 25 is sufficient for SLCNs with more than a single role. Because there is a single linear list of roles within a word node, they must appear in all the same sentences, and so if one role value is disallowed by one of the roles in the linear list of roles, it is eliminated from the role. This is because every sentence containing the first role must also contain the other role (i.e., there are no alternative paths that include both roles). Hence, we set up SLCNs with more than a single role as shown in Figure 26 and use the simpler arc consistency algorithm described in this paper.

1. Clear all windows.
2. Clear windows.
3. Clear all the windows.
4. Get all windows.
5. Give all windows.
6. Clear all of the windows.
7. Clear the windows.
8. Get all the windows.
9. Give all the windows.
10. Get all of the windows.
11. Give all of the windows.

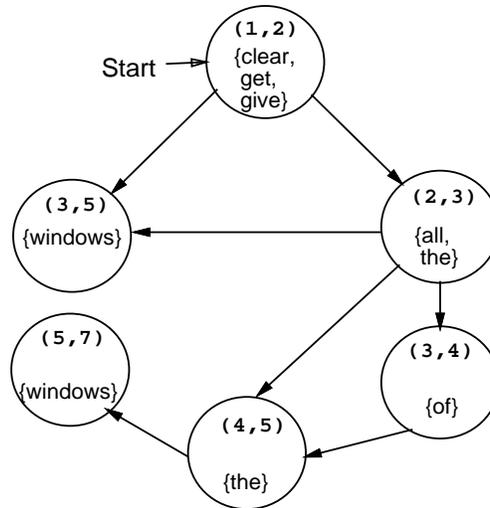


Figure 27: A word graph constructed from a list of N-best sentences.

#### 4.4 An SLCN Parsing Experiment

We have developed a C++ implementation of a CDG parser, PARSEC, which is capable of parsing CNs or SLCNs on a Sun workstation [8]. The software allows users to specify parameters of the grammar, to design features, and to write unary and binary constraints. It also checks the constraints for correctness of form given a grammar's parameters. Once a grammar is written and constraints are checked for well-formedness, the parser is ready to parse a sentence. When parsing a sentence, it first applies the core set of unary constraints, then it constructs arcs and arc matrices and propagates the core binary constraints. At this point, an X-windows interface for the parser appears on the screen. Figure 28 depicts the interface to the SLCN constructed from the word graph in Figure 27.

Once the window is present, the user can choose from several menu options. Among the options available are: perform a single step of filtering, filter completely, view the arcs joining the roles of two word nodes, apply constraints from a file, print node information to a file, and print arc information to a file. The user can also view the state of the parse. This interface allows the remaining role values for each word candidate's roles to be viewed by clicking on the word in the word node. For example, the role values for the three roles for the word *windows* over the interval (3,5) in Figure 28 are viewed by clicking on that word. This interface also allows viewing of the matrices stored on each of the arcs in the network. The arcs joining the roles of two words are displayed when a user selects the appropriate menu item along with two word nodes. Figure 29 shows the arcs joining the roles for the words *of* and *the*. The matrix associated with each of the arcs can then be viewed by clicking on its arc. Figure 30 shows the matrix for the arc joining the governor roles of the two words. This implementation of the SLCN parser provides useful tools for

