Vol. 35 No. 9 Transactions of Information Processing Society of Japan Sep. 1994

Regular Paper

Efficient Direct ID/LP Parsing with Generalized Discrimination Networks and Hasse Diagram

SURAPANT MEKNAVIN,† MANABU OKUMURA †† and HOZUMI TANAKA †††

We present a new parsing method using ID/LP rules directly without transforming them to context-free grammar rules. The method regards parsing as traversal of the generalized discrimination networks and represents the parsing states as nodes in the networks. This can yield more compact representation of the parser's state sets compared with previous methods and hence more wasteful computations can be avoided. We also optimize LP rules checking so that it can be checked efficiently. Our parsing strategy is a variant of the chart method which is customized to match ID/LP rules. Using this strategy, a large amount of overhead in processing can be omitted. Comparisons of our method with other related works are also described.

1. Introduction

Variations of word order are among the most well-known phenomena of natural languages. From a well represented sample of world languages, Steel[16] shows that about 76% of languages exhibit significant word order variation. In addition to the well-known Walpiri (Australian language), several languages such as Japanese, Thai, German, Hindi, and Finnish also allow considerable word order variations. It is widely admitted that such variations are governed by generalizations that should be expressed by the grammars.

Generalized Phrase Structure Grammars (GPSG)[8] provide a method to account for these generalizations by decomposing the grammar rules into Immediate Dominance (ID) rules and Linear Precedence (LP) rules. Using the ID/LP formalism, flexible word order languages can be more easily and consisely described. However, like other highly modular frameworks, designing an efficient algorithm to put the various components back in parsing is a difficult problem. There have been many efforts to develop efficient implementations of ID/LP parsers. Given a set of ID/LP rules, one alternative for parsing is to compile it into another grammar description language, e.g. Context-Free Grammars (CFG), for which there exist parsing algorithms. However, this approach lacks of the naturalness, especially for flexible word order language, in the sense that they do not take the language generalizations in the grammars into account while parsing. Also, this loses the modularity of ID/LP formalism. Moreover, the received object grammar tends to be so huge that the parsing time can increase dramatically.

Another set of approaches[14),2),5)] tries to parse directly by using ID/LP rules as they are without transforming to other formalisms. Shieber[14)] proposed an interesting algorithm of direct ID/LP parsing by generalizing Earley's algorithm[7)] to use the constraints of ID/LP rules directly. Barton[4)] showed that Shieber's direct parsing algorithm usually does have a time advantage over the use of Earley's algorithm on the expanded CFG. Thus the direct parsing strategy appeals to be an interesting candidate for parsing with ID/LP rules from the computational and natural viewpoints.

In fact, the reason that Shieber's algorithm wins over parsing on expanded CFG is mainly by virtue of the multiset representation of the states that can reduce the number of intermediate states drastically in average cases. However, Shieber's algorithm may still suffer from the

† National Electronics and Computer Technology Center, Gypsum Metropolitan Building Office, 539/2, Sri-ayuthaya Rd., Ratchathewi, Bangkok 10400, Thailand
†† Department of Computer Science, Japan Advanced Institute of Science and Technology, 15, Asahidai, Tatsunokuchi-cho, Noumi-gun, Ishikawa-ken, 923-12, Japan
††† Department of Computer Science, Tokyo Institute of Technology, 2-12-1, O-okayama, Meguro-ku, Tokyo 152, Japan

combinatorial explosion of the number of intermediate states while parsing. Although this cannot be avoided because ID/LP parsing is proved to be NP-complete[4] and thus can blow up in the worst case, it does not preclude us from algorithms with better average performance.

In this paper, we present a new approach for direct ID/LP rules parsing that outperforms the previous methods. Three features contribute to its efficiency. First, ID rules are precompiled to generalized discrimination networks[11] to yield compact representation of parsing states, hence less computation time. Second, LP rules are also precompiled into a Hasse diagram to minimize the time used for the order legality check at run time. And, third, its parsing strategy is based on the Word Incorporation (WI) algorithm[15] which is a specialization of the chart algorithm with less edge processing required.

