MANAGING MULTIPLE KNOWLEDGE SOURCES IN CONSTRAINT-BASED PARSING OF SPOKEN LANGUAGE

Mary P. HARPER and Randall A. HELZERMAN
School of Electrical Engineering
1285 Electrical Engineering Building
Purdue University
West Lafayette, IN 47907-1285
{harper, helz}@ecn.purdue.edu

Abstract

In this paper, we describe a system which is capable of utilizing a variety of knowledge sources to select the most appropriate parse for a spoken sentence. These knowledge sources include syntax, semantics, and contextual information. We discuss one way to utilize contextual information when determining the parse for a sentence. At its simplest level, the system can be thought of as a general-purpose query answering system for multiple topical databases. The user's input would be processed by the language processor which interfaces to the databases with the goal of interacting with the correct database in order to provide a reasonable answer to the user's spoken request. Initially, it analyzes a word graph of sentence hypotheses provided by a speech recognizer using general syntactic and semantic rules. Then, if the utterance is still ambiguous, it utilizes context-specific constraints to further refine the analysis. This brings us closer to developing a more general purpose interface for multiple databases.

1 Introduction

Developing a computer model capable of understanding language (either spoken or text-based) is a difficult problem, made more difficult by the ambiguity inherent in natural languages. Ambiguity appears in many forms, including word recognition, syntax, word-sense, ambiguity of reference, and quantifier scope. Because they are often interrelated, resolving each type of ambiguity can require that the others be handled at the same time. For example, the syntactic representation of a sentence can constrain the possible antecedents for a referential noun phrase, while the antecedent of a pronoun can also constrain the sentence's syntactic representation [10].

One way to resolve ambiguity is to utilize a wide variety of knowledge sources. The knowledge sources commonly used in speech understanding are shown in Figure 1. Ef-

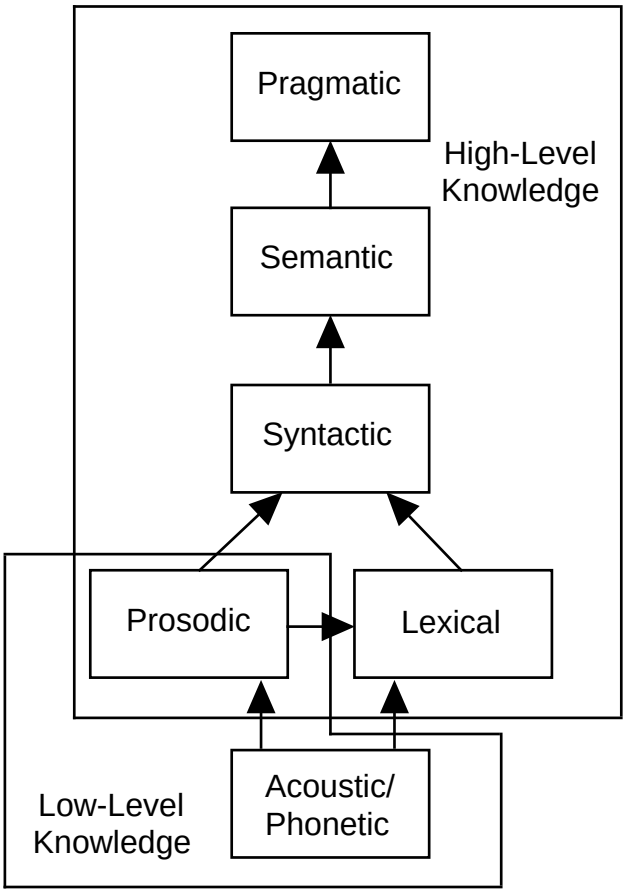


Figure 1: Knowledge sources commonly used for spoken language understanding.

fective use of multiple knowledge sources[1] plays a key role in human spoken language understanding. It is, therefore, likely that advances in spoken language understanding will require effective utilization of this information. To use the variety of knowledge sources needed to disambiguate language, we have constructed a constraint-based system [11, 14, 43] which is an extension to Constraint Dependency Grammar (CDG) parsing as defined by Maruyama [24, 25, 26]. This system is capable of propagating a wide variety of constraints, including syntactic, lexical, semantic, prosodic, and contextual constraints. The central data structure for this system is a word graph augmented with parse related information, called a spoken language constraint network (SLCN). An SLCN represents all possible parses for the represented sentence hypotheses in a compact form, and is operated on by constraints.

One of the most difficult knowledge sources to incorporate into a language understanding system is pragmatics. Pragmatics is the use of language in context. Often pragmatics deals with aspects of communication which go beyond the literal truth conditions of the sentence, as in speech acts. However, here we will only consider how context can help disambiguate the meaning of a sentence and identify precisely which context applies to a particular utterance. We treat context in detail in section 3.8; the underlying idea is that there are multiple databases that may or may not be relevant to a particular utterance. It is the parser's task to both disambiguate the utterance

[1] Prosody can help a word recognizer to rule out word candidates with unlikely stress and duration patterns, but it can also impact syntactic and semantic modules. Therefore, we depict the prosody module as both a high-level and low-level knowledge source in Figure 1.

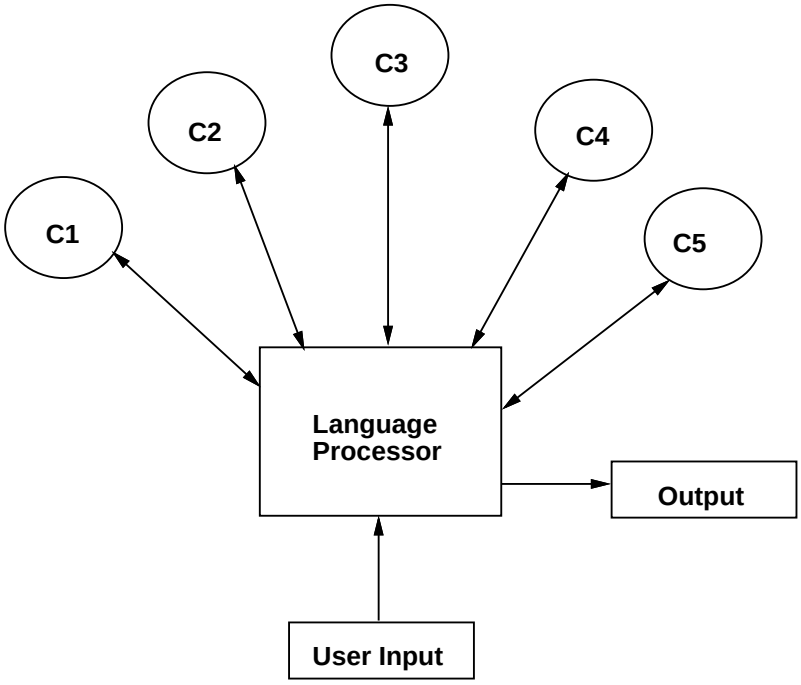


Figure 2: A language interface to multiple databases.

and identify the correct database to consult.

Two recent papers [6, 34] discuss design issues for developing a natural language interface to a database which are relevant to this problem. As an example, consider the query, *Who teaches Smith mathematics*? If a database does not contain any information about math teachers, then the interface should not even attempt to query that database. By keeping track of the language model for a database, a query concerning a topic that is outside of the range of knowledge of the database should not be answered. Taking this notion one step further, we equate the language model for a database with a context and create a single language-based interface to several topical databases, as shown in Figure 2. At its simplest level, this system can be thought of as a general-purpose query answering system for multiple topical databases. The user's input would be processed by the language processor which interfaces to the databases with the goal of interacting with the correct database in order to provide a reasonable answer to the user's spoken request. Initially, it uses general syntactic and semantic rules to analyze a word graph of sentence hypotheses provided by a speech recognizer. If the utterance is still ambiguous, it utilizes context-specific constraints to further refine the analysis.

For example, suppose that the interface is given an utterance like *Display flights to Phoenix*. Even if there is a single sentence recognized by our word recognizer, the system must still determine whether the utterance is syntactically correct and semantically reasonable for at least one database in its purview. The system utilizes all of the knowledge sources it has access to in order to identify the correct database. Knowledge about which words are used for each database together with general (applicable in all contexts) syntactic and semantic constraints could provide a single analysis for the sentence and isolate a single database (or context) the user wishes to target with a request. However, by identifying the correct context, the system should also be able to further refine the parse of the user's input. For example, the utterance *Display flights to Phoenix* may contain legal words for both the real estate and travel databases; however,

the word *flights* would be used differently in each database. By using context-specific constraints, the system should be able to determine that *to Phoenix* modifies *flights* and that the correct context is the travel database application. The synergy among syntax, semantics, and pragmatics can be handled quite effectively in our constraint-based system because it is capable of managing multiple contexts within an evolving picture of a sentence's parse. Each hypothesis is subject to constraints which helps the computer to disambiguate the input syntactically and semantically while determining which context actually applies.

The notion of context can be expanded beyond the database interface to any computer application which would benefit from a natural language interface. Computer applications such as a symbolic math package, a modem program, or a spread sheet could easily benefit from a natural language interface. Given that desk-top computers can manage multiple applications in a multi-tasking environment, this view could provide just the right control structure for the user to move fluidly among the many tasks he or she is trying to accomplish.

We begin our discussion by introducing constraint dependency grammars as defined by Maruyama in section 2. Then in section 3, we describe how that algorithm is extended to process multiple sentence hypotheses in a single constraint network. This same mechanism is utilized to handle not only multiple sentence hypotheses with shared words, but also sentences with multiple parts of speech, feature values, and contexts. We initially describe the mechanism for parsing multiple sentences, and then describe how it can be used as a general mechanism for processing all ambiguity inherent in a sentence, even the ambiguity of selecting the correct computer application with which to interact.

2 The Theoretical Basis of the SLCN Parser

Our system uses Constraint Dependency Grammar (CDG) grammar, originally defined by Maruyama [24, 25, 26], to process sentences. In the following subsections, we will describe the CDG grammar formalism, the CDG parsing algorithm, and the benefits of a constraint-based system.

2.1 Elements of a CDG Grammar

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

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

A sentence $s = w_1 w_2 w_3 \ldots w_n$ is a string of finite length n and is an element of Σ^*. All p roles in R are associated with every w_i of s, yielding $n * p$ roles for the entire sentence. The sentence s is said to be generated by the grammar G if there exists an assignment **A** which maps role values to each of the $n * p$ roles for s such that the constraint set C is satisfied. A role value is an element of the set $L \times \{\text{nil}, 1, 2, \ldots, n\}$. In other words,

it is a tuple consisting of a label from L and a modifiee, where a modifiee can be the index of a word in the sentence or nil. Role values will be denoted in the examples as *label-modifiee*. We use this notation as a shorthand notation for a role value at a particular role. $L(G)$ is the set of all sentences generated by G. Note that the null string ϵ has no roles and is always generated by any grammar by definition.

A *constraint set* is a logical formula of the form[2]:

$$\forall x_1, x_2, \ldots, x_a \in roles \ \text{(and P}_1 \ \text{P}_2 \ \ldots \text{P}_m),$$

where each x_i ranges over all of the role values in *roles*, the set of all the roles of s. Below are definitions for the possible components of a subformula P_i:

- **Variables:** x_1, x_2, ... x_a, $a = 2$ in [24].
- **Constants:** elements and subsets of $\Sigma \cup L \cup R \cup$ {nil, 1, 2, ..., n}, where n corresponds to the number of words in a sentence.
- **Access Functions:**
  (pos x) returns the position of the word for role value x.
  (rid x) returns the role-id for role value x.
  (lab x) returns the label for role value x.
  (mod x) returns the position of the modifiee for role value x.
  (cat i) returns the category (i.e., the element in Σ) for the word[3] 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 $\in$ Integers, false otherwise[4].
  (lt x y) returns true if x < y and x, y $\in$ Integers, false otherwise.
  (elt x y) returns true if x $\in$ y, false otherwise.
- **Logical Connectives:**
  (& p q) returns true if subformulas p and q are true, false otherwise.
  (V p q) returns true if subformula p or q is true, false otherwise.
  (not p) returns true if subformula p is false, false otherwise.

Each subformula 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 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 section, we use a grammar 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, we also use several need 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 [24] has proven that a grammar requires a degree and arity of at least two to be as expressive as a CFG.

To illustrate the use of CDG grammars, consider the simple grammar shown in Figure 3, $G_1 = \langle \Sigma_1, R_1, L_1, C_1 \rangle$, which has a degree of one and an arity of two[5]. A

[2]Maruyama uses an infix notation; whereas, we use a prefix notation throughout this paper.

[3]Maruyama uses the access function `word` rather than `cat`, though the function accesses the category of the word.

[4]For example, (gt 1 nil) is false, because nil is not an integer.

[5]The constraints in this grammar were chosen for simplicity, not to exemplify constraints for a wide coverage grammar.

Σ_1 = {det, noun, verb}
R_1 = {governor}
L_1 = {det, subj, root}
C_1 = $\forall\, x, y \in roles$ (and

```
;; [U-1] A det receives the label det
;; and modifies a word to its right.
(if (eq (cat (pos x)) det)
    (& (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)
    (& (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)
    (& (eq (lab x) root)
       (eq (mod x) nil)))
;; [B-1] A det is governed by a subj.
(if (& (eq (lab x) det)
       (eq (mod x) (pos y)))
    (eq (lab y) subj))
)
```

Figure 3: $G_1 = \langle \Sigma_1, R_1, L_1, C_1 \rangle$.

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

Figure 4: An assignment for *The program runs.*

subformula P_i is called a *unary constraint* if it contains one variable and a *binary constraint* if it contains two. For example, U-1, U-2, and U-3 are unary constraints, and B-1 is a binary constraint because it is a function of the variables x and y.

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. Figure 4 depicts an assignment for the sentence which satisfies C_1. This assignment can be interpreted as the parse graph shown in Figure 14.

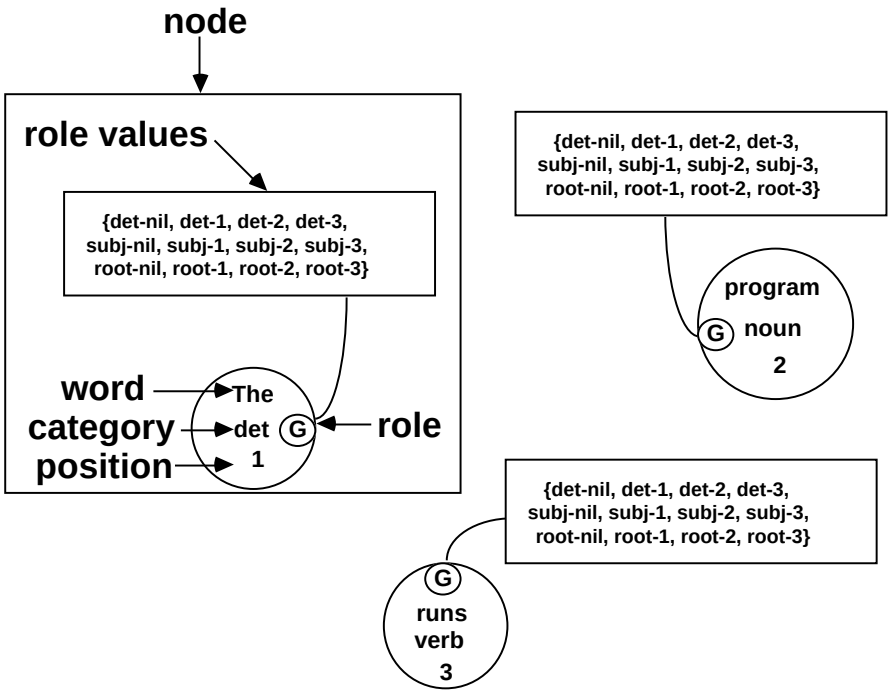


Figure 5: Initialization of roles for the sentence *The program runs*.

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 $L_1 \times \{\text{nil},1,2,\ldots,n\}$, CDG parsing can be viewed as a constraint satisfaction problem over a finite domain. Hence, constraint propagation [23, 28, 38] 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 5 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 $L_1 \times \{\text{nil},1,2,3\}$ = {det-nil, det-1, det-2, det-3, subj-nil, subj-1, subj-2, subj-3, root-nil, root-1, root-2, root-3}. Since there are $q * (n + 1) = \mathrm{O}(n)$ 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 $n * p * q * (n + 1) = \mathrm{O}(n^2)$ role values which must be initially generated for the CN, requiring $\mathrm{O}(n^2)$ time. Note that each role value has access to its role-id and the position of the word in the sentence.

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

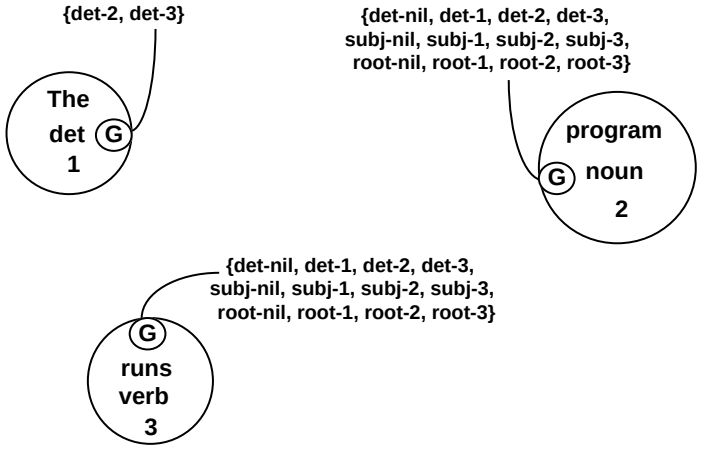


Figure 6: The CN after the propagation of U-1 for the sentence *The program runs.*

the CN 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 5, 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)
    (& (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 6 shows the remaining role values after U-1 has been applied to the CN in Figure 5.

