STRUCTURAL DISAMBIGUATION WITH CONSTRAINT PROPAGATION

Hiroshi Maruyama
IBM Research, Tokyo Research Laboratory
5-19 Sanbancho, Chiyoda-ku,
Tokyo 102 Japan
maruyama@jpntscvm.bitnet

Abstract

We present a new grammatical formalism called *Constraint Dependency Grammar* (CDG) in which every grammatical rule is given as a constraint on word-to-word modifications. CDG parsing is formalized as a constraint satisfaction problem over a finite domain so that efficient constraint-propagation algorithms can be employed to reduce structural ambiguity without generating individual parse trees. The weak generative capacity and the computational complexity of CDG parsing are also discussed.

1 INTRODUCTION

We are interested in an efficient treatment of structural ambiguity in natural language analysis. It is known that "every-way" ambiguous constructs, such as prepositional attachment in English, have a Catalan number of ambiguous parses (Church and Patil 1982), which grows at a faster than exponential rate (Knuth 1975). A parser should be provided with a disambiguation mechanism that does not involve generating such a combinatorial number of parse trees explicitly.

We have developed a parsing method in which an intermediate parsing result is represented as a data structure called a *constraint network*. Every solution that satisfies all the constraints simultaneously corresponds to an individual parse tree. No explicit parse trees are generated until ultimately necessary. Parsing and successive disambiguation are performed by adding new constraints to the constraint network. Newly added constraints are efficiently propagated over the network by *Constraint Propagation* (Waltz 1975, Montanari 1976) to remove inconsistent values.

In this paper, we present the basic ideas of a formal grammatical theory called *Constraint Dependency Grammar* (CDG for short) that makes this parsing technique possible. CDG has a reasonable time bound in its parsing, while its weak generative capacity is strictly greater than that of *Context Free Grammar* (CFG).

We give the definition of CDG in the next section. Then, in Section 3, we describe the parsing method based on constraint propagation, using a step-by-step example. Formal properties of CDG are discussed in Section 4.

2 CDG: DEFINITION

Let a sentence $s = w_1 w_2 \ldots w_n$ be a finite string on a finite alphabet Σ. Let $R = \{r_1, r_2, \ldots, r_k\}$ be a finite set of *role-ids*. Suppose that each word i in a sentence s has k-different roles $r_1(i), r_2(i), \ldots, r_k(i)$. Roles are like variables, and each role can have a pair <a,d> as its value, where the *label* a is a member of a finite set $L = \{a_1, a_2, \ldots, a_l\}$ and the *modifiee* d is either $1 \leq d \leq n$ or a special symbol `nil`. An analysis of the sentence s is obtained by assigning appropriate values to the $n \times k$ roles (we can regard this situation as one in which each word has a frame with k slots, as shown in Figure 1).

An *assignment* A of a sentence s is a function that assigns values to the roles. Given an assignment A, the label and the modifiee of a role x are determined. We define the following four functions to represent the various aspect of the role x, assuming that x is an r_j-role of the word i:

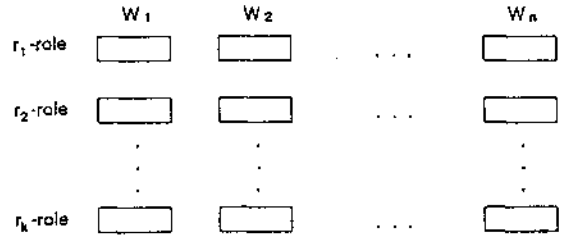

Figure 1: Words and their *roles*.

- $pos(x) \stackrel{\text{def}}{=}$ the position i
- $rid(x) \stackrel{\text{def}}{=}$ the role id r_j
- $lab(x) \stackrel{\text{def}}{=}$ the label of x
- $mod(x) \stackrel{\text{def}}{=}$ the modifiee of x

We also define $word(i)$ as the terminal symbol occurring at the position i.[1]

An individual grammar $G =< \Sigma, R, L, C >$ in the CDG theory determines a set of possible assignments of a given sentence, where

- Σ is a finite set of terminal symbols.
- R is a finite set of role-ids.
- L is a finite set of labels.
- C is a constraint that an assignment A should satisfy.