2. ID/LP Grammar Formalism

One of the most explicitly formulated and well-known theories of linear precedence in natural language is the Immediate Dominance-Linear Precedence (ID/LP) theory first proposed by Gazdar and Pullum, and slightly modifed in Ref. 8). They observed that the standard formalism of Context-free Phrase Structure Grammars (CFPSG) fail to express generalizations about linear order that natural languages appear to exhibit. They propose to account for the existence of LP generalizations by changing the form in which grammatical rules are written into two sets of rules: Immediate Dominance (ID) rules and Linear Precedence (LP) rules. The former simply state what constituents may appear as the daughters within a given constituent, while the latter specify constraints on the order in which those daughters may appear in constituent structures admitted by any rule. Consider the following example grammar G_0 that has only one ID rule and one LP rule:

G_0:

$S \rightarrow_{ID} a, b, c, d$

$a < c$

The ID rule says that categories a, b, c and d can be immediately dominanced by category S. However, it does not say anything about the linear order in which a, b, c and d must occur under S. Instead, the LP rules impose the constraint using an antisymmetric, transitive relation $<$. In this example, it means that a must precede c in the local tree induced by the ID rule above.☆

To use ID/LP rules in parsing, one alternative is to compile ID/LP rules into an object grammar with a more familiar format like CFG. The object grammar can then be parsed using any algorithms already existing for that kind of grammar formalism. However, the size of object grammar may grow exponentially after the expansion and can actually dominate complexity for a relevant range of input lengths. To get some idea of this, consider what happens if we compile the above grammar G_0 into CFG. The corresponding CFG G_0' will have $4!/2=12$ rules spelling out all possible strings formed by a, b, c and d, whose a is prior to c.

G_0':

$S \rightarrow a, b, c, d$	$S \rightarrow a, b, d, c$
$S \rightarrow a, c, b, d$	$S \rightarrow a, c, d, b$
$S \rightarrow a, d, b, c$	$S \rightarrow a, d, c, b$
$S \rightarrow b, a, c, d$	$S \rightarrow b, a, d, c$
$S \rightarrow b, d, a, c$	$S \rightarrow d, b, a, c$
$S \rightarrow d, a, b, c$	$S \rightarrow d, c, a, b$

Given the disadvantage of the above approach, direct parsing on ID/LP grammars seems to be a more attractive method. Shieber shows how to modify Earley's algorithm for parsing CFG to use the constraints of ID/LP grammars directly, without the combinatorially explosive step of grammar conversion. Dotted rules in the original one are replaced by so called dotted UCFG (Unordered Context-Free Grammar) rules which represent the elements after the dot with *multiset* instead of an ordered sequence of them. This can be an advantage because instead of expanding all surface appearances ahead of time, Shieber's algorithm works out the possibilities one step at a time, as needed. For example, if Shieber's algorithm is used to parse the string *abcd* according to G_0, the state set of the parser after a is entered contains only single one state:

$[S \rightarrow a \cdot \{b, c, d\}, 0]$

In contrast, using original Earley's algorithm to parse the same string on the expanded CFG generates 6 states for all possible orders of the remaining symbols:

☆ More precisely, $x < y$ means y cannot appear to the left of x in a local tree.

$[S \rightarrow a \cdot bcd, 0]$
$[S \rightarrow a \cdot bdc, 0]$
$[S \rightarrow a \cdot cbd, 0]$
$[S \rightarrow a \cdot cdb, 0]$
$[S \rightarrow a \cdot dbc, 0]$
$[S \rightarrow a \cdot dcb, 0]$

As seen, the representation of the elements in a rule as the multiset helps a lot in reducing the number of states by keeping many possible orders of unseen symbols unexpanded in the multiset. If the cost of LP rules checking can be neglected, the cost of Shieber's algorithm will then depend on the number of the states in the state sets and Shieber's parser should thus be faster. As noted by Barton,[4] despite its potential of blowing up due to the inherent difficulty of ID/LP parsing, Shieber's direct parsing algorithm wins out over the use of an expanded object grammar that blows up unnecessarily.

The multiset representation in Shieber's algorithm saves a lot by keeping many possibilities unexpanded in a dotted UCFG rule. In next section, we present the method that can further save more with its compact rule representation as generalized discrimination network.