Maruyama uses only 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 6, producing the network in Figure 7. Given that the number of unary constraints in a grammar is a grammatical constant denoted as k_u, the time required to apply all of the unary constraints in a grammar 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 associated *arc matrix*, whose row and column indices are the role

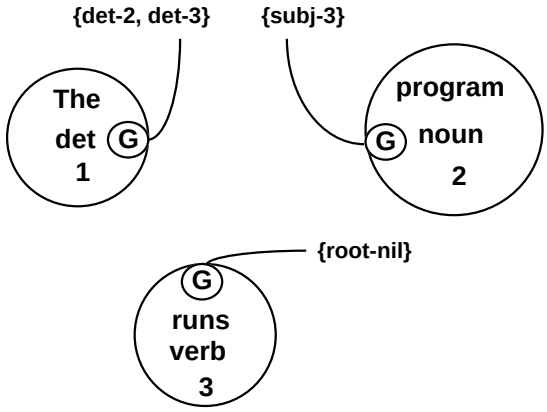


Figure 7: The CN after the propagation of all the unary constraints.

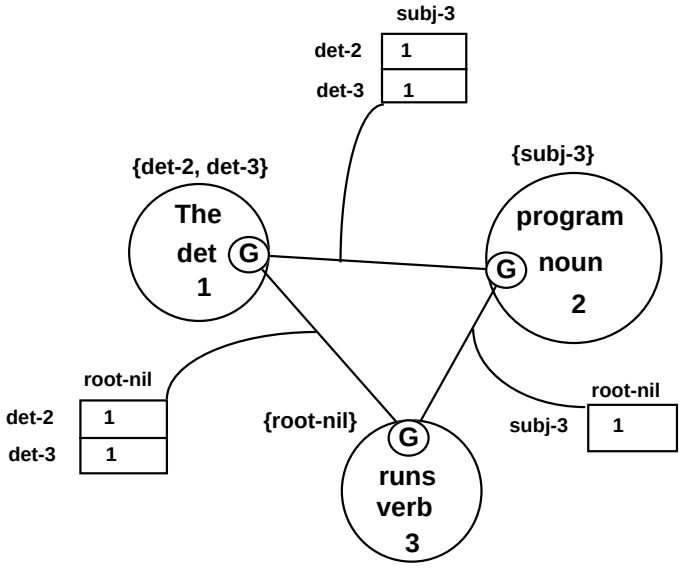


Figure 8: The CN after unary constraint propagation and before binary constraint propagation.

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 $\binom{n*p}{2} = \mathrm{O}(n^2)$ arcs required in the CN, and each arc contains a matrix with $(q*(n+1))^2 = \mathrm{O}(n^2)$ elements, the time to construct the arcs and initialize the matrices is $\mathrm{O}(n^4)$. Figure 8 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 subj.

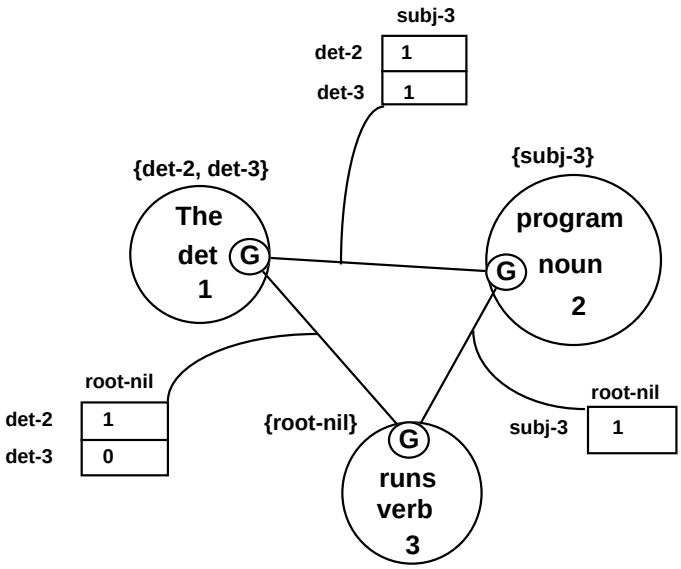


Figure 9: The CN after B-1 is propagated.

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

After the application of this constraint to the network in Figure 8, the element indexed by the role values[6] x=det-3 and y=root-nil for the matrix on the arc connecting the governor roles for *the* and *runs* is set to zero, as shown in Figure 9. This is because *the* must be governed by a word with the label subj, not root. 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 number of binary constraints in a grammar is a grammatical constant denoted as k_b, the time required to apply all of the binary constraints in a grammar 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.

[6] We use det-3 as a shorthand notation for det-3 at a particular role.

Notation	Meaning
(i,j)	An ordered pair of roles.
N	$\{i,j,\ldots\}$ is the set of all roles, with $\|N\| = n * p$.
L	$\{a,b,\ldots\}$ is the set of *role values*, with $\|L\| = q * (n+1)$.
L_i	$\{a \| a \in L$ and (i,a) is permitted by the constraints (i.e., admissible)$\}$
$R2(i,a,j,b)$	$R2(i,a,j,b) = 1$ 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,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.

Figure 10: Data structures and notation for the CN arc consistency algorithm (i.e., AC-4).

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 [27]. AC-4 builds and maintains several data structures, described in Figure 10, to allow it to efficiently perform this operation. Figure 11 shows the code for initializing the data structures, and Figure 12 contains the algorithm for eliminating inconsistent role values from the domains. This filtering algorithm requires $O(ea^2)$, where e is the number of arcs, and a is the size of the domain [27]. 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 $a \in L_i$ 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 must also keep 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 loose 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 listed in Figure 12, additional role values may loose support and be placed on $List$.

```
1. List:=φ;
2. for i ∈ N do
3.     for a ∈ L_i do
4.         begin
5.             M[i,a] := 0;
6.             S[i,a] := φ;
7.         end
8. for (i,j) ∈ E do
9.     for a ∈ L_i do
10.        begin
11.            Total := 0;
12.            for b ∈ L_j do
13.                if R2(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 ∪ {(i,a)};
22.                  L_i := L_i − {a};
23.              end
24.            Counter[(i,j),a] :=Total;
25.        end
```

Figure 11: Construction of data structures for CN arc consistency (i.e., AC-4).

```
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.                      L_i := L_i − {a}
12.                  end
13.            end
14.    end
```

Figure 12: Algorithm to enforce CN arc consistency (i.e., AC-4).

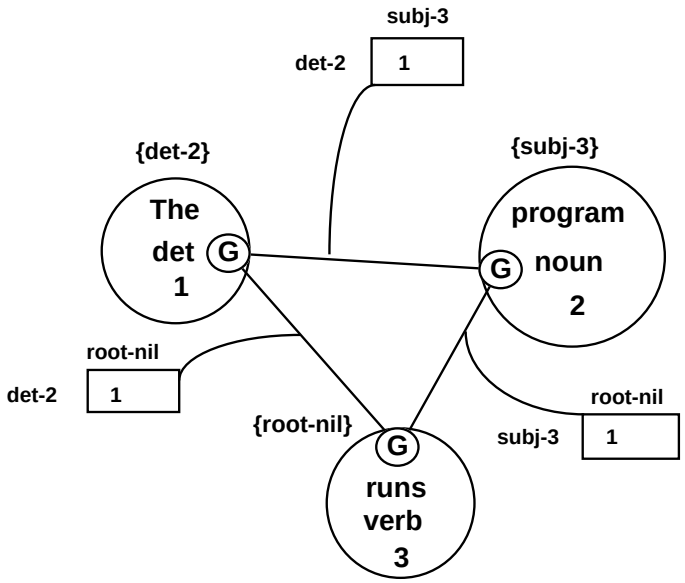


Figure 13: The CN after filtering.

Consider how filtering is applied to the CN in Figure 9. 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*[7], 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 13.

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 modifiees 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 14.

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): $O(n^4)$

[7]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.

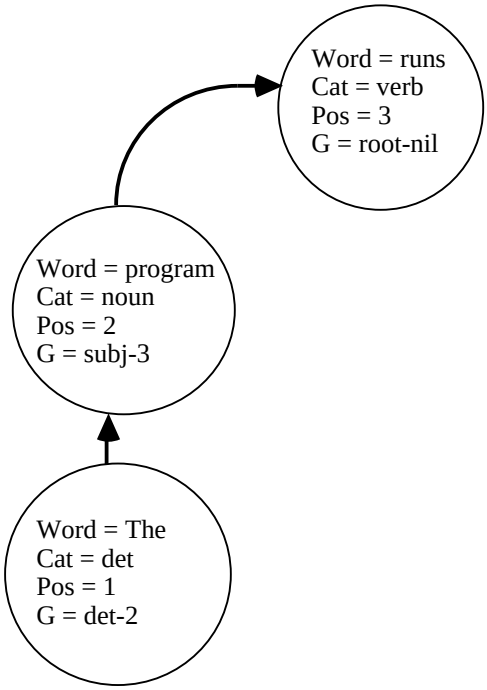


Figure 14: The parse graph for the CN in Figure 13.

2.3 Benefits of a Constraint-Based Approach

There are many benefits to using a constraint based parser, with the primary one being flexibility. When a traditional context-free grammar (CFG) parser generates a set of ambiguous parses for a sentence, it cannot invoke additional production rules to further prune the analyses. In CDG parsing, on the other hand, the presence of ambiguity can trigger the propagation of additional constraints to further refine the parse. A core set of constraints that hold universally can be propagated first, and then if ambiguity remains, additional, possibly context-dependent, constraints can be applied. We have developed semantic constraints which are used to eliminate parses with semantically anomalous readings from the set represented in the constraint network [14]. Additional knowledge sources are quite easy to add given the uniform framework provided by constraints, as we demonstrate in this paper.

Tight coupling of prosodic [3] and semantic rules with CFG grammar rules typically increases the size and complexity of the grammar and reduces its understandability. Semantic grammars have been effective for limited domains, but they often do not scale up well to larger systems [1]. The most successful modules for semantics are more loosely coupled with the syntactic module. The constraint-based approach represents a loosely-coupled approach for combining a variety of knowledge sources. It differs from a blackboard approach in that all constraints are applied using the uniform mechanism of constraint propagation [13]. Hence, the designer does not need to create a set of functionally different modules and worry about their interface with other modules. Constraint propagation is a uniform method which allows the designer to focus on the best way to order the sources of information impacting comprehension.

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 up to two variables in a constraint

(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 $O(G)$, where G is the size of the CFG. In addition, CDG can accept languages that CFGs cannot (e.g., $a^n b^n c^n$ and ww, where w is some string of terminal symbols). There has 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 [20, 36, 37]. The running time of the CDG parser compares quite favorably to the running times of parsers for languages which are beyond context-free. For example, the parser for tree adjoining grammars (TAG) has a running time of $O(n^6)$.

CFG parsing has been parallelized by several researchers. For example, Kosaraju's method [22] 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 than used by [22], as well as significantly more processors. Ruzzo's method [33] 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. In contrast, we have devised a parallelization for the single sentence CDG parser [15, 16] which uses $O(n^4)$ processors to parse in $O(k)$ time for a CRCW P-RAM 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, a massively parallel SIMD computer. The MP-1 supports up to 16K 4-bit processing elements, each with 16KB of local memory. The CDG algorithm on the MP-1 achieves an $O(k + log(n))$ running time by using $O(n^4)$ processors. By comparison, the TAG parsing algorithm has also been parallelized, and operates in linear time with $O(n^5)$ processors [30].

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 [29]. Unordered CFGs do not have this combinatorial explosion of rules, but the recognition problem for this class of grammars is NP-complete. A free-order language can easily be handled by a CDG parser because order between constituents is not a requirement of the grammatical formalism. Furthermore, CDG is capable of efficiently analyzing free-order languages because it is does not have to test for all possible word orders.

In summary, CDG supports a framework which 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 parallelizeable.

3 Parsing Spoken Language with Constraints

The output of a hidden-Markov-model-based speech recognizer is often a list of the most likely sentence hypotheses (i.e., an N-best list) where parsing can be used to rule out the impossible sentence hypotheses. CDG constraints can be used to parse single

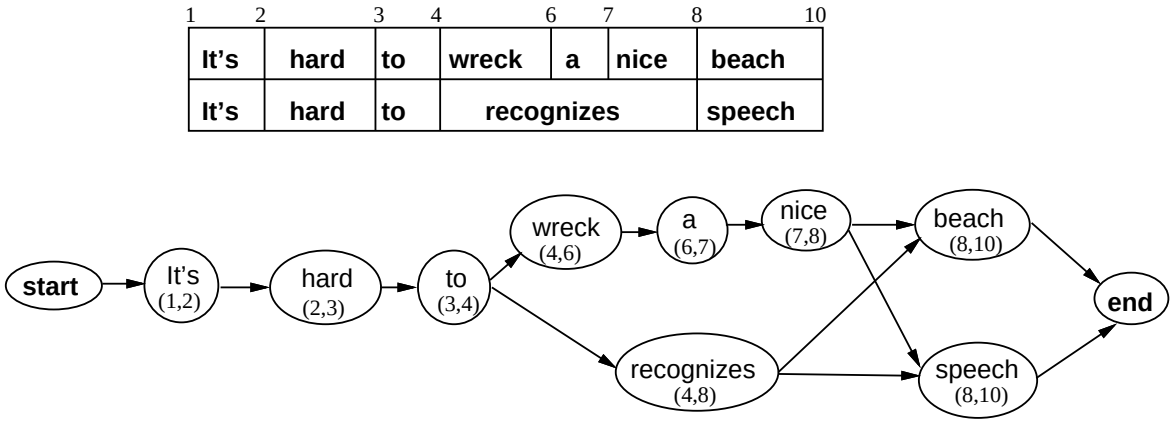


Figure 15: Multiple sentence hypotheses can be represented in a single word graph.

sentences in a CN; however, individually processing each sentence hypothesis provided by a speech recognizer is inefficient since many sentence hypotheses are generated with a high degree of similarity. An alternative representation for a list of similar sentence hypotheses is a word graph which contains information on the approximate beginning and end points of each word. A word graph represents a disjunction of all possible sentence candidates that the speech recognizer provides.

Word graphs are typically more compact and more expressive than N-best sentence lists. In [43], word graphs were constructed from three different lists of sentence hypotheses. The word graphs provided an 83% reduction in storage, and in all cases, they encoded more possible sentence hypotheses than were in the original list of hypotheses. In one case, 20 sentence hypotheses were converted into a word graph representing 432 sentence hypotheses. Figure 15 depicts a word graph containing four sentence hypotheses which was constructed from two sentence hypotheses: **It's hard to recognizes speech* and *It's hard to wreck a nice beach.* We view the problem of parsing spoken language as a graph processing problem. The constraints used to parse individual sentences can then be applied to eliminate impossible hypotheses.

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). The input to the parser is a word graph like the one shown in Figure 15. Each word node in the word graph contains information on word beginning and end points, represented as an integer tuple (b, e), with b < e. The tuple is more expressive than the single integer used for CNs and requires modification of some of the access functions and predicates. Notice that nodes that can be adjacent to one another are joined by directed *edges.* 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 number of sentence hypotheses represented by a graph of n nodes can be exponential in n. The goal of our system is to utilize constraints to eliminate as many impossible sentence hypotheses as possible, and then to select the best remaining sentence hypothesis (given the word probabilities given by the recognizer).

Some of the constraint access functions and predicates must be adapted for SLCN parsing. For example, the access functions (pos x) and (mod x) now return a tuple (b, e) which describes the position of the word associated with the role value x. The

equality predicate is extended to test for equality of intervals (e.g., (eq (1,2) (1,2)) should return true). Also, the less-than predicate, (lt (b1, e1) (b2, e2)), returns true if e1 < b2, and the greater-than predicate, (gt (b1, e1) (b2, e2)), returns true if b1 > e2.

One additional change is needed to accommodate multiple words over the same time interval. Recall that in CN parsing, a word node has a unique category and position. For example, to access the category associated with a role value, Maruyama would use the function (cat (pos i)), where (pos i) returns the position of the role value. For an SLCN, it is not always possible to determine the category for a role value by using the position of the word in the sentence because some word nodes share the same position. We handle this by allowing the role values to keep track of their part of speech, not just the position of their word node. Hence, the constraints in Figure 3 must be rewritten so that the access function `cat` operates on a role value rather than on a word node addressed by its position. For example, U-1 is rewritten as follows:

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

To apply constraints to the word graph, each word node must be annotated with a set of roles. Then each role for each word node is assigned a set of role values, 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, this requires $O(k_u * n^2)$ 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)[8]. For example, there should be an arc between the roles for *recognizes* and *speech* in Figure 16 because they are located on a path from the beginning to the end of the sentence **It's hard to recognizes speech.* However, there should not be an arc between the roles for *wreck* and *recognizes* 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 $O(n^2)$ 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 elements (in the same manner as for the CN), requiring $O(k_b * n^4)$ time, where n is the number of word candidates.