A constraint C is a logical formula in a form

$$\forall x_1 x_2 ... x_p : role; P_1 \& P_2 \& ... \& P_m$$

where the variables $x_1, x_2, ..., x_p$ range over the set of roles in an assignment A and each subformula P_i consists only of the following vocabulary:

- Variables: $x_1, x_2, ..., x_p$
- Constants: elements and subsets of $\Sigma \cup L \cup R \cup \{\texttt{nil}, 1, 2, ...\}$
- Function symbols: $word(), pos(), rid(), lab()$, and $mod()$
- Predicate symbols: $=, <, >$, and $\in$
- Logical connectors: $\&, |, \neg$, and $\Rightarrow$

Specifically, we call a subformula P_i a *unary constraint* when P_i contains only one variable, and a *binary constraint* when P_i contains exactly two variables.

The semantics of the functions have been defined above. The semantics of the predicates and the logical connectors are defined as usual, except that comparing an expression containing `nil` with another value by the inequality predicates always yields the truth value **false**.

These conditions guarantee that, given an assignment A, it is possible to compute whether the values of $x_1, x_2, ..., x_p$ satisfy C in a constant time, regardless of the sentence length n.

Definition

- The *degree* of a grammar G is the size k of the role-id set R.
- The *arity* of a grammar G is the number of variables p in the constraint C.

  Unless otherwise stated, we deal with only arity=2 cases.
- A nonnull string s over the alphabet Σ is *generated* iff there exits an assignment A that satisfies the constraint C.
- $L(G)$ is a *language* generated by the grammar G iff $L(G)$ is the set of all sentences generated by a grammar G.

Example

Let us consider $G1 =< \Sigma 1, R1, L1, C1 >$ where

- $\Sigma 1 = \{\texttt{D}, \texttt{N}, \texttt{V}\}$
- $R1 = \{governor\}$
- $L1 = \{\texttt{DET}, \texttt{SUBJ}, \texttt{ROOT}\}$
- $C1 = \forall xy : role; P1.$

The formula $P1$ of the constraint $C1$ is the conjunction of the following four subformulas (an informal description is attached to each constraint):

(G1-1) $word(pos(x))$=D $\Rightarrow$ ($lab(x)$=DET, $word(mod(x))$=N, $pos(x) < mod(x)$)

"A determiner (D) modifies a noun (N) on the right with the label DET."

[1] In this paper, when referring to a word, we purposely use the *position* (1,2,...,n) of the word rather than the word itself $(w_1, w_2, ..., w_n)$, because the same word can occur in many different positions in a sentence. For readability, however, we sometimes use the notation $word_{position}$.

Role	Value
governor("a_1")	<DET,2>
governor("dog_2")	<SUBJ,3>
governor("$runs_3$")	<ROOT,nil>

Figure 2: Assignment Satisfying (G1-1) to (G1-4)

(G1-2) $word(pos(x))$=N $\Rightarrow$ ($lab(x)$=SUBJ, $word(mod(x))$=V, $pos(x) < mod(x)$)

"A noun modifies a verb (V) on the right with the label SUBJ."

(G1-3) $word(pos(x))$=V $\Rightarrow$ ($lab(x)$=ROOT, $mod(x)$=nil)

"A verb modifies nothing and its label should be ROOT."

(G1-4) ($mod(x)=mod(y)$, $lab(x)=lab(y)$) $\Rightarrow$ $x=y$

"No two words can modify the same word with the same label."

Analyzing a sentence with $G1$ means assigning a label-modifiee pair to the only role "governor" of each word so that the assignment satisfies (G1-1) to (G1-4) simultaneously. For example, sentence (1) is analyzed as shown in Figure 2 provided that the words "a," "dog," and "runs" are given parts-of-speech D, N, and V, respectively (the subscript attached to the words indicates the position of the word in the sentence).