3. ID Rules as Generalized Discrimination Networks

In this section, we will describe how ID rules are treated in our approach. At first, we will describe about the discrimination network and how it can be related with ID rules. We also discuss the merit and problem of using the discrimination network. Then we go on describing how the problems are solved by using the generalized discrimination network.

3.1 Discrimination Network's Merit and Problem

Given a set of constraints defining a concept, one can build a corresponding discrimination network used to check that concept efficiently. Considering the ID rules with the same LHS (Left Hand Side) element as a set of constraints to construct that constituent structure, we can thus represent such a set as a discrimination network by viewing the existence of each element in RHS (Right Hand Side) of those ID rules as a discrimination constraint in the network. Figure 2 shows a discrimination network of the ID rules in the grammar G_1, shown in **Fig. 1**.

$$s \rightarrow_{ID} a, b, c, d \quad (1)$$
$$s \rightarrow_{ID} a, b, e, f \quad (2)$$
$$a, b, c < d \quad (3)$$
$$b < c \quad (4)$$
$$a, e < f \quad (5)$$

Fig. 1 An example ID/LP grammar: G_1.

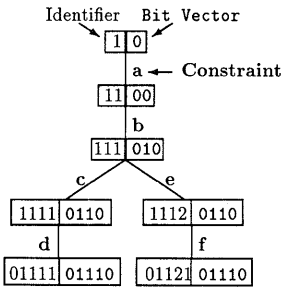


Fig. 2 Discrimination network representation of ID rules with identifier and bit vector assigned to each node.

1. For a RHS element denoted by italic letter, like a, we denote the constraint of its existence by its bold letter, like **a**. Representing such rules by a discrimination network has the merits that we can use the states in the network to track the progress of parsing and this can delay rules expansion as by using Shieber's data structure. However, since a discrimination network is constructed with the set of ID rules with the same LHS element, we can also compact RHS elements of the same form in different rules into one arc in the network. This yields more compact rules representation, especially when there are many rules for a category, and wasteful recomputation can be avoided. Shieber's representation, in contrast, considers each single ID rule separately and hence cannot capture this kind of compactness. For instance, consider the grammar G_1. Two ID rules in the grammar have a and b in common, so their corresponding constraints can be merged together into common arcs labeled **a** and **b** in the discrimination network, as shown in **Fig. 2**. If, for example, the first symbol of input is a, there will be two possible parses, corresponding to each ID rule. Using discrimination network, the two parses can be represented as a single node which is linked to the arc labeled **a**. On the other hand, using Shieber's representation, one must still represent the two parses separately as $[S \rightarrow a \cdot \{b,$

$c, d\}, 0]$ and $[S \rightarrow a \cdot \{b, e, f\}, 0]$.

Up to now, everything seems to go on nicely without any obstacle. However, traditional discrimination network has a problem in that it cannot be traversed unless constraints are entered (satisfied) in an a priori fixed order. For example, in this case, if the first symbol is the symbol other than a the traversal would be suspended. This is to say that the traditional discrimination network is rather suited for conventional grammar rules that fix the order of RHS elements, but may be in trouble with order-free characteristic of ID rules.

3.2 Generalized Discrimination Network

Okumura and Tanaka[11] propose the method of generalized discrimination networks (GDN) to solve the problem described above. GDN is a generalization of a discrimination network which can be traversed according to the order in which constraints are obtained incrementally during the analytical process, independently of order. The technique is to assign each node in the network a unique identifier and a bit vector, as illustrated in Fig. 2. The leftmost digit of an identifier of a node v indicates whether the node is a leaf or not, '0' for being a leaf and '1' for being a non-leaf. This digit is followed by the sequence $S(v)$, which is the concatenation of the sequence $S(u)$ and the integer k, where u is the immediate predecessor of v and arc u–v is the kth element in the ordered set of arcs issuing from u.☆ Note that the identifier of the root node r has only the first leftmost digit ($S(r)$ is null).

To each node identifier, a bit vector is attached. The bit vector has the same length as its associated identifier and consists of 1's in every bit except the leftmost and rightmost bits. For example, as shown in Fig. 2, identifiers 1, 11, 111, 1112 and 01121 are attached with the bit vectors 0, 00, 010, 0110 and 01110 respectively. The reason for this assignment will be explained shortly.