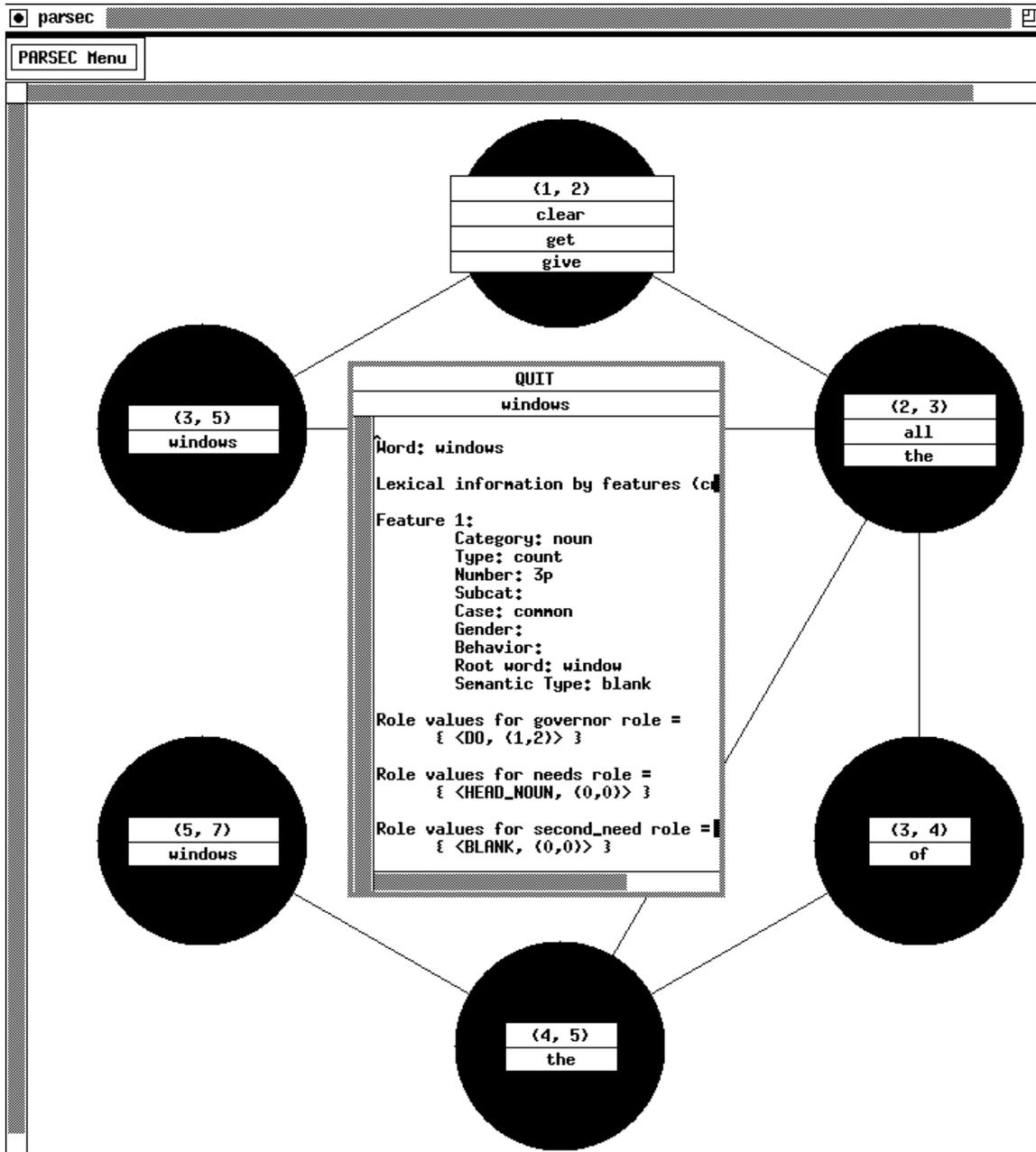


Figure 28: The X-window interface to the SLCN for the word graph in Figure 31.