Filtering an SLCN is complicated because the limitation of one word's function in

[8]To construct the arcs 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 another node if and only if there is a directed edge from the first node to the second node) and to each of the roles that the preceding word nodes' roles have arcs to.

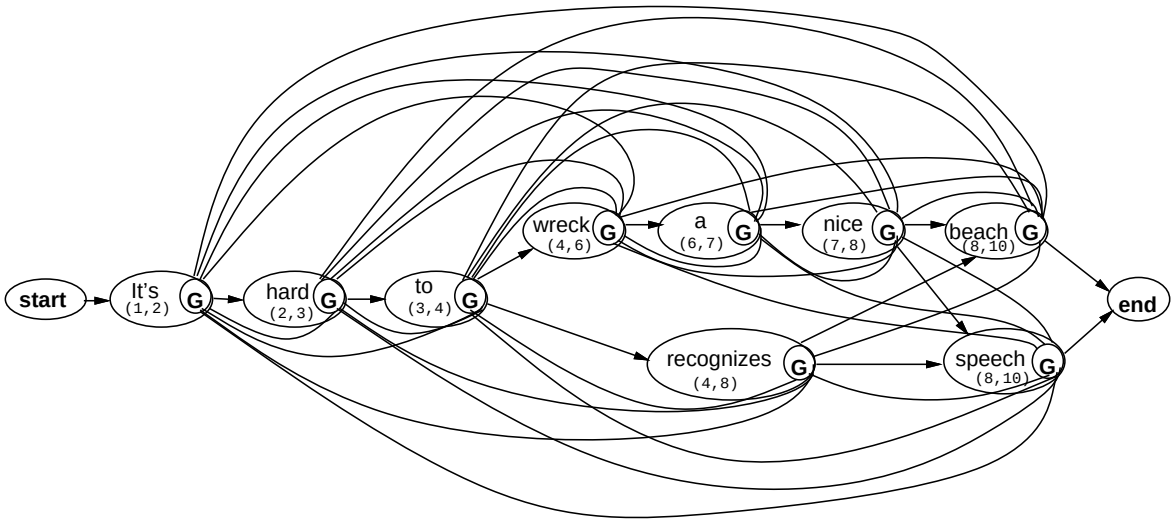


Figure 16: Multiple sentence hypotheses can be parsed simultaneously by propagating constraints over an SLCN rather than individual CNs.

one sentence hypothesis should not necessarily limit that word's function in another sentence hypothesis. For example, consider the SLCN depicted in Figure 16. Even though all the role values for *to* would be disallowed by the third person singular verb *recognizes*, those role values cannot be eliminated since they are supported by *wreck*, an infinitive verb. 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 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 [11, 17]. This algorithm, described in the next section, can process single sentences or word graphs.

3.1 SLCN Arc Consistency

When we create a constraint network representing multiple alternative sentence hypotheses, we have changed the logical meaning of the constraint network significantly. 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 17. 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 18 depicts a simple SLCN with two roles constructed from a word graph. Figure 19 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 *arc consistent* if and only if for every role

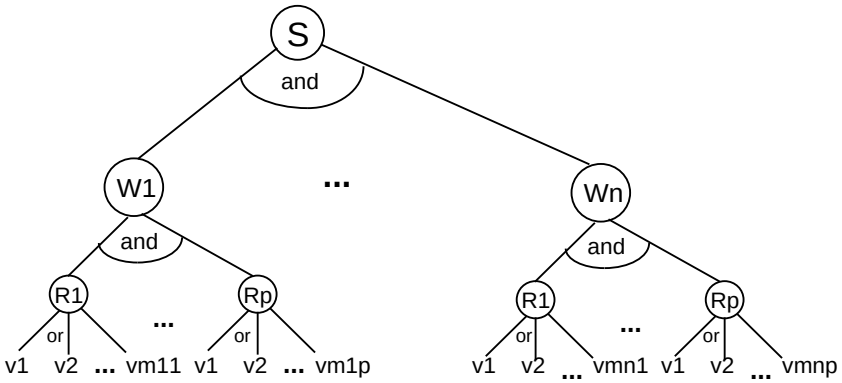


Figure 17: The AND/OR graph for a CN.

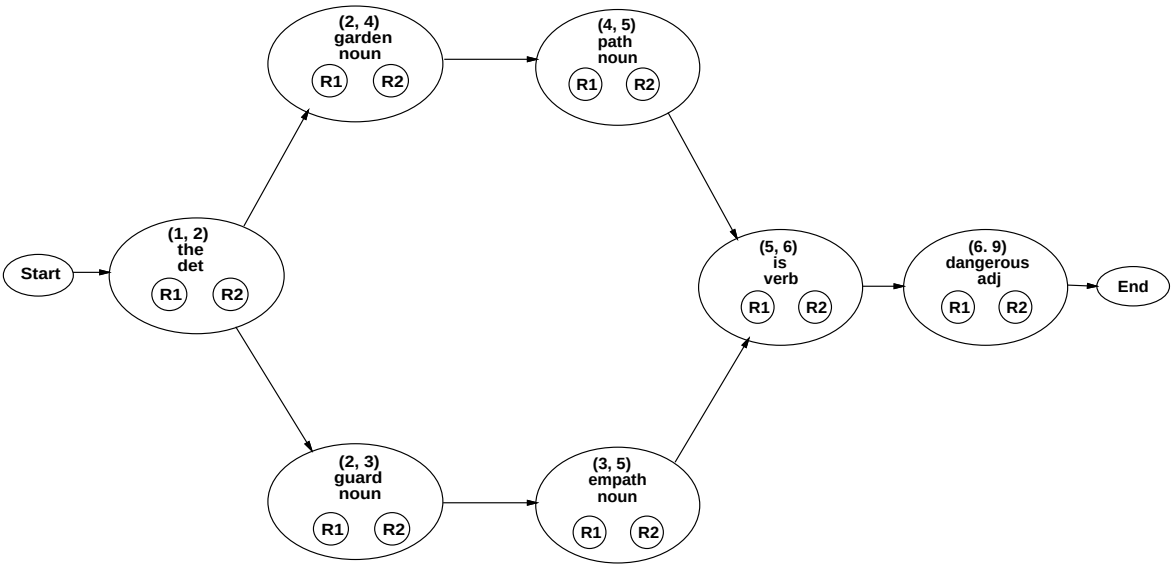


Figure 18: A simple SLCN.

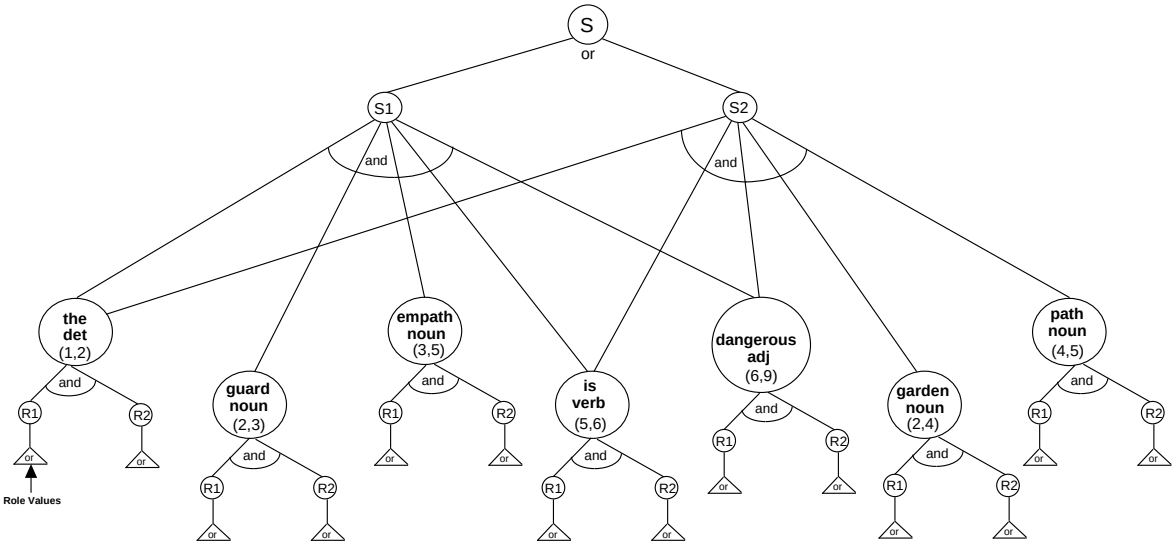


Figure 19: The AND/OR graph for the SLCN in Figure 18.

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, 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 Figure 20. Figure 23 shows the code for initializing the data structures, and Figure 24 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 can be 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 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 = (role-of (mod b)) $\wedge$ $(i,k) \in E$),
2. (mod b) = nil,
3. (mod b) = (pos a).

These three conditions are used to ensure that b is a role value that can occur in at least one sentence hypothesis with a[9]. 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 Counter$[(j,i),b]$ and Counter$[(j,i),c]$. If 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 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 solution 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 21, which shows the roles that are adjacent to role i in an SLCN. Because every sentence in the SLCN which

[9] The only condition for a supporting b in the general MUSE CSP algorithm [17, 18] is $R2(i,a,j,b)$ because not all values assigned to variables in CSP have modifiees.

Notation	Meaning
(i, j)	An ordered pair of roles.
N	$\{i, j, \ldots\}$ is the set of all roles, with $\|N\| = n * p$.
L	$\{a, b, \ldots\}$ is the set of *role values*, with $\|L\| = q * (n + 1)$.
L_i	$\{a \| a \in L$ and (i, a) is admissible$\}$
$R2(i, a, j, b)$	$R2(i, a, j, b) = 1$ 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 value $a \in L_i$ and $b \in L_j$ are simultaneously admissible. This implies that a supports b.
Next-edge$_i$	If a directed edge from i to j exists in E, then $(i, j) \in$ Next-edge$_i$.
Prev-edge$_i$	If a directed edge from j to i exists in E, then $(j, i) \in$ Prev-edge$_i$.
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$_i$ 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$_i$ and a is compatible with at least one of j's role values.
List	A queue of arc support to be deleted.

Figure 20: Data structures and notation for the SLCN arc consistency algorithm.

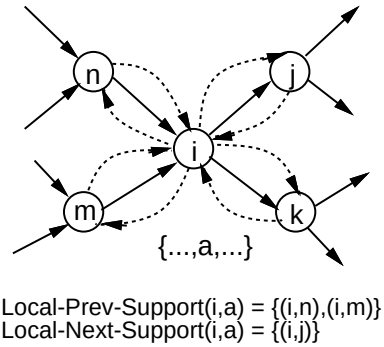


Figure 21: Local-Prev-Support and Local-Next-Support for an example SLCN.

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 $a \in L_i$, Local-Next-Support(i,a) and Local-Prev-Support(i,a). Local-Next-Support(i,a) is a set of ordered role pairs (i,j) such that $(i,j) \in$ Next-edge$_i$, and there is at least one role value $b \in L_j$ which is compatible with a. Local-Prev-Support(i,a) is a set of ordered pairs (i,j) such that $(j,i) \in$ 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 Local-Prev-Support(i,a) or Local-Next-Support(i,a), depending on whether the edge is from j to i or from i to j, respectively. If either Local-Prev-Support(i,a) or Local-Next-Support(i,a) becomes the empty set, then a is no longer a part of any solution, and may be eliminated from L_i. In Figure 21, 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, 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 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 Local-Next-Support(a,i), leaving an empty set. In that case, a would no longer be supported by any sentence.

The algorithm can utilize similar conditions for roles which may not be directly connected to i by Next-edge$_i$ or Prev-edge$_i$. Consider Figure 22. 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 Prev-Support$[(i,j),a]$ and Next-Support$[(i,j),a]$. 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. 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

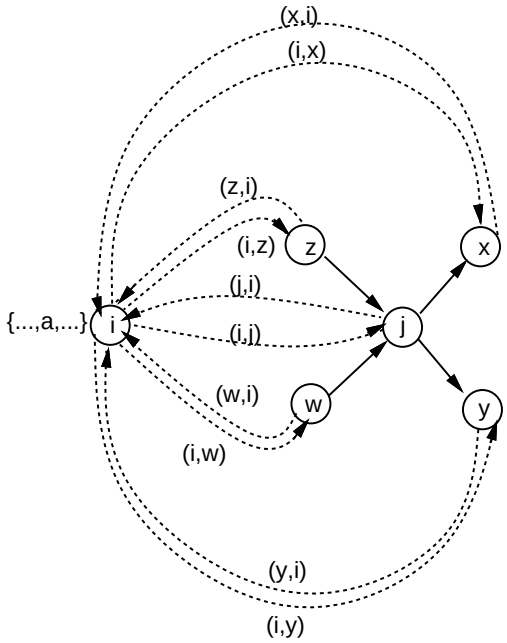
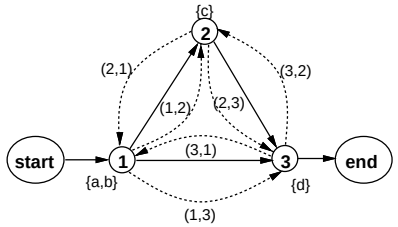


Figure 22: If Next-edge$_j = \{(j,x),(j,y)\}$, S$[(i,x),a] = \phi$, and S$[(i,y),a] = \phi$, then a is inadmissible in every sentence containing both i and j.

Prev-Support$[(i,j),a]$ will contain an ordered pair (i,j) if $(i,j) \in$ Prev-edge$_j$, and Next-Support$[(i,j),a]$ will contain an ordered pair (i,j) if $(j,i) \in$ 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 $(\mathbf{start}, j) \in$ Prev-edge$_j$, $(i, \mathbf{start})$ is added to Prev-support$[(i,j),a]$, and when $(j, \mathbf{end}) \in$ Next-edge$_j$, $(i, \mathbf{end})$ is added to 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. Figure 23 shows the Prev-Support, Next-Support, Local-Next-Support, and 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 24), consider what happens if initially $[(1,3),a] \in \mathit{List}$ for the SLCN in Figure 23. $[(1,3),a]$ 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 2. When that value is popped off List, it is necessary to remove $[(1,3),a]$'s support from all S$[(3,1),x]$ such that $[(3,1),x] \in$ S$[(1,3),a]$ by decrementing for each x, 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 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 Prev-Support$[(1,3),a] = \{(1,2),(1,3)\}$ and Next-Support$[(1,3),a] = \{(1,\mathbf{end})\}$. Note that the value $(1,\mathbf{end})$ in Next-Support$[(1,3),a]$ and the value $(1,3)$ in 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 Prev-Support$[(1,3),a]$ indicates that $(1,3)$ is a member of Next-Support$[(1,2),a]$, and since a is not admissible for $(1,3)$, $(1,3)$ should be removed from Next-Support$[(1,2),a]$, leaving an empty set. Note that because 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 Local-Next-

```
1. List:=φ;
2. E := {(i,j)|∃σ ∈ Σ (i,j ∈ σ ∧ i ≠ j ∧ i,j ∈ N)};
3. for (i,j) ∈ E do
4.     for a ∈ L_i do
5.         begin
6.             M[(i,j),a] := 0;
7.             Prev-Support[(i,j),a] := φ; Next-Support[(i,j),a] := φ;
8.             Local-Prev-Support(i,a) := φ; Local-Next-Support(i,a) := φ;
9.             S[(i,j),a] := φ;
10.        end
11. for (i,j) ∈ E do
12.     for a ∈ L_i do
13.         begin
14.             Total:=0;
15.             for b ∈ L_j do
16.                 if R2(i,a,j,b) ∧
                       [∃k ∈ N (k = (role-of (mod b)) ∧ (i,k) ∈ E) ∨
                       (mod b) = nil ∨ (mod b) = (pos a)] then
17.                 begin
18.                     Total:=Total+1;
19.                     S[(j,i),b] := S[(j,i),b] ∪ {[(i,j),a]};
20.                 end
21.             if Total=0 then
22.                 begin
23.                     M[(i,j),a] := 1;
24.                     List:=List ∪ {[(i,j),a]};
25.                 end
26.             Counter[(i,j),a]:=Total;
27.             Prev-Support[(i,j),a] := {(i,x)|(i,x) ∈ E ∧ (x,j) ∈ Prev-edge_j}
                                         ∪ {(i,j)|(i,j) ∈ Prev-edge_j}
                                         ∪ {(i,start)|(start,j) ∈ Prev-edge_j};
28.             Next-Support[(i,j),a] := {(i,x)|(i,x) ∈ E ∧ (j,x) ∈ Next-edge_j}
                                         ∪ {(i,j)|(j,i) ∈ Next-edge_j}
                                         ∪ {(i,end)|(j,end) ∈ Next-edge_j};
29.             if (i,j) ∈ Next-edge_i then
30.                 Local-Next-Support(i,a):=Local-Next-Support(i,a) ∪{(i,j)};
31.             if (j,i) ∈ Prev-edge_i then
32.                 Local-Prev-Support(i,a):=Local-Prev-Support(i,a) ∪ {(i,j)};
33.         end
```