Next, constraint-identifier pairs are extracted from the network in the following form: a branch and the subordinate node which is directly connected by that branch, as shown in **Table 1**. This correspondence between constraint and identifier means that if a constraint is satisfied, the node of corresponding identifier can be reached in the network. For example, if constraint **b** is satisfied, the network can be traversed downward to the node of corresponding identifier 111.

Table 1 Correspondence between constraint and identifier.

constraint	identifier
a	11
b	111
c	1111
d	01111
e	1112
f	01121

The bit vector attached to each identifier is for representing the positions of the unsatisfied constraints in the network. A bit of 0 or 1 indicates that the corresponding constraint is satisfied or unsatisfied respectively. The leftmost bit has no corresponding constraint and exists just to make the vector length the same as that of the identifier. Other bits from left to right correspond to the constraints from the root node to the reached node respectively. The rightmost bit corresponds to the constraint of the branch directly linked to the reached node and is thus always 0 to represent the satisfaction of the constraint. (Otherwise, the node cannot be reached.) All the remaining '1' bits indicate that other constraints between the root node and the reached node are not yet satisfied.

For example, in the case above where the constraint **b** is satisfied, the reached node's identifier 111 has the corresponding bit vector 010. The satisfaction of constraint **b** is represented by the rightmost bit whose value is 0. However, to traverse from the root node to node 111, in addition to constraint **b**, constraint **a** must also be satisfied. This condition is expressed by the second bit from the left of the vector whose value is 1. That is to say, the reachability of node 111 is 'conditional' in that the node can be reached if constraint **a** is satisfied. Similarly, in the case where the constraint **f** is satisfied, according to Fig. 3, the reached node identifier is 01121. Its attached bit vector 01110 signifies that, to reach the node constraints **a**, **b** and **e** (which correspond to the second, third and fourth bit from the left) are left to be satisfied. The rightmost '0' bit represents the satisfaction

☆ The encoding used here is a little different from the original one in Ref. 11).

of constraint **f**.

The discrimination process, viewed as traversal of the network incrementally downward to the leaf nodes, can then be represented by a 'state' which is defined as an ordered-pair of node's identifier and its associated bit vector. The initial state is at the root node (where no constraints are obtained), thus it has '1' as its identifier and '0' as its bit vector and denoted by $\langle 1, 0 \rangle$. When a constraint is entered, constraint-identifier table is used to find the identifiers of all possible nodes which can be reached if the constraint is satisfied. These identifiers together with their bit vectors form a set of states. By applying the operation between the current state and these states, we can find all possible states that are reachable from the current state. An operation between two states consists of the following operations :

operation between identifiers : Ignoring the leftmost bit, if one identifier includes the other as a prefix-numerical string, return the longer string. Otherwise, fail ;

operation between bit vectors : After adjusting the length of bit vectors by attaching 1's to the end of the shorter vector, return the bit vector for which each bit is a conjunction of the bits of two bit vectors.

The operation between identifiers checks whether one node can be reached from the other in the network. From a node A, only identifiers of reachable nodes below can include the identifier of A as a prefix string. For example, consider Fig. 2. From node 1112, node 01121 is reachable by satisfying constraint **f**, but it is impossible to reach node 01111 on any account. If one node is reachable from the other, the identifier of the subordinate node is returned.

The operation between bit vectors allows us to cope with the free order of constraints. As explained above, the bit vector represents all the constraints that must be satisfied between the root node and the reached node. By taking the conjunction of bits of these vectors, which represent the satisfied constraint as 0, bits of the resultant vector are incrementally changed to 0. If all vector bits are 0, it means that all constraints are satisfied and the network can be traversed to the reached node unconditionally. Because the bit conjunction operation is executable in any order, it is possible to cope with an arbitrary order of constraints.

As an example, let us consider the traversal where constraints are obtained in the order of **b**, **e**, **a**, **f**. After constraint **b** is obtained, the next state is computed by the operation between the initial state $\langle 1, 0 \rangle$ and the state corresponding to constraint **b** $\langle 111, 010 \rangle$. Ignoring the leftmost bit, as null string is a prefix of any string, the operation between identifiers 1 and 111 succeeds and return 111. The operation between bit vectors 0 and 010 is performed by attaching 1's to the end of 0 to yield the bit vector 011, and taking the bit conjunction between 011 and 010. This yields the bit vector 010. As the result, the next state after **b** is obtained becomes the state $\langle 111, 010 \rangle$.