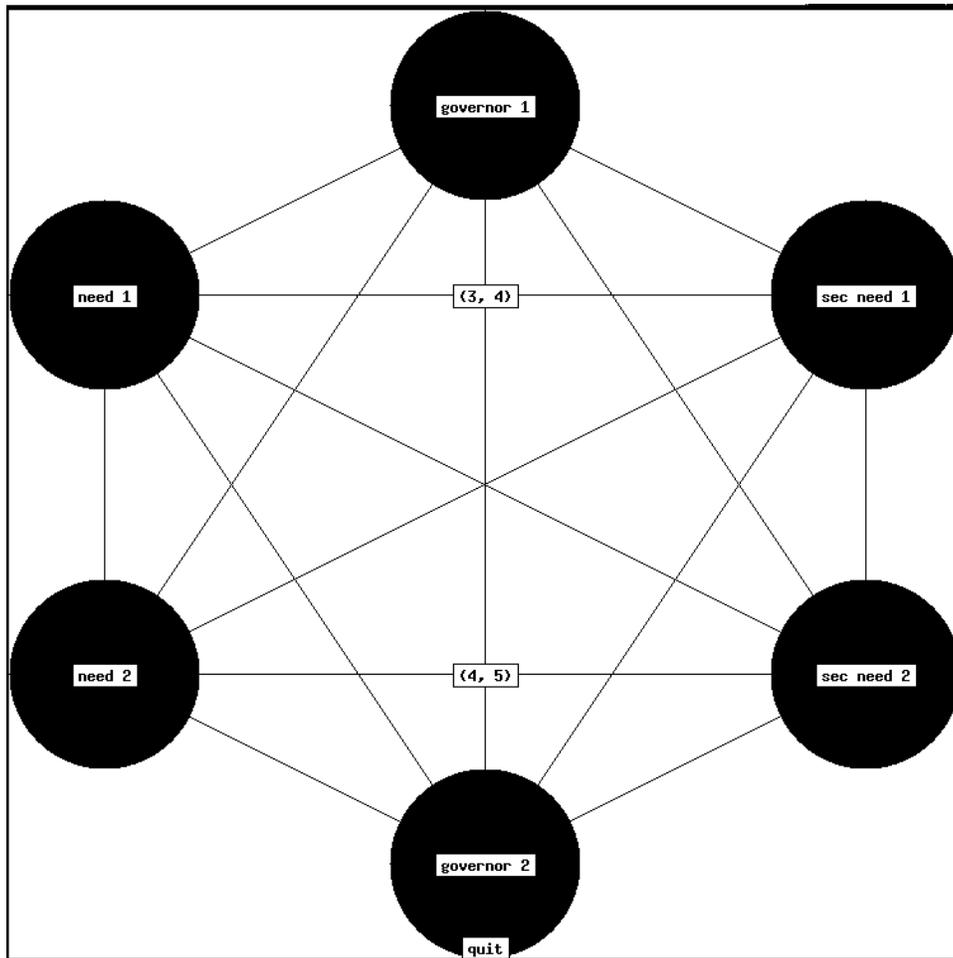


Figure 29: The X-window interface to the arcs connecting the roles of *of* and *the*.

```

QUIT
^
Arc Matrix connecting governor role of node 3 and governor role of node 4.
<DET, (5,7)>
| <DET, (5,7)>
| | <DET, (5,7)>
| | | <DET, (5,7)>
X 1 X X
[ X X X X ] X <N_PP, (1,2)>
[ X X X X ] X <N_PP, (2,3)>
[ X 1 X X ] 1 <V_PP, (1,2)>
[ X X X X ] X <V_PP, (2,3)>
[ X X X X ] X <DET_PP, (1,2)>
[ X 1 X X ] 1 <DET_PP, (2,3)>

```

Figure 30: The X-window interface to the arc matrix for the governor roles of *of* and *the*.

developing and testing constraint-based grammars.

In order to demonstrate the effectiveness of our SLCN parser, we have developed two grammars. The first grammar was designed to parse sets of sentences in the resource management database [35]. The second grammar covered sentences in the ATIS database [14, 34] (Air Travel Information System). These experiments simply demonstrate that constraints on a word graph can prune the search space of illegal sentence parses and words that do not appear in a parse. We cannot discuss the impact on word recognition since we do not currently have a word recognizer which constructs word graphs with word scores; we constructed our word graphs from lists of N-Best sentences provided by BBN. However, we were able to ascertain that the constraint parser effectively prunes ungrammatical sentence hypotheses from the word graphs. In addition, we were able to demonstrate that semantic constraints could easily be added to grammars without modifying the syntactic constraints.

We first developed constraints using the grammar templates for the resource management database task. The resulting constraint grammar contains 3 roles, 11 categories, 70 labels, 100 unary constraints, and 200 binary constraints, which are capable of parsing statements, yes-no questions, commands, and wh-questions [56]. The constraints consist of phrase structure rules and feature constraints. The feature constraints enforce subject-verb agreement, determiner-head-noun agreement, and case restrictions on pronouns. Additionally, the subcategorization feature is used to make certain that a verb has the appropriate set of objects and complements to be complete. The lexicon used along with this grammar contains many lexically ambiguous words, and its word entries contain the necessary feature information to support our feature constraints.