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

Figure 23: 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 ≠ φ do
2.     begin
3.         choose [(j,i),b] from List and remove it from List;
4.         for [(i,j),a] ∈ S[(j,i),b] do
5.             begin
6.                 Counter[(i,j),a]:=Counter[(i,j),a] − 1;
7.                 if Counter[(i,j),a] = 0 ∧ M[(i,j),a] = 0 then
8.                   begin
9.                       List:=List ∪{[(i,j),a]};
10.                      M[(i,j),a] := 1;
11.                  end
12.            end
13.        for (j,x) ∈ Next-Support[(j,i),b] do
14.            begin
15.                Prev-Support[(j,x),b]:=Prev-Support[(j,x),b] − {(j,i)};
16.                if Prev-Support[(j,x),b] = φ ∧ M[(j,x),b] = 0 then
17.                  begin
18.                      List:=List ∪{[(j,x),b]};
19.                      M[(j,x),b] := 1;
20.                  end
21.            end
22.        for (j,x) ∈ Prev-Support[(j,i),b] do
23.            begin
24.                Next-Support[(j,x),b]:=Next-Support[(j,x),b] − {(j,i)};
25.                if Next-Support[(j,x),b] = φ ∧ M[(j,x),b] = 0 then
26.                  begin
27.                      List:=List ∪{[(j,x),b]};
28.                      M[(j,x),b] := 1;
29.                  end
30.            end
31.        if (j,i) ∈ Next-edge_j then
32.          Local-Next-Support(j,b):=Local-Next-Support(j,b) − {(j,i)};
33.        if Local-Next-Support(j,b) = φ then
34.          begin
35.              L_j := L_j − {b};
36.              for (j,x) ∈ Local-Prev-Support(j,b) do
37.                  if M[(j,x),b] = 0 then
38.                    begin
39.                        List:=List ∪{[(j,x),b]};
40.                        M[(j,x),b] := 1;
41.                    end
42.          end
43.        if (i,j) ∈ Prev-edge_j then
44.          Local-Prev-Support(j,b):=Local-Prev-Support(j,b) − {(j,i)};
45.        if Local-Prev-Support(j,b) = φ then
46.          begin
47.              L_j := L_j − {b};
48.              for (j,x) ∈ Local-Next-Support(j,b) do
49.                  if M[(j,x),b] = 0 then
50.                    begin
51.                        List:=List ∪{[(j,x),b]};
52.                        M[(j,x),b] := 1;
53.                    end
54.          end
55.    end
```

Figure 24: Algorithm to enforce SLCN arc consistency.

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 Prev-Support$[(1,2),a]$ and Next-Support$[(1,2),a]$ are processed, Next-Support$[(1,2),a] = \phi$ and Prev-Support$[(1,2),a]$ contains only a dummy, which is removed. When $(1,2)$ is removed from 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 $(1,x) \in$ Local-Prev-Support$(1,a)$. Since Local-Prev-Support$(1,a) = \{(1, \textbf{start})\}$, and **start** is a dummy role, there is no more work to be done.

In contrast, consider what happens if initially $[(1,2),a] \in List$ for the SLCN in Figure 23. In this case, Prev-Support$[(1,2),a]$ contains $(1,2)$ which requires no additional work; whereas, Next-Support$[(1,2),a]$ contains $(1,3)$, indicating that $(1,2)$ must be removed from Prev-Support$[(1,3),a]$'s set. After the removal, 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.

3.2 The Running Time and Correctness of SLCN Arc Consistency

The running time of the routine to initialize the SLCN arc consistency structures (in Figure 23) is $O(n^4)$, and the running time for the algorithm which prunes labels that are not arc consistent (in Figure 24) also operates in $O(n^4)$ time, where n is the number of word nodes in network. By comparison, the running time for CN arc consistency is $O(n^4)$, assuming that there are n words in a sentence. The proof of correctness of this algorithm is detailed in [17, 18], but we will summarize it below.

A role value is eliminated from a domain by SLCN arc consistency only if its Local-Prev-Support or its Local-Next-Support set becomes empty. Therefore, we must show that a role value's local support sets become empty if and only if that role value cannot participate in an SLCN arc consistent solution. 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 i in the SLCN, then it cannot participate in an SLCN arc consistent solution. On line 32 of Figure 24, (i,j) is removed from Local-Next-Support(i,a) set only if $[(i,j),a]$ has been popped off $List$. Therefore, we show that $[(i,j),a]$ is put on $List$, only if $a \in L_i$ is incompatible with every sentence which contains i and j, by induction on the number of iterations of the while loop.

For the 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 24 of Figure 23). Therefore, $a \in L_i$ is in no solution for any sentences which contain i and j. Assume the condition holds for the first k iterations of the while loop in Figure 24, then during the (k+1)th iteration, tuples of the form $[(i,j),a]$ for the (k+1)th iteration were put on $List$ by one of the following:

- line 24 in Figure 23 (in which case, a is no longer compatible with any values in L_j),

- line 9 in Figure 24 (in which case, a is no longer compatible with any values in L_j),
- line 18 in Figure 24 (in which case, Prev-Support($[(i,j),a]$) = ϕ),
- line 27 in Figure 24 (in which case, Next-Support($[(i,j),a]$) = ϕ),
- line 39 in Figure 24 (in which case, Local-Next-Support$[i,a] = \phi$).

In any of these cases, $a \in L_i$ is incompatible with every sentence which contains i and j. We can therefore conclude that this is true for all iterations of the while loop.

3.3 Multiple Roles in an SLCN

As shown in Figure 18, an SLCN can have more than a single role. In our initial development of the SLCN arc consistency algorithm discussed in [11], 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 23 and 24, but the intra-arcs were handled differently. They seemed to 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 simpler algorithm shown in Figures 23 and 24. It simply involves setting up the SLCN in a slightly different way than in Figure 18. The edges in the SLCN in Figure 18 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 25, then the arc consistency algorithm in Figure 24 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 25 and use the simpler arc consistency algorithm described in this paper.

3.4 Lexical Ambiguity

Many words in the English language have more than a single part of speech. For example, the word *garden* in Figure 25 can either be 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, the 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 [7, 11].

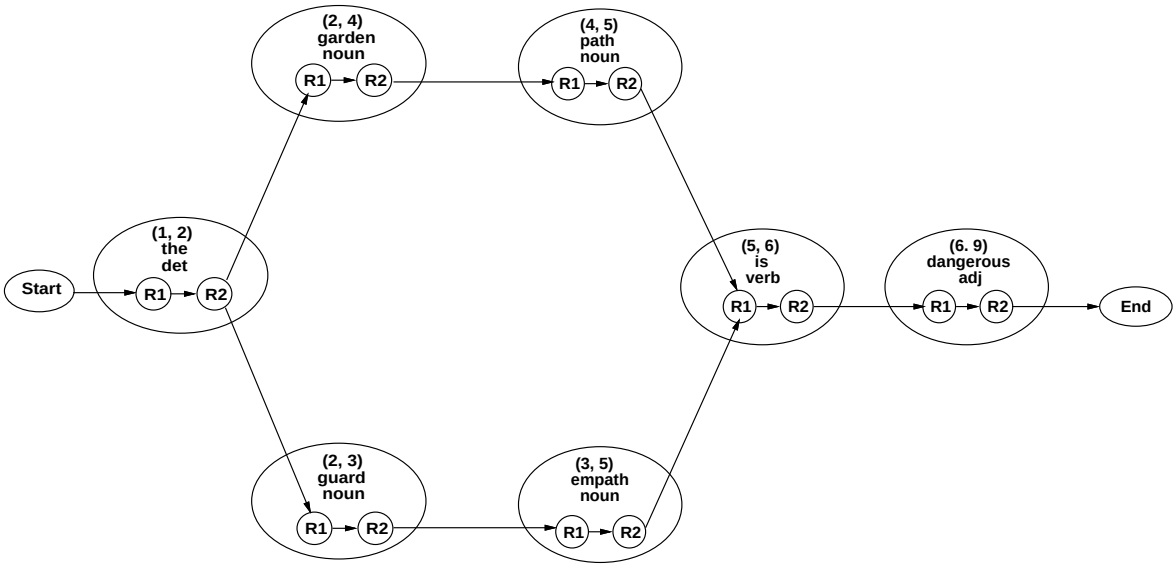


Figure 25: An SLCN with multiple roles compatible with our new algorithm.

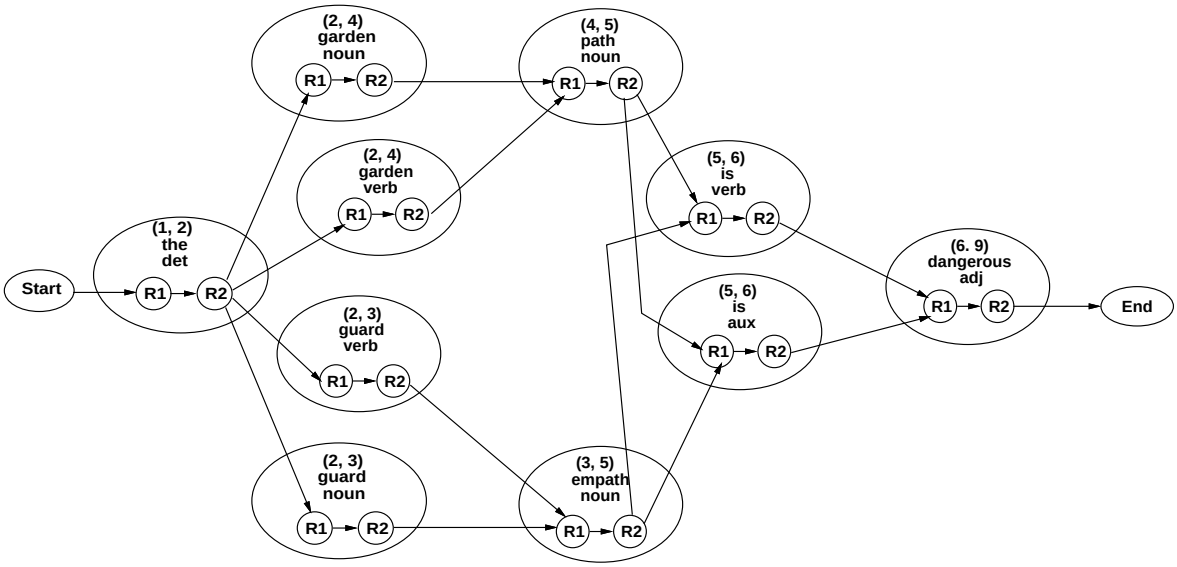


Figure 26: An SLCN with multiple parts of speech for some words.

To handle lexically ambiguous words, we create a word node for each legal part of speech for a word. These word nodes cannot appear in the same sentence hypotheses, and so are not connected to each other by directed edges and do not share arcs for binary constraints. For example, Figure 26 depicts an SLCN which supports multiple parts of speech for several different word nodes. This method of handling multiple parts of speech within our constraint parsing algorithm requires the use of the SLCN arc consistency algorithm described in this paper.

Multiple parts of speech can be handled in a CN algorithm [11] by creating separate role values for each part of speech within the same word node, where it becomes the responsibility of the role value to keep track of its lexical category. However, this approach uses more space and requires the propagation of an additional constraint when compared with the method described in this paper. In a CN, role values associated with two different roles within a word node should not be allowed to support each other if

they do not correspond to the same part of speech. Hence, we must propagate a binary constraint which zeros out the elements of all intra-arc arc matrices that are indexed by role values corresponding to different parts of speech. By using an SLCN to handle lexical ambiguity, the roles of the word nodes corresponding to different parts of speech cannot appear in any common sentences and so are not connected by arcs, eliminating the wasted space and the need for an additional constraint.

3.5 Feature Analysis

Lexical features, like number, person, and case, 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 or for eliminating impossible sentence hypotheses; hence, our parser supports them.

Many times, even if a word is not lexically ambiguous, it can have ambiguity in the feature information associated with the word. For example, the noun *fish* can take the number/person feature value of third person singular or third person plural. For each feature type, it is important to associate a single feature value with each role value being tested for that feature. Because our parser utilizes only unary and binary constraints, role values with feature value ambiguity can only be tested pairwise for consistency, and yet the feature values associated with one word in a valid parse must often be compatible with more than one word in the sentence. For example, if we store sets of features with a node when we parse the sentence **A fish eat*, it is easy to ensure that *a* and *fish* agree in number and person by using a binary constraint, and that *fish* and *eat* agree, but without using ternary constraints, there is no way to ensure that *a*, *fish*, and *eat* have jointly compatible feature values. Furthermore, there is no guarantee that ternary constraints (or for that matter n-ary constraints, for any prespecified n) are sufficient to ensure that the parser rejects sentences that are ungrammatical because of incompatible feature values (e.g., **The fish which are eating swims*).

In order to enforce feature value compatibility across a set of words, word nodes should be duplicated (along with their role values and arc matrices) and assigned a single feature value for the feature being tested by a constraint, not a set of features. A naive and computationally expensive way to achieve this end is to initially duplicate each node so that all of the possible combinations of feature values are covered. Fortunately, there is a better alternative; when the parser is propagating a constraint with a particular feature test, a node with multiple values for that feature can be duplicated and assigned one of the feature values, and then that constraint containing the feature test can be applied to its role values, eliminating many of them before other types of feature constraints are propagated. A grammar writer can order constraints in a constraint file in such a way that role value duplication is minimized. For example, by placing pure phrase structure constraints before constraints containing feature tests, syntactically eliminated role values will not have to be duplicated and tested against the feature constraints. Given this strategy, the running time of the parser with feature constraints is comparable to the running time of the parser without feature tests.

In the next section, we illustrate how our system parses a word graph provided by a speech recognizer. This example shows how node splitting is used to handle feature value ambiguity during the propagation of a number feature constraint.

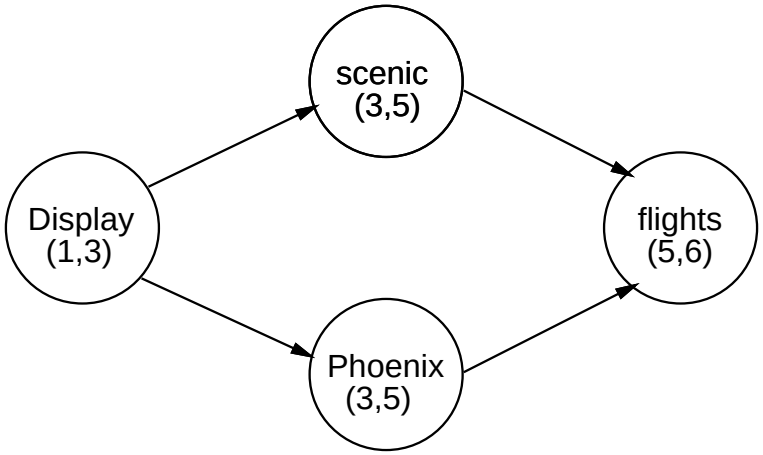


Figure 27: A word graph provided to our system by the speech recognizer.

3.6 A Parsing Example

Our parser requires a grammar designer to specify the grammar parameters, write and test constraints consistent with the grammar parameters, and design a grammar-appropriate lexicon. Assume that a grammar is needed to parse the word graph shown in Figure 27.

The grammar designer provides a file containing the grammar parameters to the parser. This file provides information needed to check constraints for good form and to create the constraint network. Figure 28 depicts an example of a grammar parameter file for a simple grammar to parse our example. Notice that in addition to categories, roles, and labels, there is a label_table which restricts labels by part of speech and role-id, a set of grammar features restricting the feature values for each feature type, and a feature_table restricting the the legal feature types for each part of speech.

Our parser uses the feature_table to check constraints and dictionary entries for good form. During SLCN construction, a role is assigned only those role values with labels indexed by the role's category and role-id in the label_table. In practice, this table reduces the number of role values in the initial SLCN by a factor of five to seven, and eliminates the need to propagate some unary constraints. Even though this does not improve the asymptotic running time of the algorithm, it does decrease the actual running time of the CDG algorithm (and the size of the SLCN).

The dictionary for our simple example is described next. The lexicon is represented as a list of word entries, where each word entry is a list headed by the associated word along with other important information. Each word entry includes information on its legal parts of speech. In addition, syntactic and semantic features are stored for each part of speech of a word. The lexicon for our example appears in Figure 29. This lexicon stores information about number, semantic type, and subcategorization. The word *flights* inherits features that are not mentioned in its entry from its root form *flight*.

Given the grammar parameters, our system constructs the SLCN in Figure 30 from the word graph in Figure 27 by looking up information on each word in the dictionary. Notice that there are two roles for each word node, governor (i.e., **G**) and need (i.e., **N**), and that two nodes are created for the word *display* because it has two possible

```
; A list of the legal parts of speech

(categories adj noun verb propernoun)

; A list of role names for the grammar

(roles governor need)

; A list of legal labels for the grammar

(labels root obj adj noun_mod s blank)

; A label table for restricting the domains given
; part of speech and role name

(label_table (governor (noun noun_mod obj)
                       (propernoun noun_mod obj)
                       (verb root)
                       (adj adj))
             (need (noun blank)
                   (propernoun blank)
                   (verb s)
                   (adj blank)))

; A list of legal feature types and possible values

(grammar_features (number 1s 2s 3s 1p 2p 3p)
                  (subcat dobj)
                  (sem_type sign v_qual city air_transfer show))

; A feature table indicating the possible values for
; each feature type associated with a part of speech
; and its default value if none is specified

(feature_table (adj (sem_type v_qual [ ]))
               (noun (number 3s 3p [3s])
                     (sem_type sign air_transfer [ ]))
               (propernoun (number 3s 3p [3s])
                           (sem_type city [ ]))
               (verb (number 1s 2s 3s 1p 2p 3p [3s])
                     (subcat dobj)
                     (sem_type show [ ])))
```

Figure 28: A grammar parameter file for a simple example grammar.