Computing in the same way, we will get the state $\langle 1112, 0100 \rangle$ after constraint **e** is entered, the state $\langle 1112, 0000 \rangle$ after constraint **a** is entered and the state $\langle 01121, 00000 \rangle$ after constraint **f** is entered. Note that because the last state is at a leaf node and all vector bits are 0, the discrimination process is complete with all constraints satisfied.

Using this technique, we need not check the constraints in the order as in the network because we can reach any node even though there must be some not yet satisfied constraints between it and the root node by just representing those constraints by '1' bits in its bit vector. Thus we can cope with an arbitrary order of entered constraints. Provided with this characteristic, we can represent the ID rules as the GDNs and the parsing process can then be naturally considered as a traversal of the GDN incrementally downward to the leaf nodes.

Note that the compilation of ID rules into GDN format does not suffer from the explosion of grammar size as in the case of the compilation to CFG. A GDN can be viewed as a set of optimized ID rules whose common parts are shared. Its size, therefore, is kept small in the same order as that of the original ID rules.

4. LP Rules as a Hasse Diagram

If a natural language has completely free word order, we may simply use the GDN to represent the language's ID rules and parse the language easily. Unfortunately, in reality, a language usually has some constraints on the word order, as expressed by its LP rules. Therefore, to parse

such a language, we have to take these constraints also into account. In this section, we introduce the organization that can cope with the linear precedence constraints of the LP rules. The method is to code every relevant item by distinct identiers and use them to check legality. By embedding the precedence relation between the items in the identifiers, we can check the legality of a sequence of items efficiently.

4.1 Constructing the Diagram for a Set of LP Constraints

Given a set of LP constraints, we can build the corresponding so called 'Hasse diagram' to represent the precedence relations between the relevant symbols. Hasse diagram is a representation of a partially ordered set used in graph theory.[10] The process of constructing a Hasse diagram from the given LP rules is explained below.

First, build a directed graph representing the LP relations where a node represents a symbol and an arc between two nodes represents the precedence between them, where the direction of an arc goes from the lower precedence node to the higher one. For the LP rules in G_1, we get the directed graph in **Fig. 3**.

Next, we can simplify this graph into a Hasse diagram by deleting unnecessary arcs and omitting the arrowheads representing direction. In Fig. 3, we can delete arc b-d because we know that "$<$" is a transitive relation, and consequently, the existence of arcs b-c and c-d is enough to imply the arc b-d. In addition, because we know that the direction of the arrowheads between nodes always go upward, all arrowheads can also be omitted. **Figure 4** shows the Hasse diagram for the graph in Fig. 3.

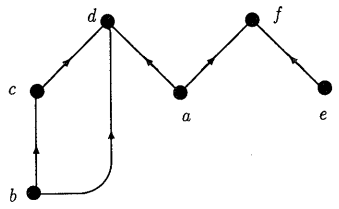


Fig. 3 The directed graph representing the precedence between items.

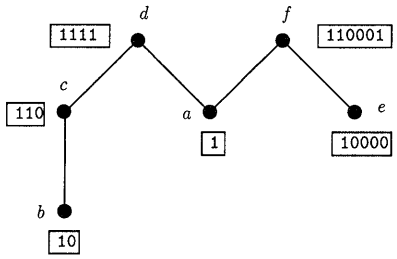


Fig. 4 Hasse diagram with the identifier assigned to each node.

4.2 Coding the Precedence Vectors

After building the Hasse diagram for a given set of LP constraints, we assign a precedence vector to each node by the following algorithm.

Algorithm : AssignPrecedenceVector

Let A be the set of all nodes in a Hasse diagram.

1. Assign a unique bit as the flag of each node in A.
2. Compute the precedence vector of each node by setting the flags of the node and all its subordinate nodes to 1, and setting others to 0.