To demonstrate the effectiveness of syntactic constraints for eliminating spurious sentence hypotheses from a word graph, we converted three lists of N-best sentences provided to us by BBN for the resource management task into word graphs. Each word graph was constructed by using the syllable count for each word as its duration and maintaining the syllable count through the utterance. Next, the word graphs were converted into SLCNs and parsed using our constraint grammar. More grammatical sentences were parsed in each SLCN than were available in the original sets of sentences, as can be seen in Table 5; however, all of the additional parses had a form similar to one of the original grammatical N-best sentences. For example, the SLCN depicted in Figure 28, constructed from the word graph in Figure 27, contains three verbs: *clear*, *give*, and *get*. Each is the main verb in 5 minor variations of the same sentence, each with a legal parse. In contrast, only 11 of the 15 forms appear in the N-Best list used to construct the SLCN. Note that none of the paths corresponding to ungrammatical sentences had legal parses.

Syntactic constraints were effective at pruning a word graph of ungrammatical sentence hypotheses and limiting the possible parses for the remaining sentences. However, it is often the case

<b>Sentence Type</b>	<b>Number Grammatical Sentences in N-best</b>	<b>Number Grammatical Sentences in SLCN</b>
<b>Command</b>	11	15
<b>Yes-No-Q</b>	8	12
<b>Wh-Q</b>	7	16

Table 5: **N-best versus SLCN sentence parses.**

that syntactic information alone is insufficient for selecting a single sentence hypothesis from a word graph. Effective use of multiple knowledge sources plays a key role in human spoken language understanding. It is, therefore, likely that advances in spoken language understanding will require effective utilization of higher level knowledge.

To demonstrate the flexibility of constraint-based parsing for utilizing a variety of knowledge sources, we have incorporated semantic constraints into our parser. To do so, we developed a second grammar for sentences in the ATIS database (Air Travel Information System), which was chosen because of its greater degree of semantic richness. Unfortunately, we did not have grammar templates for this task, and so had to design a syntactic constraints from a list of correct sentences for the task. These constraints were merged with the constraints for the resource management grammar. It was actually quite easy to add additional grammar constraints to the previously developed grammar.

Next, semantic constraints were constructed to further limit ambiguity [10]. Semantic constraints were relatively easy to create and incorporate into our parser because they were based on semantic feature testing in certain syntactic configurations. For example, some constraints limited the semantic type associated with a prepositional phrase based on the semantic type of its object, and others limited the cites of attachment for a prepositional phrase based on semantic type compatibility. We then conducted a simple experiment to compare the effectiveness of syntactic and semantic constraints for reducing the ambiguity of word networks constructed from sets of BBN’s N-best sentence hypotheses [41] from the ATIS database.

For this experiment, we selected twenty sets of 10-best sentence hypotheses for three different types of utterances: a command, a yes-no question, and a wh-question. The lists of the N-best sentences were converted to word graphs using syllable count. These word graphs were then converted to SLCNs and parsed with the constraints. We determined for each eliminated word candidate whether a syntactic constraint or a semantic constraint was responsible for the deletion. Syntactic and semantic constraints together were very effective at reducing the number of parses for sentences in the SLCN when compared with syntactic constraints alone. However, syntactic constraints alone

played the major role in pruning inappropriate word candidates from the network. On the average, syntactic constraints alone eliminated 3.11 word candidates per SLCN; whereas, semantic constraints, when applied after syntactic constraints, eliminated an average of .66 additional word candidates per SLCN<sup>11</sup>.

These experiments were performed to demonstrate that constraints are effective at pruning ungrammatical sentence hypotheses from a word graph. As we do not currently have a word recognizer which constructs word graphs with word scores, we cannot discuss how well our parser reduces recognition error. However with a word recognizer, we would be able to focus on the valid parses for the most likely paths through the SLCN. We are currently developing a word recognizer that constructs word graphs for our parser.

#### 4.5 Search Strategies in SLCN Parsing

Parses in single sentence CNs can be generated by using backtracking (or fancier search algorithms) to assemble a set of role values, one for each role, which are consistently admissible. Extracting parses from an SLCN can be done in a similar way, but it is desirable to make a few modifications to the search algorithms to take advantage of the extra information which is contained in the SLCN arc consistency data structures. In particular, the Local-Prev-Support( $i, a$ ) and Local-Next-Support( $i, a$ ) sets contain information about which roles following role  $i$  support role  $a$ . If we select  $a$  as a value for  $i$ , then we can use this information to avoid selecting values for adjacent nodes which are incompatible with  $a$ . However, just looking at the local support sets ignores some other useful information provided by the Prev and Next support sets. By restricting search to those that are viable given the local support sets, and then selecting the role  $j$  to expand by using the Next-Support[( $i, j$ ),  $a$ ] and Prev-Support[( $i, j$ ),  $a$ ] sets will more effectively guide the search toward a parse.