```
(display (category noun (number 3s)
                        (sem_type sign))
         (category verb (number 1s 2s 1p 2p 3p)
                        (subcat dobj)
                        (sem_type show)))
(flight (category noun (number 3s)
                       (sem_type air_transfer)))
(flights (root_word flight)
         (category noun (number 3p))))
(Phoenix (category propernoun (number 3s)
                              (sem_type city)))
(scenic (category adj (sem_type v_qual)))
```

Figure 29: A dictionary for parsing our example.

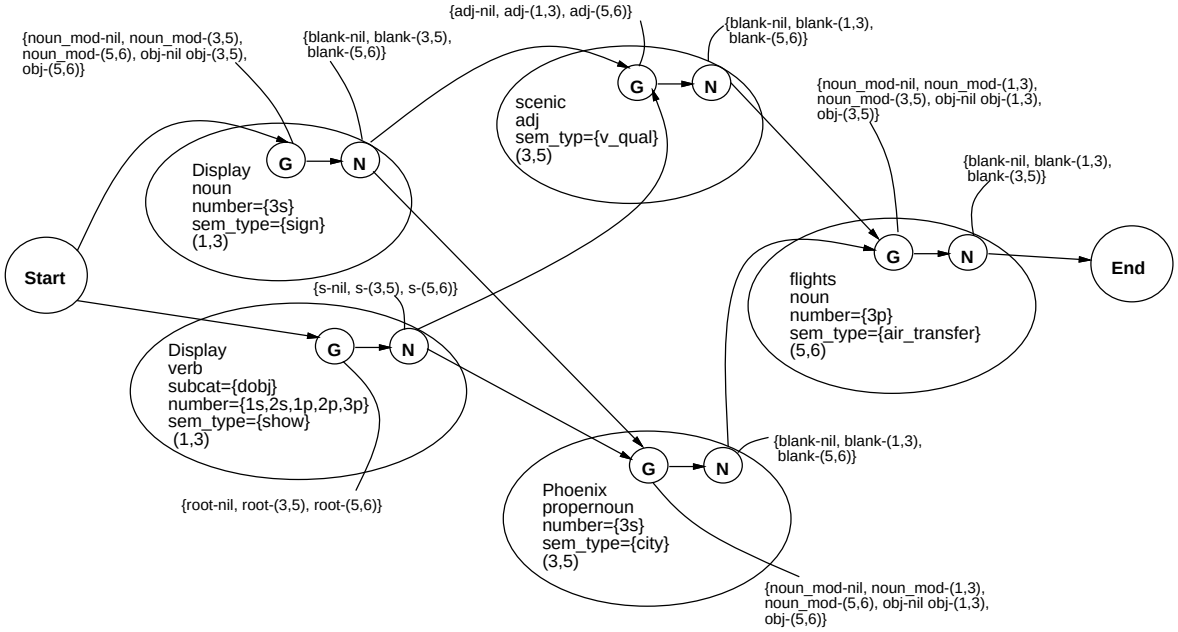


Figure 30: An SLCN constructed from the word graph in Figure 27.

parts of speech. All features are initially stored as sets on each word node. The role values assigned to each role are restricted to those that are allowed by the label_table (and the restriction that no word ever modifies itself).

The constraints for our grammar are shown in Figure 31. Application of the unary constraints over the SLCN in Figure 30 eliminates many of the role values, as shown in Figure 32. To apply binary constraints, it is necessary to create arcs (and their corresponding arc matrices) joining those roles that can appear in at least one common sentence hypothesis. Drawing all of the arcs would clutter the picture, so we will only depict those that are pertinent to pinpointing the parse of the sentence. However, the reader should be aware that no arcs join the roles of the noun form of *display* with the verb form of that word or the roles of *scenic* with the roles of *Phoenix*, because they cannot appear in the same sentence hypotheses. Figure 33 depicts the state of the matrices that are affected by the binary constraints. All other matrices contain only

```
UNARY CONSTRAINTS:
; A role value with label root modifies nothing.
(if (eq (lab x) root)
    (eq (mod x) nil))
; A role value with label blank modifies nothing.
(if (eq (lab x) blank)
    (eq (mod x) nil))
; A role value with label adj modifies a word to its right.
(if (eq (lab x) adj)
    (lt (pos x) (mod x)))
; A role value with label noun_mod modifies a word to its right.
(if (eq (lab x) noun_mod)
    (lt (pos x) (mod x)))
; A role value with label obj modifies a word to its left.
(if (eq (lab x) obj)
    (gt (pos x) (mod x)))
; A role value with label s modifies a word to its right.
(if (eq (lab x) s)
    (lt (pos x) (mod x)))

BINARY CONSTRAINTS:
; A role value for a verb with label s needs an obj which it governs.
(if (& (eq (lab x) s)
       (eq (rid y) governor)
       (eq (mod x) (pos y)))
   (& (eq (lab y) obj)
       (eq (mod y) (pos x))))
; A role value for a noun with label obj is governed by an s which needs it.
(if (& (eq (lab x) obj)
       (eq (rid y) need)
       (eq (mod x) (pos y)))
   (& (eq (lab y) s)
       (eq (mod y) (pos x))))
```

Figure 31: The unary and binary constraints for our example.

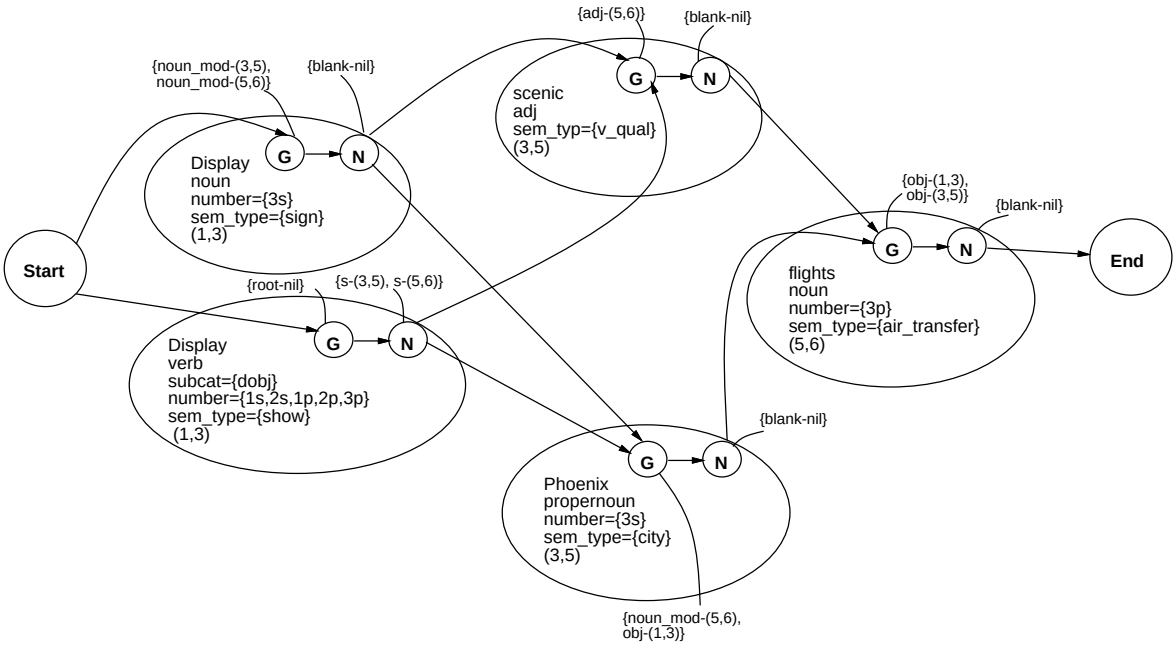


Figure 32: The SLCN after unary constraint propagation.

ones. Notice that we have numbered the roles in the SLCN. These numbers will allow us to discuss Prev-Support, Next-Support, Local-Prev-Support, and Local-Next-Support sets when filtering is applied next.

After the propagation of binary constraints, we must apply the filtering algorithm. First, the preprocessing procedure in Figure 23 is invoked to examine all of the arc matrices associated with the arcs and determine how the role values for each role are supported by elements associated with the roles they share an arc with. If a role value $a \in L_i$ is not supported by any of the role values associated with the role values of j, then the item $[(i,j),a]$ is placed on $List$. The procedure also calculates the Prev-Support, Next-Support, Local-Prev-Support, and Local-Next-Support sets for each role value. Once this preprocessing step is complete, the arc consistency procedure in Figure 24 loops until all items on $List$ have been processed. As items on $List$ are processed, new items can be inserted onto $List$.

After the preprocessing procedure has been executed, $List$ contains the several items to process: [(9,6),obj-(3,5)], [(9,8),obj-(3,5)], [(9,2),obj-(1,3)], [(7,2),obj-(1,3)], and [(4,5),s-(3,5)]. For example, the presence of [(9,6),obj-(3,5)] on $List$ indicates that the role value obj-(3,5) in role 9 is not supported by any of the role values associated with role 6.

Note that (9,8) and (9,6) are the only members of Local-Prev-Support(9,obj-(3,5)) and so once [(9,6),obj-(3,5)] and [(9,8),obj-(3,5)] are processed by the arc consistency procedure in Figure 24, the Local-Prev-Support(9,obj-(3,5)) set becomes empty indicating that obj-(3,5) is not a legal role value in any sentence hypothesis, and so it can be removed from role 9. In addition, all of the arcs attached to role 9 will eliminate their support for obj-(3,5), and determine whether the elimination of that role value makes it possible to add new items onto $List$. When the algorithm eliminates [(9,4),obj-(3,5)] from the SLCN, it notices that [(4,9),s-(3,5)] should also be added to the $List$ because the removal of obj-(3,5) causes s-(3,5) to become unsupported.

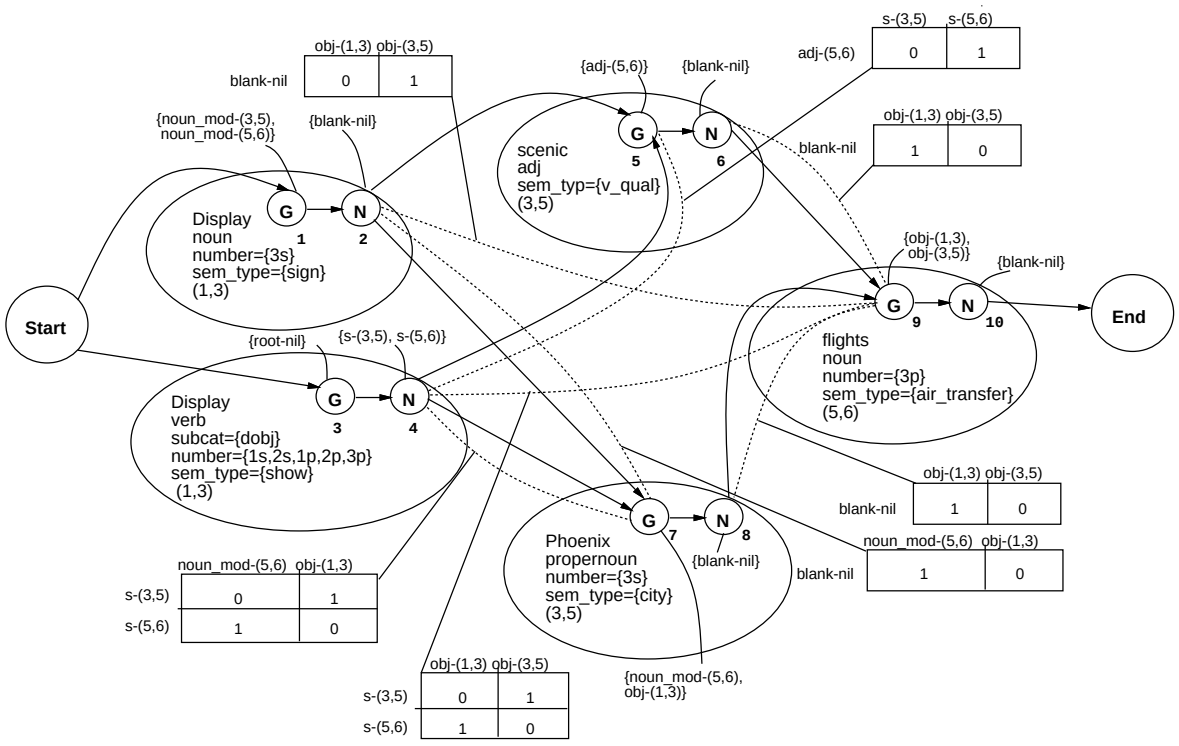


Figure 33: The SLCN after binary constraint propagation.

When [(4,9),s-(3,5)] is processed, the algorithm must eliminate s-(3,5)'s support support given Prev-Support[(4,9),s-(3,5)] = $\{(4,8),(4,6)\}$ and Next-Support[(4,9),s-(3,5)] = $\{(4,10)\}$. This is achieved by removing $(4,9)$ from Next-Support[(4,8),s-(3,5)] = $\{(4,9)\}$ and Next-Support[(4,6),s-(3,5)] = $\{(4,9)\}$, leaving both sets empty, causing [(4,8),s-(3,5)] and [(4,6),s-(3,5)] to be added to *List*. It must also remove $(4,9)$ from Prev-Support[(4,10),s-(3,5)] = $\{(4,9)\}$, causing [(4,10),s-(3,5)] to be added to the *List*. When [(4,6),s-(3,5)] is processed, the algorithm adds [(4,5),s-(3,5)] to the *List*, and when [(4,8),s-(3,5)] is processed, it adds [(4,7),s-(3,5)]. Because Local-Next-Support(4,s-(3,5)) = $\{(4,5),(4,7)\}$, once these two items are processed, s-(3,5) can be deleted from node 4. When the role value is eliminated, the algorithm is able to add [(7,4),obj-(1,3)] to *List*. When [(7,4),obj-(1,3)] and [(7,2),obj-(1,3)] have both been processed, Local-Prev-Support(7,obj-(1,3)), which initially was $\{(7,4),(7,2)\}$, becomes empty, and so obj-(1,3) can be eliminated from role 7.

When obj-(3,5) is deleted from role 9, it also causes a chain of events to eliminate blank-nil from role 2 (since role 9 is in every sentence hypothesis containing role 2), thereby eliminating the only role value from role 2. After additional processing, the role values on role 1 are also eliminated, leaving the word node for the noun form of *display* without any support. The word node can be pruned from the SLCN under these circumstances.

Once filtering is complete, the SLCN is in the state shown in Figure 34. The network is still ambiguous since it contains a verb with multiple number values and two paths through the network. To restrict the number for the word *display*, we would propagate a number feature constraint indicating that a command verb must agree with the second person pronoun, *you*. To propagate this constraint, the word node would be split into five similar nodes with the only difference being the value stored for the number feature. The arcs and arc matrices associated with the node would also be duplicated.

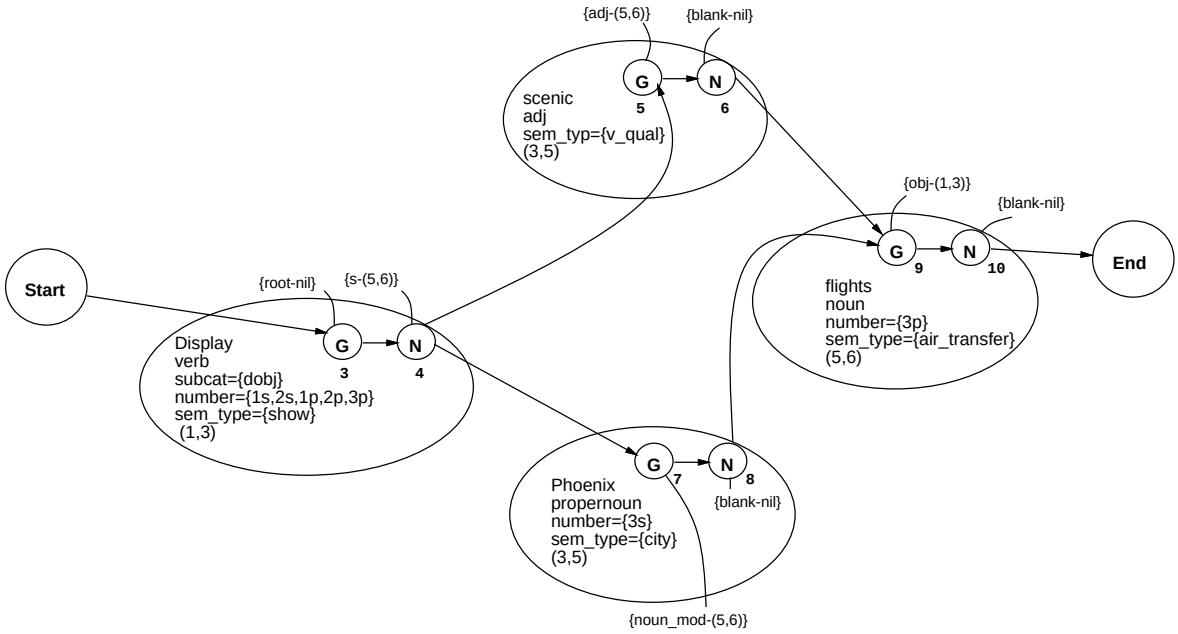


Figure 34: The SLCN after filtering.

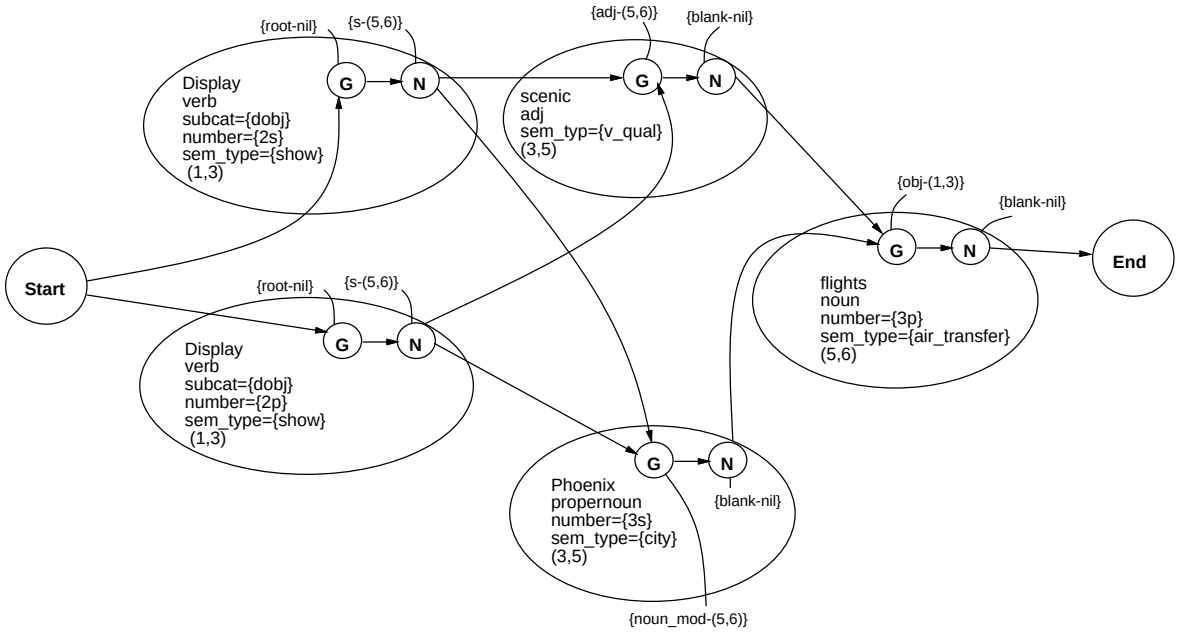


Figure 35: The SLCN after number constraints.

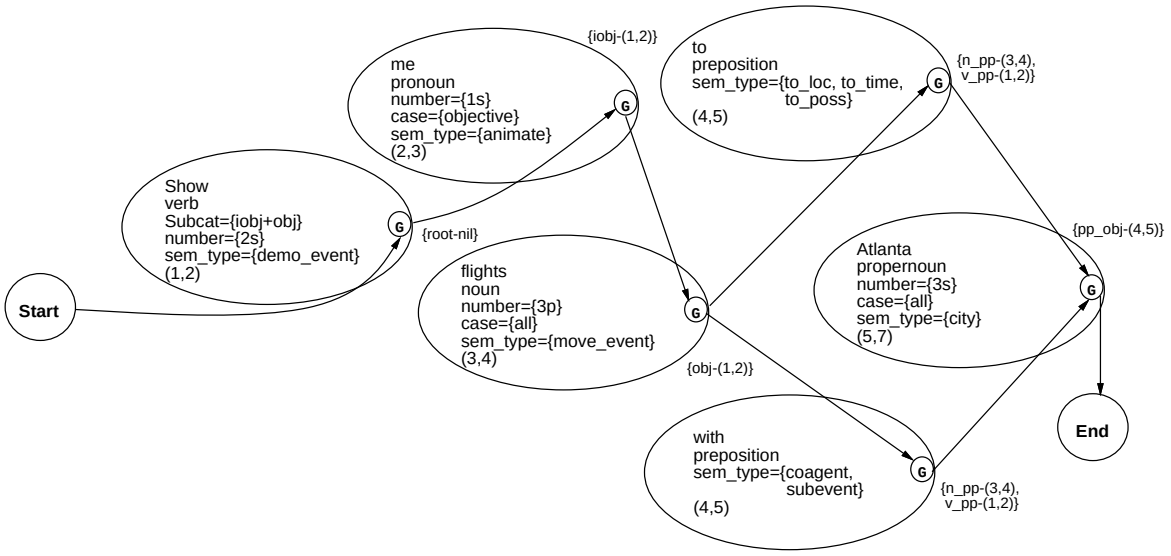


Figure 36: An SLCN after syntactic constraint propagation and filtering.

After node duplication, a constraint would be propagated requiring agreement with the word *you*, which has a number feature of either second person singular or plural. This constraint would eliminate all but two of the nodes for the verb, as shown in Figure 35. To determine which of the four remaining sentence hypotheses is correct requires the use of additional constraints.

Constraint propagation provides a uniform method for applying higher-level knowledge sources to prune a word graph. Because the constraints for each knowledge source can be developed independently, it is not as difficult to add another knowledge source to our parser. In the next section, we discuss the addition of semantic constraints.

3.7 Adding Semantic Constraints

Semantic features associated with the words in a dictionary can also be used to help disambiguate an SLCN. To understand the use of semantic constraints, consider the SLCN in Figure 36, assuming that syntactic constraints and syntactic feature constraints have been propagated, and filtering has been performed. This SLCN represents two distinct sentence hypotheses: *Show me flights to Atlanta* and *Show me flights with Atlanta*. Notice that the prepositions *to* and *with* are syntactically ambiguous since they can either modify the noun *flights* or the verb *show*. In addition, the prepositions are semantically ambiguous because they can fill a number of semantic functions in the sentence. By using the semantic features in this sentence and two simple semantic constraints, we are able to disambiguate this network entirely. The constraints are shown in figure 37.

To propagate the semantic constraints, we must duplicate the word nodes associated with the prepositions and assign each of them a unique semantic feature value, as shown in Figure 38. The first constraint uses the semantic type of the head word in the object of the preposition to restrict the semantic type of the preposition. After this constraint is propagated, *to* with to_loc as its semantic feature is the only prepositional word node remaining in the network. Because the semantic types of the other instances of *to* and *with* are incompatible with the semantic type of its object, they are eliminated from the