As for this example, we assign (a)'s flag the first bit, (b)'s flag the second bit,$\cdots$, and (f)'s flag the sixth bit. As shown in Fig. 4, because node a, b and e are minimal elements which have no subordinate node, their precedence vectors are set to 1 only at their own flags, as 000001, 000010 and 010000 respectively. To calculate node c's precedence vector, we set its flag (third bit) and its subordinate node b's flag (second bit) to 1, resulting in a precedence vector 000110. The precedence vectors of d and f can also be calculated in the same way. The resultant precedence vectors are shown in Fig. 4 with 0's in their left parts omitted.

4.3 Order Legality Checking

Now that each node has its corresponding precedence vector, we can check the order legality of the corresponding symbols easily by the algorithm below.

Algorithm : CheckOrder

Input : Precedence vectors Pre_A and Pre_B of symbols A and B, where A precedes B in the input.

1. Take the bitwise disjunction between Pre_A and Pre_B.
2. Check equality : if the result in 1. is equal to Pre_A, reject. Otherwise, accept and return the result as the precedence vector of the concatenated string AB.

For example, using the LP rules of G_1, we can easily check that the sequence b, e, a, f is legal,

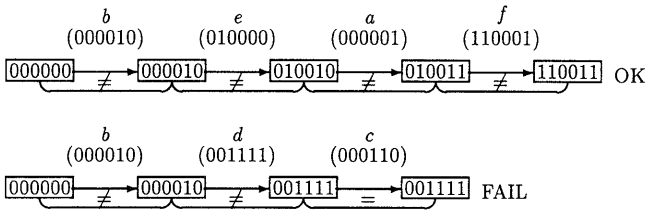


Fig. 5 The processes checking the order legality.

while the sequence like b, d, c, a is not. The check processes are shown in **Fig. 5**, where the precedence vector of each symbol is shown in a bracket and the current precedence vector of the string is shown in a block.

Note that, in the encoding algorithm described above, the precedence vector of a symbol A that must precede a symbol B is always included in B's precedence vector. As a result, if A comes behind B the disjunction of their precedence vectors will be equal to B's precedence vector. The above algorithm thus employs this fact to detect the order violation easily. Moreover, note that all LP constraints concerning the symbols concatenated are propagated with the resultant string's precedence vector by the result of disjunction. We can then use the algorithm to easily check the legality of next input symbol respective to all preceding symbols by checking only with the resultant string's precedence vector. In real implementations, a word can be used to represent a precedence vector and the order legality can thus be checked efficiently, in constant time, by using Boolean bitwise operations between words provided in most machines.

5. Parsing

5.1 Table for ID/LP Parsing

GDN can cope with any order of input constraints by referring to the table of constraint-identifier which is extracted from the network by collecting pairs of branches and their immediate subordinate nodes. But, as described, GDN is first proposed to handle the *completely* order-free constraint system. So, in order to apply the model to parse natural language of which word order is restricted by some linear precedence constraints, some modifications have to be done to take those constraints into account.

First, the definition of a state is changed from a 2-tuple $\langle Id, BitV\rangle$ to a 4-tuple $\langle Cat, Id, Pre, BitV\rangle$ where each element is defined as the following:

reduce(a, (S, 11, 1, 00)).
reduce(b, (S, 111, 10, 010)).
reduce(c, (S, 1111, 110, 0110)).
reduce(d, (S, 01111, 1111, 01110)).
reduce(e, (S, 1112, 10000, 0110)).
reduce(f, (S, 01121, 110001, 01110)).

Fig. 6 Category-state table generated from ID/LP rules: G_1.

Cat: the mother category of the state;
Id: the identifier of the state;
Pre: the precedence vector of the state;
BitV: the bit vector of the state.

Because we have many networks for all nonterminal categories in a grammar, *Cat* is added to indicate which network the state belongs to. Moreover, in addition to the elements for ID rules checking, the precedence vector *Pre* is added for the sake of LP rules checking.

Next, the constraint-identifier table is replaced by the category-state table, notated as *reduce* (*category*, *state*), viewing each category as a constraint and using the new definition of a state. This table will be used to reduce a constituent to a higher level constituent state when it is complete. A constituent is complete if its current state is at a leaf node and all bits of *BitV* are set to 0. **Figure 6** shows the table derived from G_1.