(1) A_1 dog_2 $runs_3$.

Thus, sentence (1) is generated by the grammar $G1$. On the other hand, sentences (2) and (3) are not generated since there are no proper assignments for such sentences.

(2) A runs.

(3) Dog dog runs.

We can graphically represent the parsing result of sentence (1) as shown in Figure 3 if we interpret the *governor* role of a word as a pointer to the syntactic governor of the word. Thus, the syntactic structure produced by a CDG is usually a *dependency structure* (Hays 1964) rather than a *phrase structure*.

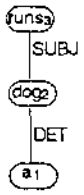

Figure 3: Dependency tree

3 PARSING WITH CONSTRAINT PROPAGATION

CDG parsing is done by assigning values to $n \times k$ roles, whose values are selected from a finite set $L \times \{1, 2, \ldots, n, \mathtt{nil}\}$. Therefore, CDG parsing can be viewed as a constraint satisfaction problem over a finite domain. Many interesting artificial intelligence problems, including graph coloring and scene labeling, are classified in this group of problems, and much effort has been spent on the development of efficient techniques to solve these problems. *Constraint propagation* (Waltz 1975, Montanari 1976), sometimes called *filtering*, is one such technique. One advantage of the filtering algorithm is that it allows new constraints to be added easily so that a better solution can be obtained when many candidates remain. Usually, CDG parsing is done in the following three steps:

1. Form an initial constraint network using a "core" grammar.
2. Remove local inconsistencies by filtering.
3. If any ambiguity remains, add new constraints and go to Step 2.

In this section, we will show, through a step-by-step example, that the filtering algorithms can be effectively used to narrow down the structural ambiguities of CDG parsing.

The Example

We use a PP-attachment example. Consider sentence (4). Because of the three consecutive prepositional phrases (PPs), this sentence has many structural ambiguities.

(4) Put the block on the floor on the table in the room.

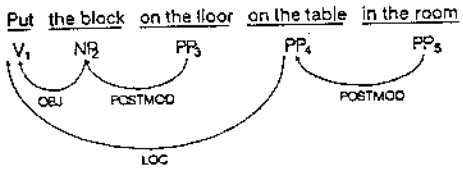

Figure 4: Possible dependency structure

One of the possible syntactic structures is shown in Figure 4[2].

To simplify the following discussion, we treat the grammatical symbols V, NP, and PP as terminal symbols (words), since the analysis of the internal structures of such phrases is irrelevant to the point being made. The correspondence between such simplified dependency structures and the equivalent phrase structures should be clear. Formally, the input sentence that we will parse with CDG is (5).

(5) $\mathtt{V}_1$ $\mathtt{NP}_2$ $\mathtt{PP}_3$ $\mathtt{PP}_4$ $\mathtt{PP}_5$

First, we consider a "core" grammar that contains purely syntactic rules only. We define a CDG $G2a =< \Sigma 2, R2, L2, C2 >$ as follows:

- $\Sigma 2 = \{\mathtt{V},\mathtt{NP},\mathtt{PP}\}$
- $R2 = \{governor\}$
- $L2 = \{\mathtt{ROOT},\mathtt{OBJ},\mathtt{LOC},\mathtt{POSTMOD}\}$
- $C2 = \forall xy : role; P2,$

where the formula $P2$ is the conjunction of the following unary and binary constraints :

(G2a-1) $word(pos(x))$=PP $\Rightarrow$ ($word(mod(x)) \in$ {PP,NP,V}, $mod(x) < pos(x)$)

"A PP modifies a PP, an NP, or a V on the left."

(G2a-2) $word(pos(x))$=PP, $word(mod(x)) \in$ {PP,NP} $\Rightarrow$ $lab(x)$=POSTMOD

"If a PP modifies a PP or an NP, its label should be POSTMOD."