Word recognition scores can be used to drive the application of constraints and the arc consistency algorithm along the most likely path given word scores. It makes sense to apply unary constraints to all nodes and achieve node consistency since this is a fairly inexpensive operation. However, binary constraint propagation and arc consistency are more expensive. Clearly, we can apply the binary constraints to all pairs of roles along the most likely path and attempt to pull out a parse along that path. By doing this, we can avoid propagating constraints on arcs to words not on our path of words. The untouched words that share arcs with the target words still appear to support all the role values for the roles along our path, but the support sets can be used to find a solution along the path if there is one. If there is no solution along the best path, we can select

---

<sup>11</sup>Since most semantic rules use some syntactic information, it makes sense to propagate the syntactic constraints before the semantic constraints. When we propagate the semantic constraints first, no word candidates are typically eliminated because of the high ambiguity in the CN without syntactic constraints.

another, with possibly overlapping words, from which to extract a solution. Note that we do not have to repropagate constraints between roles that have already been modified.

## 5 Conclusion

SLCN parsing has several advantages that make it attractive for speech. First, it provides a flexible uniform framework for using lexical, syntactic, semantic, prosodic, and contextual constraints to incrementally reduce the ambiguity found in a word graph provided by a speech recognition system (We have already developed lexical, syntactic, and semantic constraints for our parser (see Section 4.4) and are currently developing prosodic constraints). Second, it is able to handle grammars that are beyond context-free. Third, the parser is able to support the use of context when determining the meaning of a sentence. Fourth, the flexibility of incremental constraint-based parsing should allow us to develop strategies for reducing sensitivity to the syntactic irregularities common in spontaneous speech. Current spoken language recognition systems are not as accurate as humans, in part, because they do not utilize the wide range of information that people do when understanding speech. Hence, we believe that further investigations along these lines will result in more effective processing of speech.

The filtering algorithm developed in this paper is useful, not only for processing speech, but also for other CSP problems. Up to this time, CSP arc consistency has always assumed perfect segmentation of input. Speech recognition is only one area where segmenting the signal into higher-level chunks is problematic. Vision systems and handwriting analysis systems have a comparable problem.

## 6 Acknowledgments

This work was supported in part by Purdue Research Foundation, NSF grant number IRI-9011179, and NSF Parallel Infrastructure Grant CDA-9015696. We thank BBN for providing us with the N-best lists of sentences. We would especially like to thank those students who were involved in the implementation of PARSEC: Yin Chan, Mark Rowland, Todd Stewart, Christopher White, Boon Lock Yeo, and Carla Zoltowski. We would also like to thank Carl Mitchell, Carla Zoltowski, and Leah Jamieson for their encouragement and comments on various drafts of this paper.

## References

- [1] M. A. Covington. A parsing algorithm that extends phrases. *Computational Linguistics*, 4:234–236, 1990.