5.2 The Parsing Algorithm

Our parsing strategy is based on the Word Incorporation (WI) algorithm with some modifications to accommodate ID/LP formalism. The WI algorithm can be viewed as a specialization of the chart method. Traditional chart parsing[19] allows any arbitrary strategy of selecting the next pending edge to combine with the chart. The entered edge then may span any vertices and create the edges already in the chart. Since this can cause looping, the newly created

edges have to be checked with the entire chart to filter out the duplicated ones. The WI algorithm, in contrast, is restricted to be solely bottom-up and depth-first. This makes the parsing proceed along the input in an ordely fashion (say, left to right) and keep running at a vertex until no more new edge ending at the vertex can be generated. Once the parsing goes beyond a vertex, the processing will never be redone at that vertex again. As a consequence, the edge check can be omitted and thus the parsing time can be reduced significantly. Our algorithm can be described as follows:

Algorithm: Parse

Given a category-state table T generated from ID/LP grammar G, a dictionary D, a goal category S and an input string $s=w_1w_2\cdots w_n$, where w_i is a terminal in G, we construct chart as follows:

$k\leftarrow 0$;

while $k<n$ **do begin**

1. Loop up D for the entry of w_{k+1}. Span the inactive (complete) edges corresponding to every possible categories of w_{k+1} between vertices k and $k+1$.
   Now perform steps (2) and (3) until no new edges can be added.
2. For each inactive edge of category β spanned between vertices j and $k+1$ $(j<k+1)$, if *reduce*(β, ϕ) is an entry in T, span the edge ϕ between vertices j and $k+1$.
3. For each active (incomplete) edge of category β spanned between vertices j and $k+1$, search for active edge spanned between vertices i and j $(i<j)$. For each one found, perform the check operation between the two edges. If this succeeds, add the resultant new edge between vertices i and $k+1$.
4. $k\leftarrow k+1$

end;

The string is accepted if and only if there exists some complete edges of category S spanned between vertices 0 and n in the chart.

Here, the check operation between two edges (states) includes all of the following operations:

operation between *Cat*s: If *Cat*s are the same then return *Cat*. Otherwise, fail;

operation between *Id*s: As defined in 3.2;

operation between *Pre*s: As described in CheckOrder algorithm;

operation between *BitV*s: As defined in 3.2.

The operation between *Cat*s first checks whether the two states are in the same network. The operation between *Id*s then checks whether one node can be reached from the other in the network. The operation between *Pre*s tests whether the catenation of the edges violates LP constraints and returns the precedence vector of the successfully combinded edge as described in section 4. The operation between *BitV*s allows us to cope with the free order of constraints.

Example. Suppose we are given the string of words to parse whose categories are b, e, a, f, using grammar in Fig. 1. First, the edge $\langle S, 111, 10, 010\rangle$ is spanned between vertices 0 and 1, since the first element of the string is a b. No more iterations of step 2 and 3 are possible, so we move on to the next word. After category e is obtained, its corresponding state $\langle S, 1112, 10000, 0110\rangle$ is then operated with the state $\langle S, 111, 10, 010\rangle$. The operation between categories succeeds because both states have the same category S. The operation between identifiers 111 and 1112 succeeds because 111 is a prefix of 1112, thus 1112 is returned. The operation between precedence values 10 and 10000 also succeeds because the bitwise disjunction of them yields 10010 which is not equal to 10. Last, the operation between bit vectors 010 and 0110 returns the result of conjunction between 0100 and 0110 which is thus 0100. So the above operations yield the resultant state $\langle S, 1112, 10010, 0100\rangle$ as the edge spanned between vertices 0 and 2.

Continuing in this manner, we will get $\langle S, 1112, 10011, 0000\rangle$ between vertices 0 and 3, and $\langle S, 11121, 110011, 00000\rangle$ between vertices 0 and 4. Because the latter is a complete edge of

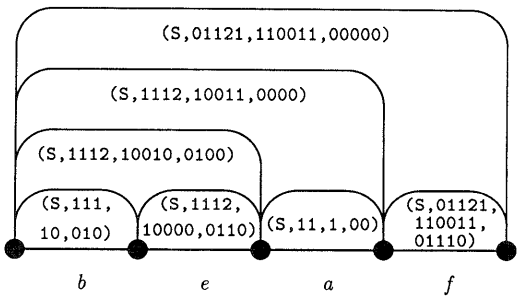


Fig. 7 Chart of parsing *beaf*.

goal category 'S', the input string is thus accepted. The chart is shown in **Fig. 7**.