```
;; A location that is an object of a preposition modifies
;; a locative preposition.
(if (& (eq (lab x) pp_obj)
       (eq (mod x) (pos y))
       (eq (sem_type x) location)) ;; Note that a city is a location.
   (V (eq (sem_type y) to_loc)
       (eq (sem_type y) from_loc)
       (eq (sem_type y) at_loc)))
;; A to_loc preposition modifies a move_event.
(if (and (V (eq (lab x) n_pp)
            (eq (lab x) v_pp))
            (eq (mod x) (pos y))
            (eq (sem_type x) to_loc))
   (eq (sem_type y) move_event))
```

Figure 37: Semantic constraints for our example.

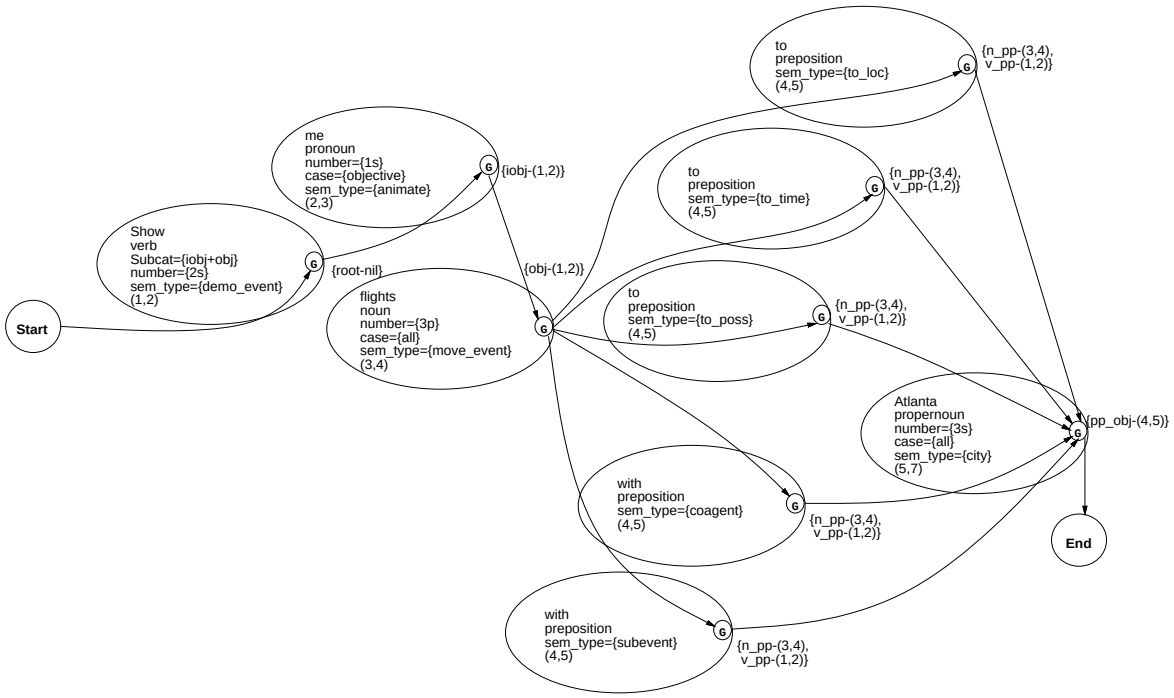


Figure 38: An SLCN just prior to semantic constraint propagation.

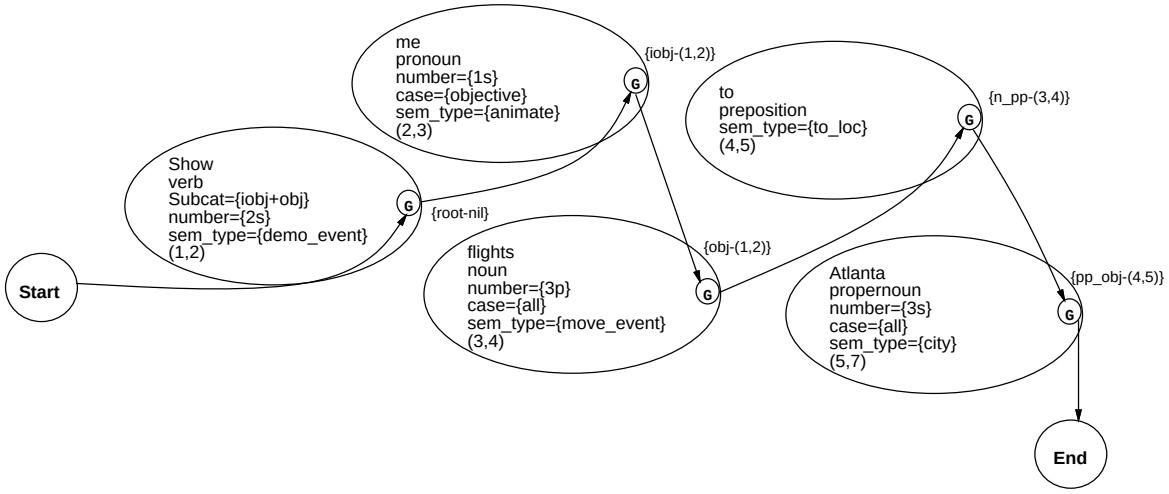


Figure 39: An SLCN after semantic constraint propagation and filtering.

network. The second semantic constraint can now restrict the type of word to which a to_loc preposition can attach. This constraint eliminates the role value v_pp-(1,2), giving a completely disambiguated word network, as shown in Figure 39.

This approach differs from the way that a context-free parser would incorporate semantics. There are several possible ways that semantics have been traditionally incorporated. The first method, the semantic grammar approach [5, 19, 39], introduces semantics directly into the production rules of the grammar. However, this tight coupling of semantic rules with syntactic rules typically increases the size and complexity of the grammar and reduces its understandability. Semantic grammars have been effective for limited domains, but they do not scale up well to larger systems [1]. The second method, the interleaved semantics approach [4, 40, 41], separates syntactic and semantic processing into two modules, where the semantic module eliminates constituents that are semantically anomalous from further consideration by the syntactic parser. The third method adds semantic tests to the grammar as an extra-grammatical test which is consulted during parsing [21, 31, 32]. A fourth approach is to use semantics as a post-processor to the parser in order to rule out the parses for semantically anomalous sentences. The last three methods do not use context-free production rules to handle semantic processing and so are more loosely coupled than the semantic grammar approach and more extendible. Our constraint-based approach differs from all four approaches because semantics is handled using constraints, just like syntax, but the approach is more loosely coupled than the semantic grammar approach. The syntactic constraints and semantic constraints can be developed independently.

We conducted a simple experiment to determine the effectiveness of syntactic and semantic constraints for reducing the ambiguity of word networks constructed from sets of BBN's N-best sentence hypotheses [35] from the ATIS database (Air Travel Information System). For this experiment, we selected twenty sets of 10 N-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 in which the duration of each node was determined by maintaining a syllable count through the utterance. Syntactic constraints were constructed first, then semantic constraints were constructed to further limit ambiguity [14]. Semantic constraints were relatively easy to create and incorporate into our parser. In fact, they were added to the grammar

without modifying a single syntactic rule. Preliminary work also suggests that prosodic constraints should be as simple to add to our grammar [12, 42].

Syntactic and semantic constraints are very useful for pruning syntactically or semantically anomalous word nodes from an SLCN. However, they do not, in many cases, sufficiently constrain the SLCN to a single sentence hypothesis with a single parse. Contextual information represents an additional knowledge source that can be exploited to reduce the ambiguity in an SLCN, as discussed in the next section.

3.8 Incorporating Pragmatic Constraints into SLCN Processing

Barwise and Perry [2] suggest that ambiguity of language is just another aspect of the efficiency of language. The fact that an expression can be used in more than one way is just another feature of that expression. Understanding an expression more fully comes at the cost of identifying the context. If we can identify the context in which the sentence occurs (i.e., the situation), then we can understand the expression more fully. Clearly, to provide a general model capable for smoothly handling all the contexts that a human being can is beyond the reach of current technology. However, we believe that by exploiting context as a feature of language, we can develop a constraint-based system capable of utilizing this information.

For the purposes of this paper, we define a context as a computer application. A user can interact with a language processor which interfaces with two or more applications as depicted in Figure 2. Here the function of the natural language interface is to interact with the correct application given the user's input. The process of parsing the input language should help to identify the correct context, and the identification of the correct context should help to disambiguate the user's input.

As an example of the usefulness of context, assume our system receives the word graph from our speech recognition module in Figure 27. Assume that there are three contexts, where c1 corresponds to an air travel database, c2 corresponds to a road map database, and c3 corresponds to a program for designing signs. Our system constructs an SLCN from the word graph by looking up information on each word in the dictionary which stores the same information as in our previous example along with additional information about each word's possible contexts. The contextually augmented dictionary for our example is shown in Figure 40. Many words can be shared across contexts, but some content words should appear in one context but not another. Just knowing which contexts are supported by a word provides a useful clue for selecting the correct context for an utterance.

The SLCN in Figure 41 is constructed from the word graph in Figure 27, assuming the grammar parameter file in Figure 28 and the lexicon in Figure 40. As before, two nodes are created for the word *display* because it has two possible parts of speech, and all remaining features are initially stored as sets on the word nodes, including contextual information.

By storing the allowable contexts with a word node, it is possible, without propagating even one constraint to eliminate a context from consideration. For a sentence to be a legal utterance (given our restricted sense of context), it must have at least