(G2a-3) $word(pos(x))$=PP, $word(mod(x))$=V $\Rightarrow$ $lab(x)$=LOC

"If a PP modifies a V, its label should be LOC."

[2]In linguistics, arrows are usually drawn in the opposite direction in a dependency diagram: from a governor (modifiee) to its dependent (modifier). In this paper, however, we draw an arrow from a modifier to its modifiee in order to emphasize that this information is contained in a modifier's role.

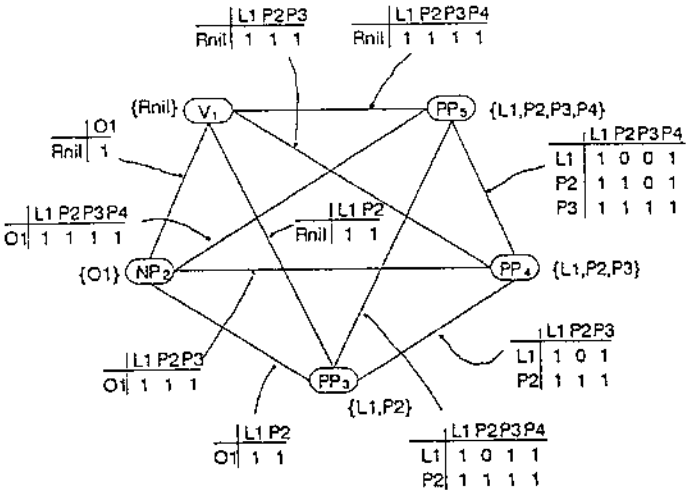

Figure 5: Initial constraint network (the values Rnil, L1, P2, ... should be read as <ROOT,nil>, <LOC,1>, <POSTMOD,2>, ..., and so on.)

(G2a-4) $word(pos(x))$=NP $\Rightarrow$ ($word(mod(x))$=V, $lab(x)$=OBJ, $mod(x) < pos(x)$)

"An NP modifies a V on the left with the label OBJ."

(G2a-5) $word(pos(x))$=V $\Rightarrow$ ($mod(x)$=nil, $lab(x)$=ROOT)

"A V modifies nothing with the label ROOT."

(G2a-6) $mod(x) < pos(y) < pos(x) \Rightarrow$ $mod(x) \leq mod(y) \leq pos(x)$

"Modification links do not cross each other."

According to the grammar $G2a$, sentence (5) has 14 (= $Catalan(4)$) different syntactic structures. We do not generate these syntactic structures one by one, since the number of the structures may grow more rapidly than exponentially when the sentence becomes long. Instead, we build a packed data structure, called a *constraint network*, that contains all the syntactic structures implicitly. Explicit parse trees can be generated whenever necessary, but it may take a more than exponential computation time.

Formation of initial network

Figure 5 shows the initial constraint network for sentence (5). A node in a constraint network corresponds to a role. Since each word has only one role *governor* in the grammar $G2$, the constraint network has five nodes corresponding to the five words in the

sentence. In the figure, the node labeled V_1 represents the governor role of the word V_1, and so on. A node is associated with a set of possible values that the role can take as its value, called a *domain*. The domains of the initial constraint network are computed by examining unary constraints ((G2a-1) to (G2a-5) in our example). For example, the modifiee of the role of the word V_1 must be ROOT and its label must be nil according to the unary constraint (G2a-5), and therefore the domain of the corresponding node is a singleton set {<ROOT,nil>}. In the figure, values are abbreviated by concatenating the initial letter of the label and the modifiee, such as Rnil for <ROOT,nil>, O1 for <OBJ,1>, and so on.

An arc in a constraint network represents a binary constraint imposed on two roles. Each arc is associated with a two-dimensional matrix called a *constraint matrix*, whose xy-elements are either 1 or 0. The rows and the columns correspond to the possible values of each of the two roles. The value 0 indicates that this particular combination of role values violates the binary constraints. A constraint matrix is calculated by generating every possible pair of values and by checking its validity according to the binary constraints. For example, the case in which *governor*(PP_3) = <LOC,1> and *governor*(PP_4) = <POSTMOD,2> violates the binary constraint (G2a-6), so the L1-P2 element of the constraint matrix between PP_3 and PP_4 is set to zero.