- [2] P. Dey and B. R. Bryant. Lexical ambiguity in tree adjoining grammars. *Information Processing Letters*, 34:65–69, 1990.
- [3] J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13:94–102, 1970.
- [4] L. D. Erman and V. R. Lesser. The Hearsay-II speech understanding system: A tutorial. In W. A. Lea, editor, *Trends in Speech Recognition*, pages 361–381. Speech Science Publications, Apple Valley, MN, 1986.
- [5] E. P. Giachin. Automatic training of stochastic finite-state language models for speech understanding. In *IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, March 1992.
- [6] S. L. Graham, M. A. Harrison, and W. L. Ruzzo. An improved context-free recognizer. *ACM Transactions on Programming Languages and Systems*, 2(3):415–462, 1980.
- [7] M. P. Harper and R. A. Helzerman. PARSEC: A constraint-based parser for spoken language parsing. Technical Report EE-93-28, Purdue University, School of Electrical Engineering, West Lafayette, IN, 1993.
- [8] M. P. Harper, R. A. Helzerman, C. B. Zoltowski, B. L. Yeo, Y. Chan, T. Stewart, and B. L. Pellom. Implementation issues in the development of the parsec parser. *SOFTWARE – Practice and Experience*, to appear.
- [9] M. P. Harper, L. H. Jamieson, C. D. Mitchell, G. Ying, S. Potisuk, P. N. Srinivasan, R. Chen, C. B. Zoltowski, L. L. McPheters, B. Pellom, and R. A. Helzerman. Integrating language models with speech recognition. In *The Proceedings of the AAAI-94 Workshop on Integration of Natural Language and Speech Processing*, 1994.
- [10] M. P. Harper, L. H. Jamieson, C. B. Zoltowski, and R. A. Helzerman. Semantics and constraint parsing of word graphs. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, volume II, pages 63–66, April 1992.
- [11] R. A. Helzerman. PARSEC: A framework for parallel natural language understanding. Master’s thesis, Purdue University, School of Electrical Engineering, West Lafayette, IN, 1993.
- [12] R. A. Helzerman and M. P. Harper. Log time parsing on the MasPar MP-1. In *Proceedings of the Sixth International Conference on Parallel Processing*, August 1992.
- [13] Randall A. Helzerman and Mary P. Harper. An approach to multiply-segmented constraint satisfaction problems. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 350–355, 1994.
- [14] C. T. Hemphill, J. J. Godfrey, and G. R. Doddington. The ATIS spoken language systems pilot corpus. Technical Report NTIS PB91-505354, 1990. NIST Speech Disc 5-1.1.
- [15] F. Jelinek. Self-organized language modeling for speech recognition. In Alex Waibel and Kai-Fu Lee, editors, *Readings in Speech Recognition*. Morgan Kaufman Publishers, Inc., San Mateo, CA, 1990.

- [16] A. K. Joshi, L.S. Levy, and M. Takahashi. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10:136–163, 1975.
- [17] M. Kay. The MIND system. In R. Rustin, editor, *Natural Language Processing*. Algorithmics Press, New York, 1973.
- [18] S. R. Kosaraju. Speed of recognition of context-free languages by array automata. *SIAM Journal of Computing*, 4(3) :331–340, September 1975.
- [19] V. Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 13(1):32–44, 1992.
- [20] C.-H. Lee, E. Giachin, L.R. Rabiner, R. Pieraccini, and A. E. Rosenburg. Improved acoustic modeling for large vocabulary continuous speech recognition. *Computer Speech and Language*, 6(2):103–127, 1992.
- [21] K. F. Lee, H. W. Hon, and R. Reddy. An overview of the SPHINX speech recognition system. In *IEEE Transactions on Acoustic, Speech, Signal Processing*, pages 35–45, January 1990.
- [22] K.F. Lee. *The Development of the SPHINX System*. Kluwer Academic Publishers, 1989.
- [23] V. R. Lesser, R. D. Fennell, L. D. Erman, and D.R. Reddy. Organization of the Hearsay-II speech understanding system. *IEEE Trans. Acoust., Speech, Signal Processing*, ASSP-23:11–23, 1975.
- [24] H. Maruyama. Constraint dependency grammar. Technical Report #RT0044, IBM, Tokyo, Japan, 1990.
- [25] H. Maruyama. Constraint dependency grammar and its weak generative capacity. *Computer Software*, 1990.
- [26] H. Maruyama. Structural disambiguation with constraint propagation. In *The Proceedings of the Annual Meeting of ACL*, 1990.
- [27] J. T. Maxwell and R. M. Kaplan. The interface between phrasal and functional constraints. *Computational Linguistics*, 19:571–590, 1993.
- [28] R. Mohr and T. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28, 1986.
- [29] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Science*, 1976.
- [30] M. D. Moshier and W. C. Rounds. On the succinctness properties of unordered context-free grammars. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, 1987.
- [31] M. A. Palis, S. Shende, and D. S. L. Wei. An optimal linear-time parallel parser for tree adjoining languages. *SIAM Journal of Computing*, 19:1–31, 1990.
- [32] D. B. Paul. An efficient A\* stack decoder algorithm for continuous speech recognition with a stochastic language model. In *IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, March 1992.
- [33] F. C. N. Pereira and D. H. D. Warren. Definite clause grammars for language analysis– A survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13(3):231–278, 1980.