```
(display (category noun (context c3)
                        (number 3s)
                        (sem_type sign))
         (category verb (context c1 c2)
                        (number 1s 2s 1p 2p 3p)
                        (subcat dobj)
                        (sem_type show)))
(flight (category noun (context c1 c3)
                       (number 3s)
                       (sem_type air_transfer)))
(flights (root_word flight)
         (category noun (number 3p))))
(Phoenix (category propernoun (context c1 c3)
                              (number 3s)
                              (sem_type city)))
(scenic (category adj (context c2)
                      (sem_type v_qual)))
```

Figure 40: A contextually expanded dictionary for parsing our example.

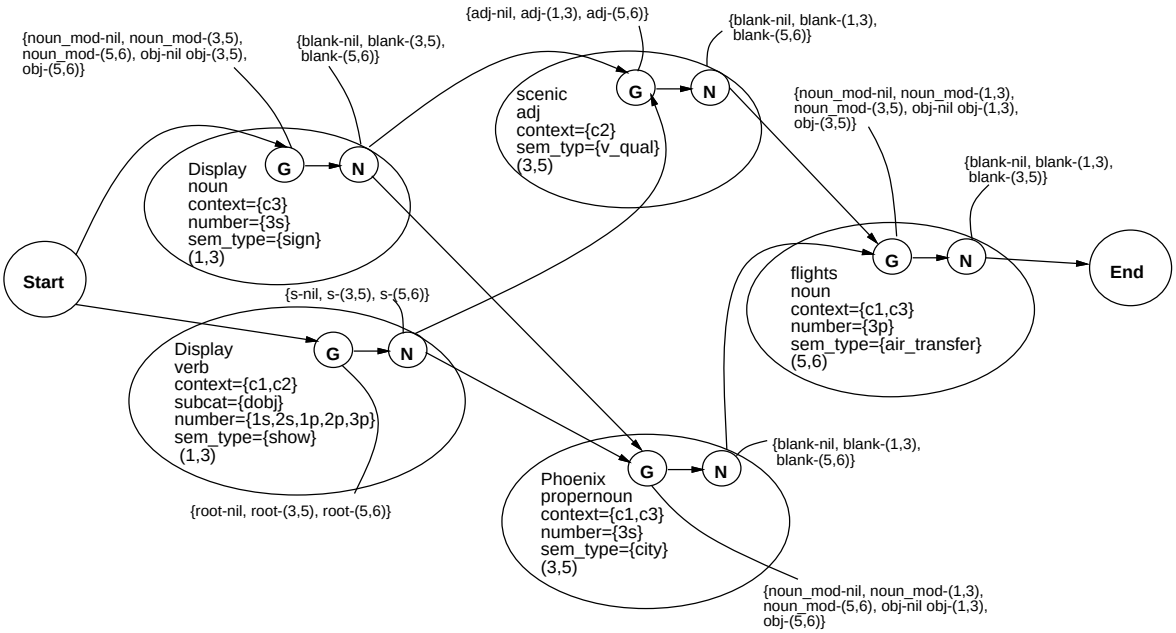


Figure 41: An SLCN constructed from the word graph in Figure 27.

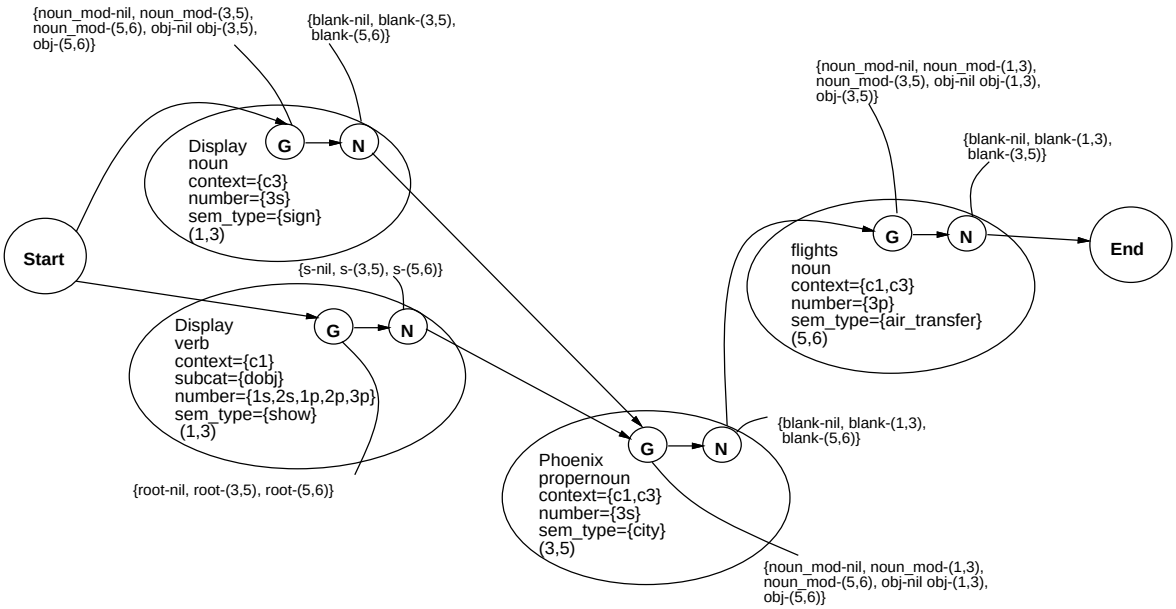


Figure 42: The SLCN pruned of words associated with context c2.

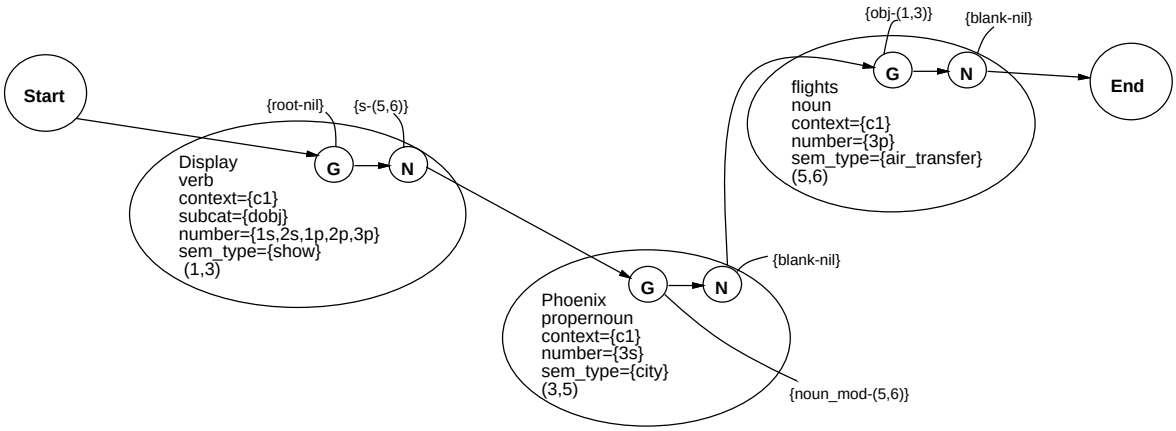


Figure 43: The SLCN after context-independent syntactic constraints are propagated, and filtering is performed.

one common context across all word nodes on the path. Also, for a context to be admissible, there must be a path from start to end containing word nodes that support that context. For example, in Figure 41, context c2 is not supported by the SLCN even before the propagation of constraints. If a context is not supported by at least one path through the network, then all word nodes that support only the disallowed context can be immediately pruned, as shown in Figure 42. Notice that c2 is also eliminated as a context for the verb form of *display*.

The next step in processing a contextual SLCN is to propagate the context-independent unary and binary syntactic constraints in Figure 31 and filter the SLCN. Once this step is completed, the SLCN is in the state depicted in Figure 43. Notice that following this, the word node for the noun form of *display* is eliminated; therefore, context c3 is no longer supported and has also been pruned from the network.

Next we select a set of constraints for a context-independent syntactic feature to propagate. For each node that has more than one value for the feature type, we

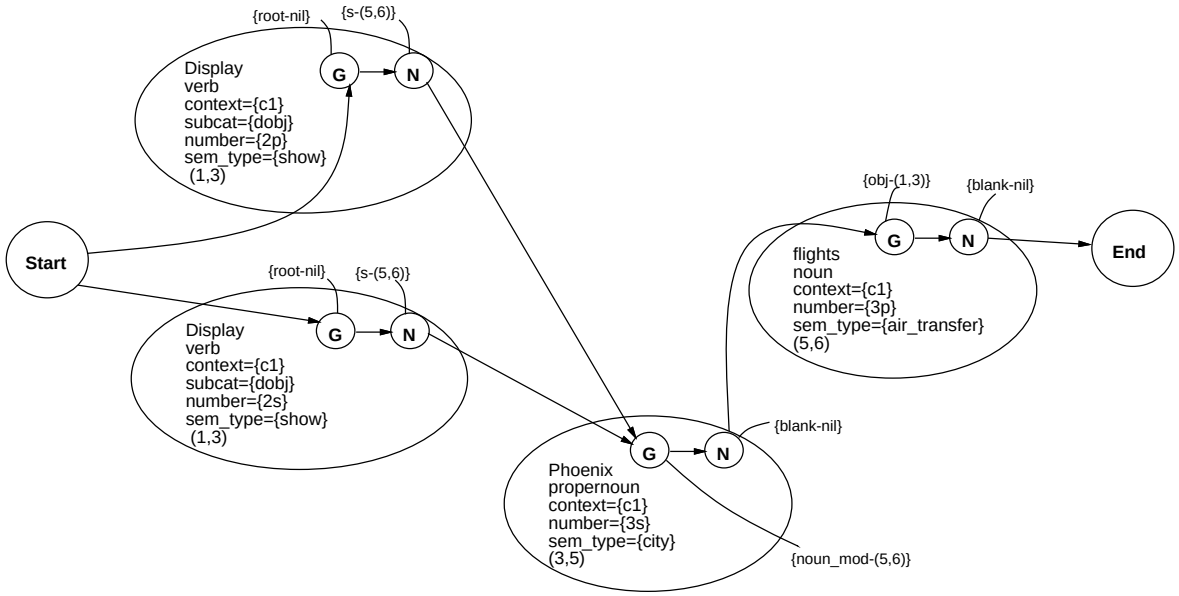


Figure 44: The SLCN after context-independent number feature constraints are propagated, and filtering is performed.

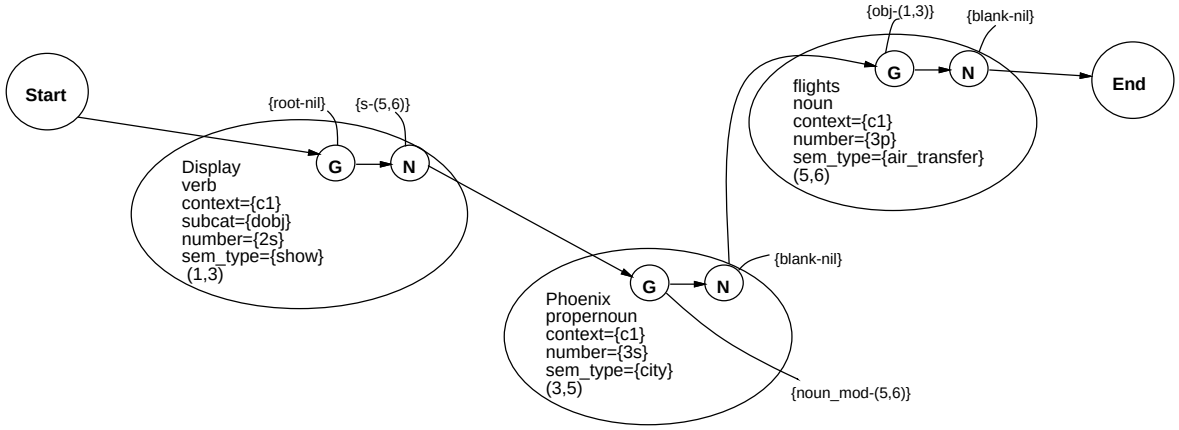


Figure 45: The SLCN with a single sentence hypothesis in context c1.

duplicate the node and assign it a single feature value. For example, if we propagate a number feature constraint, we would split the node corresponding to the verb *display*, and propagate the constraint to limit the number of the command verb to be 2s or 2p. Once this constraint is propagated, the network is in the state depicted in Figure 44.

At this point, we can utilize contextual constraints in an attempt to further refine the parse. One constraint that would work well in the case of context c1, c2, or c3 is to restrict the number of a command verb to second person singular (since the computer is singular). Because this constraint is common to all of the contexts, this constraint could be applied to a contextually ambiguous network without creating separate nodes for each context. However, in the case of our example, the context has already been isolated, and we apply the constraint to simply eliminate the number ambiguity, with the resulting SLCN depicted in Figure 45. This sentence now has a single parse and is valid only in context c1.

We have observed that semantic features and context are often highly correlated.

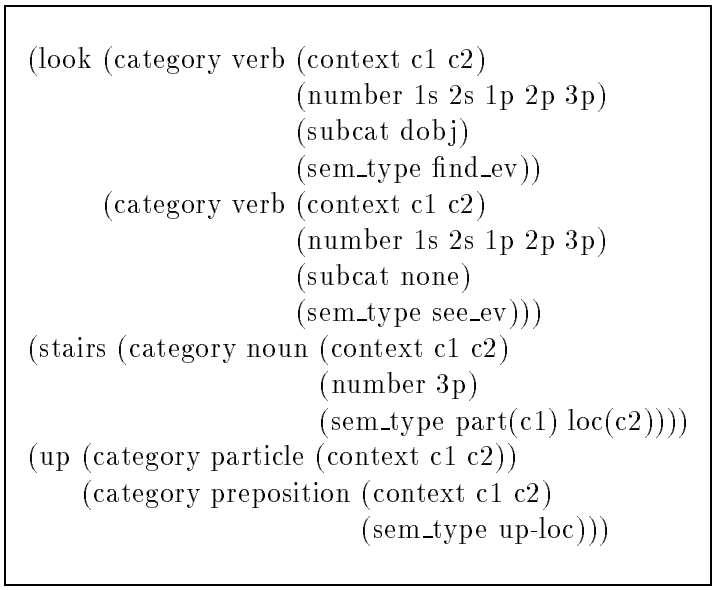


Figure 46: Another contextually expanded dictionary.

Hence, rather than require all semantic types associated with a word to be allowed in each of its contexts, we provide a mechanism for indicating which semantic features are defined for each context in the dictionary. Consider the dictionary in Figure 46. Notice that the semantic features for the word *stairs* are annotated with a specific context, even though the word is defined in both contexts. If c1 is a parts database and c2 is a mobile robot with an on-board phone, then the word *stairs* should have very different semantic features for each of the contexts. In the first case, the stairs represent a part in the database; whereas, in the second, the stairs represent a location. In contrast, the word *look* is also defined in both contexts, but because its semantic features can appear in both contexts, they are not annotated with contextual information.

Associating contextual information with a semantic feature allows us to parse the SLCN for *Look up stairs* in two different ways, depending on which context is chosen. Figure 47 depicts an SLCN for this sentence after syntactic constraints, number and subcategorization constraints, and filtering. When *look* is intransitive, it is incompatible with *up* as a particle because that verb form cannot take an object. This incompatibility is indicated in the parse of the sentence in two ways. First, no role values in the governor role of the particle *up* support the role value s-nil in the need role of the intransitive form of *look*. Second, the arc between the governor role of the particle *up* and the need role of the intransitive verb is removed from the Local-Next-Support set for s-nil. Since s-nil is the only role value in the need role of the intransitive verb, we know that *up* as a particle and *look* as an intransitive verb cannot participate in a common sentence hypothesis; hence, we eliminate the directed arc to simplify the figure. We have also removed the directed edge between the transitive form of *look* and the preposition *up* to indicate that they cannot participate in a common sentence. Notice that there are two paths through the network, one using *up* as a preposition, the other using it as a particle. Each word in the network can be used in either context c1 or c2; however, the semantic feature associated with the word *stairs* is different for each context.

To prepare for context-independent semantic constraint propagation, the nodes with multiple semantic features are duplicated and assigned a single semantic feature as shown in Figure 48. Notice that the two nodes for *stairs* each participate in a different

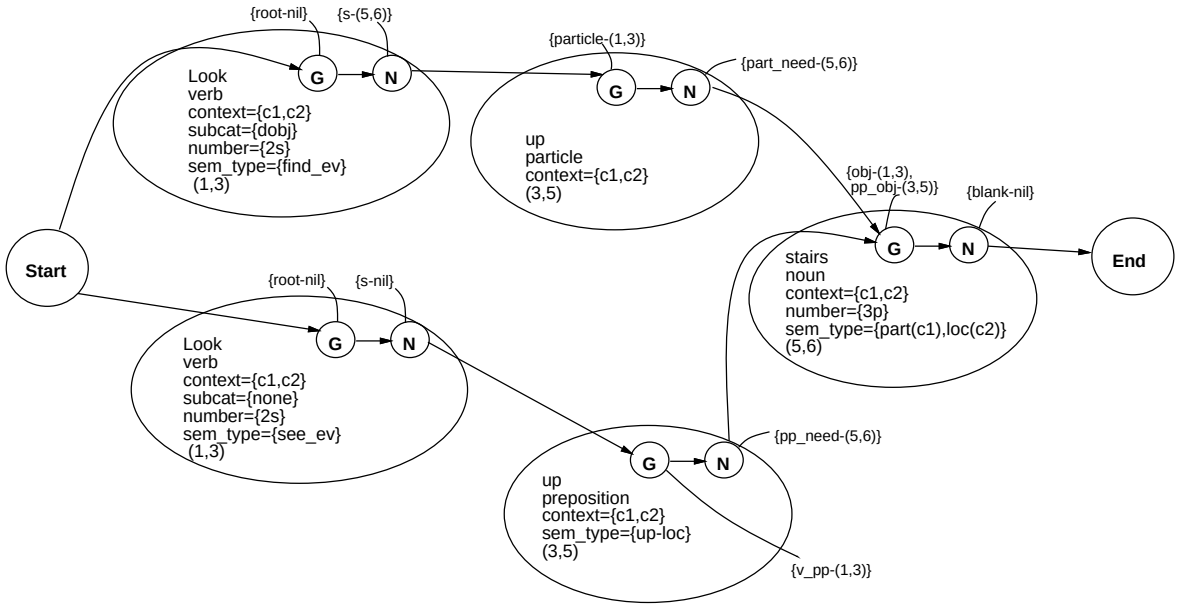


Figure 47: An SLCN after the propagation of syntactic, number, and subcategorization constraints and filtering.

context. If we propagate a context-independent semantic constraint requiring that the object of an up-loc be a loc (i.e., a location), then c1 becomes incompatible with the parse where *up* is a preposition. This incompatibility would be indicated in the matrix associated with the arc between the governor role of *stairs* in context c1 and the need role of the preposition. It would also be indicated by the fact that the arc between those two roles would be removed from the Local-Next-Support set associated the role value pp_need-(5,6), the only role value for that role.

If we propagate context-specific semantic constraints, further refinement is possible. In preparation for context-specific constraint propagation, the nodes are duplicated and assigned a single context as shown in Figure 49. Because the nodes in c1 do not appear in any sentence hypotheses with nodes in c2, directed edges appear only between nodes with the same context. Notice that there is only one node for the preposition *up* and for the intransitive form of *look*, both in c2. The nodes for c1 are pruned after the nodes are split because the context-independent semantic constraint (concerning location) makes it impossible for the nodes to participate in a legal sentence hypothesis in context c1. Now, if we propagate a constraint which requires the object of *look up* to be a part in c1 and a number in c2, the SLCN ends up in the state shown in Figure 50. Although the SLCN is still ambiguous, it contains a single parse for the sentence in each context.

3.9 Handling Multiple Knowledge Sources

A language model that can utilize a variety of knowledge sources to disambiguate spoken language is more likely to achieve a level of accuracy comparable to humans. We have described a constraint-based system which is able to utilize a variety of knowledge sources to disambiguate speech. Knowledge sources commonly used in speech understanding are shown in Figure 1. There is good evidence that there is an implicit ordering among these knowledge sources such that one type of information must be

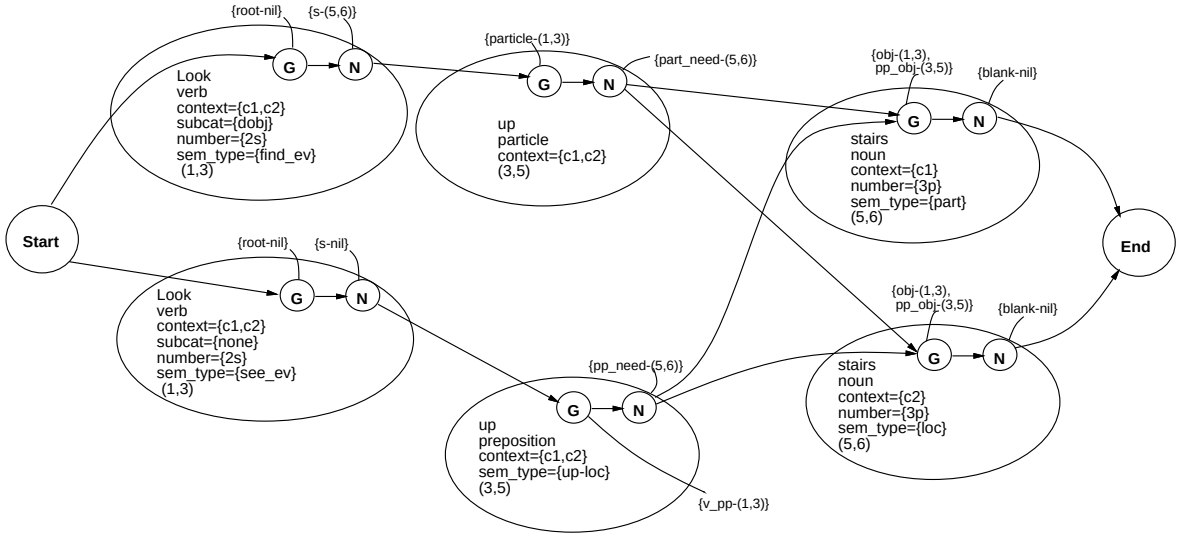


Figure 48: An SLCN before context-free semantic constraint propagation.

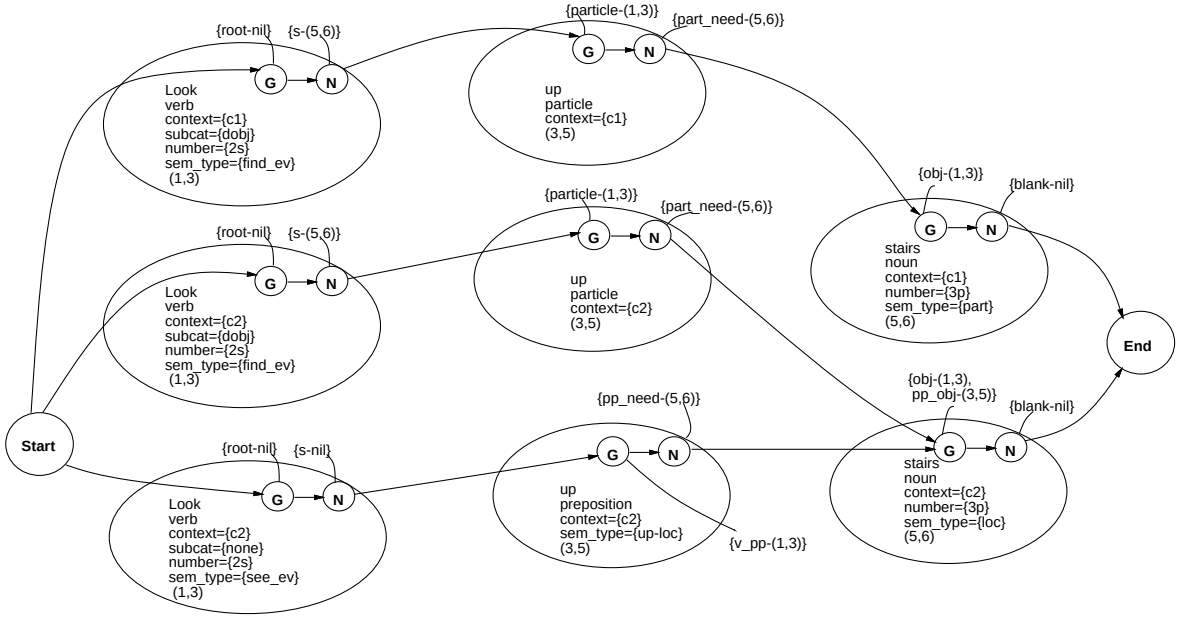


Figure 49: An SLCN before contextual constraint propagation.

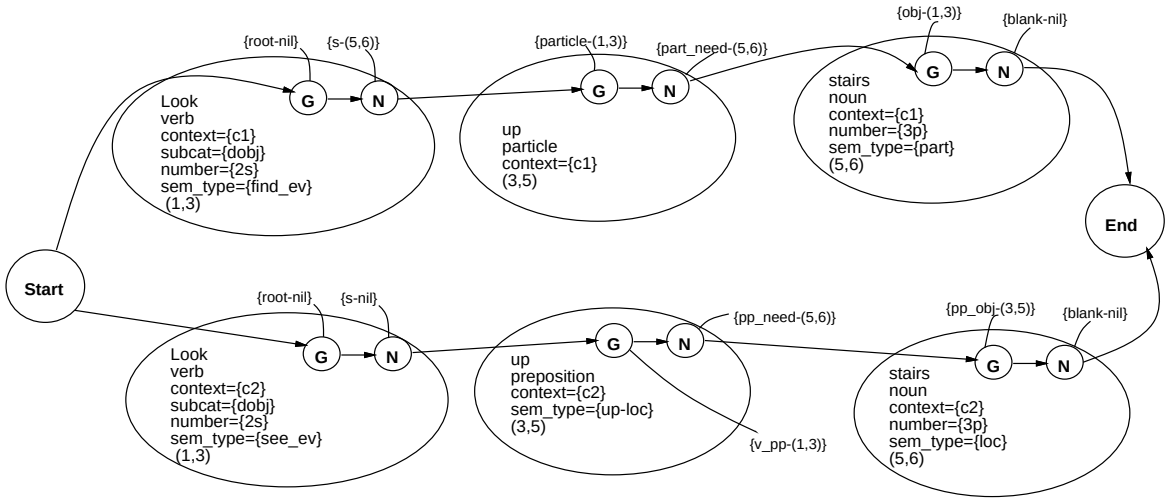


Figure 50: An SLCN after contextual constraint propagation.

available before it makes sense to progress to the next level. If we combine two ordered knowledge sources together in a single module, the resulting system can be difficult to understand and can often become intractable. For example, combining prosodic processing [3] with CFG grammar rules typically increases the size and complexity of the grammar and reduces its understandability. A similar problem holds for semantic grammars [1].