The reader should not confuse the undirected arcs in a constraint network with the directed modification links in a dependency diagram. An arc in a constraint network represents the existence of a binary constraint between two nodes, and has nothing to do with the modifier-modifiee relationships. The possible modification relationships are represented as the *modifiee* part of the domain values in a constraint network.

A constraint network contains all the information needed to produce the parsing results. No grammatical knowledge is necessary to recover parse trees from a constraint network. A simple backtrack search can generate the 14 parse trees of sentence (5) from the constraint network shown in Figure 5 at any time. Therefore, we regard a constraint network as a packed representation of parsing results.

Filtering

A constraint network is said to be *arc consistent* if, for any constraint matrix, there are no rows and no columns that contain only zeros. A node value corresponding to such a row or a column cannot participate in any solution, so it can be abandoned without further checking. The filtering algorithm identifies such inconsistent values and removes them from the domains. Removing a value from one domain may make another value in another domain inconsistent, so the process is propagated over the network until the network becomes arc consistent.

Filtering does not generate solutions, but may significantly reduce the search space. In our example, the constraint network shown in Figure 5 is already arc consistent, so nothing can be done by filtering at this point.

Adding New Constraints

To illustrate how we can add new constraints to narrow down the ambiguity, let us introduce additional constraints (G2b-1) and (G2b-2), assuming that appropriate syntactic and/or semantic features are attached to each word and that the function *fe(i)* is provided to access these features.

(G2b-1) $word(pos(x))$=PP, on_table $\in fe(pos(x)) \Rightarrow$
$\neg$(floor $\in fe(mod(x))$)

"A floor is not on a table."

(G2b-2) $lab(x)$=LOC, $lab(y)$=LOC, $mod(x)$=$mod(y)$,
$word(mod(x))$=V $\Rightarrow x$=y
"No verb can take two locatives."

Note that these constraints are not purely syntactic. Any kind of knowledge, syntactic, semantic, or even pragmatic, can be applied in CDG parsing as long as it is expressed as a unary or binary constraint on word-to-word modifications.

Each value or pair of values is tested against the newly added constraints. In the network in Figure 5, the value P3 (i.e. <POSTMOD,3>) of the node PP_4 (i.e., "on the table (PP_4)" modifies "on the floor (PP_3)") violates the constraint (G2b-1), so we remove P3 from the domain of PP_4. Accordingly, corresponding rows and columns in the four constraint matrices adjacent to the node PP_4 are removed. The binary constraint (G2b-2) affects the elements of the constraint matrices. For the matrix between the nodes PP_3 and