- [34] P. J. Price. Evaluation of spoken language systems: The ATIS domain. In *Proceedings of the DARPA Workshop on Speech and Natural Language*, pages 91–95, 1990.
- [35] P. J. Price, W. Fischer, J. Bernstein, and D. Pallett. A database for continuous speech recognition in a 1000-word domain. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, 1988.
- [36] W. Ruzzo. Tree-size bounded alternation. *Journal of Computers and System Sciences*, 21:218–235, 1980.
- [37] Y. Schabes. *Mathematical and Computational Aspects of Lexicalized Grammars*. PhD thesis, Department of Computer and Information Sciences, University of Pennsylvania, 1990.
- [38] Y. Schabes. Polynomial time and space shift-reduce parsing of arbitrary context-free grammars. In *The Proceedings of the Annual Meeting of ACL*, 1991.
- [39] R. Schwartz and S. Austin. A comparison of several approximate algorithms for finding multiple N-best sentence hypotheses. In *IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, May 1991.
- [40] R. Schwartz, Y. Chow, O. Kimball, S. Roucos, M. Krasner, and J. Makhoul. Context-dependent modeling for acoustic-phonetic recognition of continuous speech. In *IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, March 1985.
- [41] R. Schwartz and Y-L. Chow. The N-best algorithm: An efficient and exact procedure for finding the N most likely sentence hypotheses. In *IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, April 1990.
- [42] S. Seneff. Robust parsing for spoken language systems. In *IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, March 1992.
- [43] S. Seneff. TINA: A natural language system for spoken language applications. *American Journal of Computational Linguistics*, 18:61–86, 1992.
- [44] J. Seo and R. F. Simmons. Syntactic graphs: A representation for the union of all ambiguous parse trees. *Computational Linguistics*, 15:19–32, 1989.
- [45] S. M. Shieber. *Constraint-based Grammar Formalisms*. MIT Press, Cambridge, MA, 1992.
- [46] M. Tomita. *Efficient Parsing for Natural Language*. Kluwer Academic Publishers, Boston, MA, 1985.
- [47] M. Tomita. An efficient word lattice parsing algorithm for continuous speech recognition. In *IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, April 1986.
- [48] K. Vijayshanker and A. K. Joshi. Some computational properties of tree adjoining grammars. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, 1986.
- [49] K. Vijayshanker, D.J. Weir, and A. K. Joshi. Characterizing structural descriptions produced by various grammatical formalisms. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, 1987.

- [50] D. L. Waltz. Understanding line drawings of scenes with shadows. In P.H. Winston, editor, *The Psychology of Computer Vision*. McGraw Hill, New York, 1975.
- [51] W. Ward. Understanding spontaneous speech: The Phoenix system. In *IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, May 1991.
- [52] J. J. Wolf and W. A. Woods. The HWIM speech understanding system. In W. A. Lea, editor, *Trends in Speech Recognition*, pages 316–339. Speech Science Publications, Apple Valley, MN, 1986.
- [53] W. A. Woods. Transition network grammars for natural language analysis. *Communications of the ACM*, 13:591–606, 1970.
- [54] W. A. Woods, M. Bates, G. Brown, B. Bruce, C. Cook, J. Klovstad, J. Makhoul, B. Nash-Webber, R. Schwartz, J. Wolf, and V. Zue. Speech understanding systems: Final technical progress report. Technical Report 3438, Bolt, Beranek, and Newman, Inc., Cambridge, MA, 1976.
- [55] J.H. Wright. LR parsing of probabilistic grammars with input uncertainty for speech recognition. *Computer Speech and Language*, 4:298–323, 1990.
- [56] C. B. Zoltowski, M. P. Harper, L. H. Jamieson, and R. A. Helzerman. PARSEC: A constraint-based framework for spoken language understanding. In *Proceedings of the International Conference on Spoken Language Understanding*, October 1992.
- [57] V. Zue, J. Glass, D. Goodine, H. Leung, M. Phillips, J. Polifroni, and S. Seneff. Integration of speech recognition and natural language processing in the MIT Voyager system. In *IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, May 1991.