The ordering of knowledge sources in Figure 1 suggests that there should be a way to order constraints to limit the combinatorial explosion of managing all knowledge sources at once. Below we enumerate the steps of our algorithm:

1. Create an SLCN for the word graph by using a dictionary and eliminate impossible contexts,
2. Propagate context-independent syntactic unary constraints (no feature testing).
3. Construct arcs for binary constraint propagation.
4. Propagate context-independent syntactic binary constraints and filter the SLCN.
5. Loop for each set of syntactic feature type constraints:
   - Duplicate the nodes with more than a single value for that feature (role values, arcs, and matrices).
   - Propagate the constraints for that feature type.
   - Filter the network.
   - Eliminate impossible contexts.
6. For semantic feature constraints:
   - Duplicate the nodes with more than a single semantic feature value.
   - Propagate the semantic constraints.
   - Filter the network.
   - Eliminate impossible contexts.
7. For contextual constraints:
   - Duplicate the nodes with more than a single context.
   - Propagate the contextual constraints.
   - Filter the network.
   - Eliminate impossible contexts.

Depending on the number of contexts and the degree of sharing between contexts, we could utilize a courser granularity for propagating context-specific constraints. For example, if contexts c1 and c2 have many common constraints, then splitting a node that is ambiguous between c1 and c2 makes no sense until after the shared constraints are propagated[10]. Constraint parsing does provide a level of control to allow a system designer to avoid duplication of effort.

4 Conclusion

We have described a system which is capable of utilizing a variety of knowledge sources to select the most appropriate parse for a spoken sentence. These knowledge sources include syntax, semantics, and context. The parser uses a uniform mechanism, constraint propagation, to apply these high-level knowledge sources to prune a word graph provided by the speech recognizer. Constraints for different knowledge sources can be developed independently and used incrementally when parsing a sentence. Our constraint-based parser should be useful for developing a spoken language interface to several computer applications, where each application defines its own context.

We have introduced a method for incorporating context-specific constraints into the parsing process. However, we have not discussed how discourse structure [8, 9] would impact the system. It would be very helpful for tracking interactions between a user and the general purpose interface to multiple applications and is a topic which we plan to study in the future. Also, it is important to handle multiple valid interacting contexts. For example, suppose you want to find the cheapest way to travel from Boston to Los Angeles in two days. The solution to this problem might require that the travel task be broken down into subtasks which could be handled by both an air travel and a bus travel program. This is something that our system cannot handle because the only domain knowledge that is available to the natural language front-end is used to determine whether the request is well-formed given the constraints for the various databases. We could, of course, create a meta-database which combines air and ground travel databases with appropriate domain knowledge to achieve these needs. This is a topic for future research.

5 Acknowledgments

This work was supported in part by Purdue Research Foundation, Intel, a General Electric Faculty of the Future grant, and NSF grant number IRI-9011179. We would like to thank Carl Mitchell for his critical reading of this paper and the referees for their helpful comments.

[10] Although it is necessary to split the word node into two nodes, one for contexts c1 and c2 and one for the other contexts.

References

[1] J. Allen. *Natural Language Understanding.* The Benjamin Cummings Publishing Company, Inc., Menlo Park, CA, 1987.

[2] J. Barwise and J. Perry. *Situations and Attitudes.* The MIT Press, Cambridge, MA, 1983.

[3] J. Bear and P. Price. Prosody, syntax, and parsing. In *Proceedings of the 28th Annual Meeting of Association for Computational Linguistics*, pages 17–22, 1990.

[4] R. J. Bobrow and B. L. Webber. Knowledge representation for syntactic/semantic processing. In *The Proceedings of the National Meeting of American Association for Artificial Intelligence*, pages 316–323, 1980.

[5] J. S. Brown and R. R. Burton. Multiple representations of knowledge for tutorial reasoning. In D. G. Bobrow and A. Collins, editors, *Representation and Understanding.* Academic Press, New York, 1975.

[6] A. Copestake and K. Sparck-Jones. Natural language interfaces to databases. *The Knowledge Engineering Review*, 4(5):225–249, 1990.

[7] P. Dey and B. R. Bryant. Lexical ambiguity in tree adjoining grammars. *Information Processing Letters*, 34:65–69, 1990.

[8] B. J. Grosz. The representation and use of focus in a system of understanding dialogs. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 67–76, 1977.

[9] B. J. Grosz and C. Sidner. Attention, intention, and the structure of discourse. *Computational Linguistics*, 12:175–204, 1986.

[10] M. P. Harper. Ambiguous noun phrases in logical form. *Computational Linguistics*, 18(4):419–465, 1992.

[11] 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.

[12] M. P. Harper, R. A. Helzerman, and C. B. Zoltowski. Constraint parsing: A powerful framework for text-based and spoken language processing. Technical Report EE-91-34, Purdue University, School of Electrical Engineering, West Lafayette, IN, 1991.

[13] 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 *Proceedings of the AAAI-94 Workshop on the Integration of Natural Language and Speech Processing*, 1994.

[14] 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.

[15] R. A. Helzerman. PARSEC: A framework for parallel natural language understanding. Master's thesis, Purdue University, School of Electrical Engineering, West Lafayette, IN, 1993.

[16] 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*, pages 209–217, August 1992.

[17] R. A. Helzerman and M. P. Harper. An approach to multiply segmented constraint satisfaction problems. In *Proceedings of the National Conference on Artificial Intelligence*, July 1994.

[18] R. A. Helzerman and M. P. Harper. MUSE CSP: An extension to the constraint satisfaction problem. Technical Report EE-94-8, Purdue University, School of Electrical Engineering, West Lafayette, IN, 1994.

[19] G. G. Hendrix, E. Sacerdoti, D. Sagalowicz, and J. Slocum. Developing a natural language interface to complex data. *ACM Transactions on Database Systems*, 3(2):105–147, 1978.

[20] A. K. Joshi, L. S. Levy, and M. Takahashi. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10:136–163, 1975.

[21] M. Kay. Parsing in functional unification grammar. In D. R. Dowty, L. Kartunnen, and A. Zwickey, editors, *Natural Language Parsing*, pages 251–278. Cambridge University Press, Cambridge, 1985.

[22] S. R. Kosaraju. Speed of recognition of context-free languages by array automata. *SIAM Journal of Computing*, 4(3):331–340, September 1975.

[23] V. Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 13(1):32–44, 1992.

[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 the Association for Computational Linguistics*, pages 31–38, 1990.

[27] R. Mohr and T. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.

[28] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Science*, 7:95–132, 1974.

[29] 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*, pages 112–116, 1987.

[30] 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.

[31] 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.

[32] J. J. Robinson. DIAGRAM: A grammar for dialogues. *Communications of the ACM*, 25(1):27–47, 1982.

[33] W. Ruzzo. Tree-size bounded alternation. *Journal of Computers and System Sciences*, 21:218–235, 1980.

[34] M. N. Sap and D. R. McGregor. Natural language interfaces to databases: State of the art. Technical Report IKBS-7-92, University of Strathclyde, U. K., 1992.

[35] 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 International Conference on Acoustics, Speech, and Signal Processing*, pages 81–84, April 1990.

[36] K. Vijay-Shanker and A. K. Joshi. Some computational properties of tree adjoining grammars. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, pages 82–93, 1985.

[37] K. Vijay-Shanker, 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*, pages 104–111, 1987.

[38] D. L. Waltz. Understanding line drawings of scenes with shadows. In P. H. Winston, editor, *The Psychology of Computer Vision*, pages 19–91. McGraw Hill, Cambridge, MA, 1975.

[39] D. L. Waltz and B. A. Goodman. Writing a natural language data base system. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 144–150, 1977.

[40] T. Winograd. *Understanding Natural Language*. Academic Press, New York, 1972.

[41] W. Woods. Cascaded ATN grammars. *American Journal of Computational Linguistics*, 6(1):1–12, 1980.

[42] C. B. Zoltowski. Current research in the development of a spoken language understanding system using PARSEC. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pages 353–354, 1991.

[43] 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 Processing*, pages 249–252, October 1992.