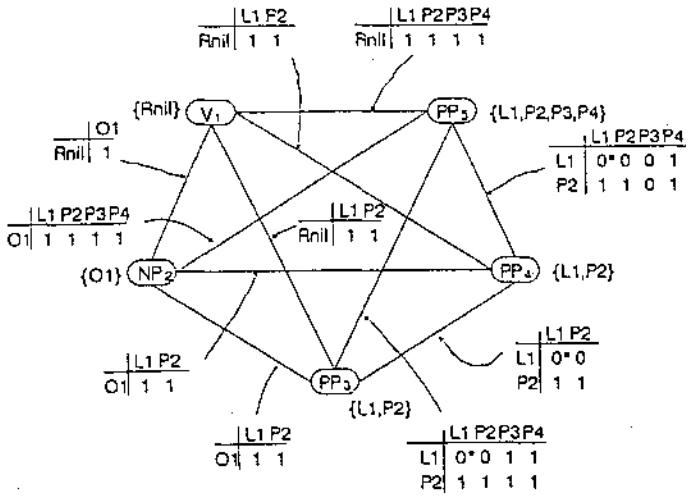

Figure 6: Modified network

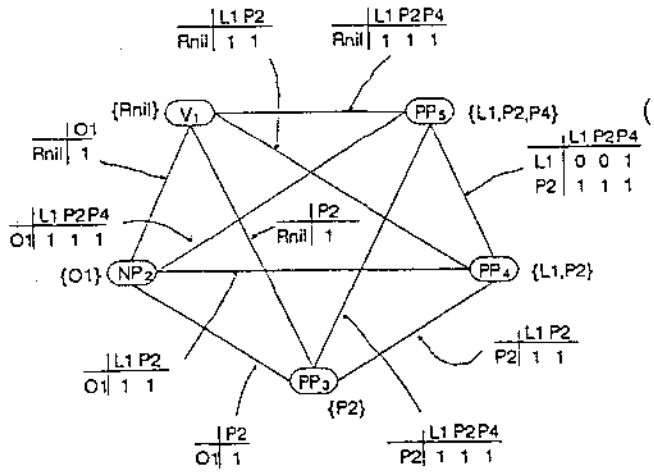

Figure 7: Filtered network

PP_4, the element in row L1 (<LOC,1>) and column L1 (<LOC,1>) is set to zero, since both are modifications to V_1 with the label LOC. Similarly, the L1-L1 elements of the matrices PP_3-PP_5 and PP_4-PP_5 are set to zero. The modified network is shown in Figure 6, where the updated elements are indicated by asterisks.

Note that the network in Figure 6 is not arc consistent. For example, the L1 row of the matrix PP_3-PP_4 consists of all zero elements. The filtering algorithm identifies such locally inconsistent values and eliminates them until there are no more inconsistent values left. The resultant network is shown in Figure 7. This network implicitly represents the remaining four parses of sentence (5).

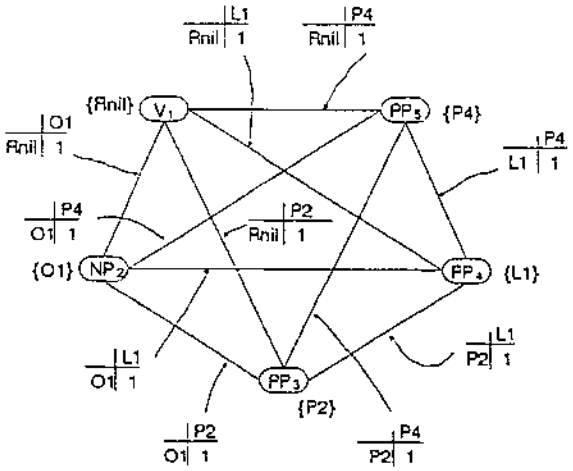

Figure 8: Unambiguous parsing result

Since the sentence is still ambiguous, let us consider another constraint.

(G2c-1) $lab(x)$=POSTMOD, $lab(y)$=POSTMOD, $mod(x)$=$mod(y)$, on $\in fe(pos(x))$, on $\in fe(pos(y)) \Rightarrow x$=$y$

"No object can be on two distinct objects."

This sets the P2-P2 element of the matrix PP_3-PP_4 to zero. Filtering on this network again results in the network shown in Figure 8, which is unambiguous, since every node has a singleton domain. Recovering the dependency structure (the one in Figure 4) from this network is straightforward.

Related Work

Several researchers have proposed variant data structures for representing a set of syntactic structures.

Chart (Kaplan 1973) and *shared, packed forest* (Tomita 1987) are packed data structures for context-free parsing. In these data structures, a substring that is recognized as a certain phrase is represented as a single edge or node regardless of how many different readings are possible for this phrase. Since the production rules are context free, it is unnecessary to check the internal structure of an edge when combining it with another edge to form a higher edge. However, this property is true only when the grammar is purely context-free. If one introduces context sensitivity by attaching augmentations and controlling the applicability of the production rules, different readings of the same string with

the same nonterminal symbol have to be represented by separate edges, and this may cause a combinatorial explosion.

Seo and Simmons (1988) propose a data structure called a *syntactic graph* as a packed representation of context-free parsing. A syntactic graph is similar to a constraint network in the sense that it is dependency-oriented (nodes are words) and that an *exclusion matrix* is used to represent the co-occurrence conditions between modification links. A syntactic graph is, however, built after context-free parsing and is therefore used to represent only context-free parse trees. The formal descriptive power of syntactic graphs is not known. As will be discussed in Section 4, the formal descriptive power of CDG is strictly greater than that of CFG and hence, a constraint network can represent non-context-free parse trees as well.

Sugimura et al. (1988) propose the use of a constraint logic program for analyzing modifier-modifiee relationships of Japanese. An arbitrary logical formula can be a constraint, and a constraint solver called CIL (Mukai 1985) is responsible for solving the constraints. The generative capacity and the computational complexity of this formalism are not clear.

The above-mentioned works seem to have concentrated on the efficient representation of the output of a parsing process, and lacked the formalization of a structural disambiguation process, that is, they did not specify what kind of knowledge can be used in what way for structural disambiguation. In CDG parsing, any knowledge is applicable to a constraint network as long as it can be expressed as a constraint between two modifications, and an efficient filtering algorithm effectively uses it to reduce structural ambiguities.

4 FORMAL PROPERTIES

Weak Generative Capacity of CDG

Consider the language $Lww = \{ww | w \in (a+b)^*\}$, the language of strings that are obtained by concatenating the same arbitrary string over an alphabet {a,b}. Lww is known to be non-context-free (Hopcroft and Ullman 1979), and is frequently mentioned when discussing the non-context-freeness of the "respectively" construct (e.g. "A, B, and C do D, E, and F, respectively") of various natural languages (e.g., Savitch et al. 1987). Although there

$\Sigma = \{a, b\}$
$L = \{l\}$
$R = \{partner\}$
C = conjunction of the following subformulas:

- $(word(pos(x))$=a $\Rightarrow$ $word(mod(x))$=a)
  & $(word(pos(x))$=b $\Rightarrow$ $word(mod(x))$=b)
- $mod(x) = pos(y) \Rightarrow mod(y) = pos(x)$
- $mod(x) \neq pos(x)$ & $mod(x) \neq$ nil
- $pos(x) < pos(y) < mod(y) \Rightarrow$
  $pos(x) < mod(x) < mod(y)$
- $mod(y) < pos(y) < pos(x) \Rightarrow$
  $mod(y) < mod(x) < pos(x)$

Figure 9: Definition of Gww

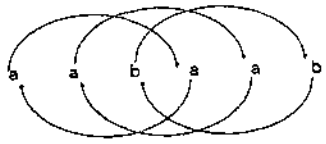

Figure 10: Assignment for a sentence of Lww

is no context-free grammar that generates Lww, the grammar $Gww =< \Sigma, L, R, C >$ shown in Figure 9 generates it (Maruyama 1990). An assignment given to a sentence "aabaab" is shown in Figure 10.

On the other hand, any context-free language can be generated by a degree=2 CDG. This can be proved by constructing a constraint dependency grammar G_{CDG} from an arbitrary context-free grammar G_{CFG} in Greibach Normal Form, and by showing that the two grammars generate exactly the same language. Since G_{CFG} is in Greibach Normal Form, it is easy to make one-to-one correspondence between a word in a sentence and a rule application in a phrase-structure tree. The details of the proof are given in Maruyama (1990). This, combined with the fact that Gww generates Lww, means that the weak generative capacity of CDG with degree=2 is strictly greater than that of CFG.

Computational complexity of CDG parsing

Let us consider a constraint dependency grammar $G =< \Sigma, R, L, C >$ with arity=2 and degree=k. Let n be the length of the input sentence. Consider the space complexity of the constraint network first. In

CDG parsing, every word has k roles, so there are $n \times k$ nodes in total. A role can have $n \times l$ possible values, where l is the size of L, so the maximum domain size is $n \times l$. Binary constraints may be imposed on arbitrary pairs of roles, and therefore the number of constraint matrices is at most proportional to $(nk)^2$. Since the size of a constraint matrix is $(nl)^2$, the total space complexity of the constraint network is $O(l^2k^2n^4)$. Since k and l are grammatical constants, it is $O(n^4)$ for the sentence length n.

As the initial formation of a constraint network takes a computation time proportional to the size of the constraint network, the time complexity of the initial formation of a constraint network is $O(n^4)$. The complexity of adding new constraints to a constraint network never exceeds the complexity of the initial formation of a constraint network, so it is also bounded by $O(n^4)$.

The most efficient filtering algorithm developed so far runs in $O(ea^2)$ time, where e is the number of arcs and a is the size of the domains in a constraint network (Mohr and Henderson 1986). Since the number of arcs is at most $O((nk)^2)$, filtering can be performed in $O((nk)^2(nl)^2)$, which is $O(n^4)$ without grammatical constants.

Thus, in CDG parsing with arity 2, both the initial formation of a constraint network and filtering are bounded in $O(n^4)$ time.

5 CONCLUSION

We have proposed a formal grammar that allows efficient structural disambiguation. Grammar rules are constraints on word-to-word modifications, and parsing is done by adding the constraints to a data structure called a constraint network. The initial formation of a constraint network and the filtering have a polynomial time bound whereas the weak generative capacity of CDG is strictly greater than that of CFG.

CDG is actually being used for an interactive Japanese parser of a Japanese-to-English machine translation system for a newspaper domain (Maruyama et. al. 1990). A parser for such a wide domain should make use of any kind of information available to the system, including user-supplied information. The parser treats this information as another set of unary constraints and applies it to the constraint network.

References

1. Church, K. and Patil, R. 1982, "Coping with syntactic ambiguity, or how to put the block in the box on the table," *American Journal of Computational Linguistics*, Vol. 8, No. 3-4.
2. Hays, D.E. 1964, "Dependency theory: a formalism and some observations," *Language*, Vol. 40.
3. Hopcroft, J.E. and Ullman, J.D., 1979, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley.
4. Kaplan, R.M. 1973, "A general syntactic processor," in: Rustin, R. (ed.) *Natural Language Processing*, Algorithmics Press.
5. Maruyama, H. 1990, "Constraint Dependency Grammar," *TRL Research Report* RT0044, IBM Research, Tokyo Research Laboratory.
6. Maruyama, H., Watanabe, H., and Ogino, S, 1990, "An interactive Japanese parser for machine translation," *COLING '90*, to appear.
7. Mohr, R. and Henderson, T. 1986, "Arc and path consistency revisited," *Artificial Intelligence*, Vol. 28.
8. Montanari, U. 1976, "Networks of constraints: Fundamental properties and applications to picture processing," *Information Science*, Vol. 7.
9. Mukai, K. 1985, "Unification over complex indeterminates in Prolog," ICOT Technical Report TR-113.
10. Savitch, W.J. et al. (eds.) 1987, *The Formal Complexity of Natural Language*, Reidel.
11. Seo, J. and Simmons, R. 1988, "Syntactic graphs: a representation for the union of all ambiguous parse trees," *Computational Linguistics*, Vol. 15, No. 7.
12. Sugimura, R., Miyoshi, H., and Mukai, K. 1988, "Constraint analysis on Japanese modification," in: Dahl, V. and Saint-Dizier, P. (eds.) *Natural Language Understanding and Logic Programming, II*, Elsevier.
13. Tomita, M. 1987, "An efficient augmented-context-free parsing algorithm," *Computational Linguistics*, Vol. 13.
14. Waltz, D.L. 1975, "Understanding line drawings of scenes with shadows," in: Winston, P.H. (ed.): *The Psychology of Computer Vision*, McGraw-Hill.