6. Comparison with Related Works

To compare our method with Shieber's, both methods have the same worst-case space complexity: $O(2^{|G|})$, where $|G|$ denotes the size of the grammar G. This combinatorial explosion of data structure is due to the inherent difficulty of ID/LP parsing. However, our method tends to perform better in general case, as it needs less bookkeeping of intermediate states and checks LP rules more efficiently.

Other than Shieber's work, there are many works in the past concerning ID/LP parsing.[2),3),5),9),13)] Popewich's FIGG[13)] treats ID/LP rules by compiling them into Discontinuous Grammar rules which, although elegant, do not have a very efficient implementation. The different approach of top-down ID/LP parsing using logic grammars is presented by Abramson.[2)] The approach relies in the use of metarules.[1)] This approach is attractive in that it can be simply added on top of logic grammars that are directly available in Prolog. However, the main drawback in using top down recursive descent parsing methods is that it might result in an infinite loop for left recursive grammars. The recent version using Static Discontinuity Grammars (SDG)[6)] augmented with Abramson's metarules can solve this problem by adding loop control as a constraint on parsing. According to the comparison tests reported in Ref. 3), the approach appears to be considerably faster than Popowich's one.

7. Experiments

7.1 Experiment I

As an investigation of our approach, we have implemented a small parser, called GHW, using SICStus prolog on a NEWS3860 workstation. GHW is compared with the SDG+metarules parser, Shieber's parser and LangLAB's parser[17)] running on the same environments. LangLAB is a natural language analysis system written in prolog. We test LangLAB to compare the use of ID/LP rules and the use of expanded CFG rules in parsing. In experimentation, we use grammar X and the set of sentences from Ref. 3) that was used to compare SDG+metarules approach with FIGG and Evans and Gazdar's approach. The grammar is a small grammar containing 8 ID rules, 4 LP rules and 3 lexicon rules (see Appendix). For LangLAB, the grammar is expanded to the equivalent 19 DCG rules. All tested sentences of length two to six are generated in every combination from the grammar. The timings are averaged over 100 runs using compiled bytecode and reflect the average amount of CPU time in milliseconds required to parse the sentences of several lengths. The result is shown in Table 2. Because Shieber's and our parser develop all parses in parallel and thus the time used to find the 1st parse and the time used to find all parses are about the same, only the all parses time is shown.

Comparing GHW with Shieber's parser, GHW outperforms the other parser for all lengths of input. When comparing with SDG+metarules parser, SDG+metarules wins over our approach in finding the 1st parse for short sentences, but for longer sentences our approach surpasses the other in all cases. This is because our method needs to do more initialization at the beginning of parsing and thus for short sentences this cost will affect parse time significantly. However, in the case of longer sentences the cost will be small compared to over-all costs and can be neglected. So our method should be more suited to use in real applications with relatively long and complicated sentences. GHW also performs better than

Table 2 The result of the comprison test using grammar X.

Length of sentences		$n=2$	$n=3$	$n=4$	$n=5$	$n=6$
LangLAB	1st	9.	9.	30.	21.	35.
	total	18.	18.	39.	30.	49.
SDG+metarules	1st	1.1	3.4	6.7	16.7	101.3
	total	25.	49.8	89.	76.	267.
Shieber	total	65.	123.	160.	157.	321.
GHW	total	3.5	6.6	11.7	12.9	21.0

Table 3 Results of experiment II.

sentence length	trees	LangLAB first (msec)	LangLAB all (msec)	Shieber all (msec)	GHW all (msec)
9	2	299	942	3689	79
14	4	559	1037	4680	120
17	8	1414	2804	8449	289
18	4	1629	2366	8629	270
18	27	830	3136	10820	1429
19	3	1860	2884	11279	329
19	12	440	1288	5410	359
25	16	4195	9067	17389	859
28	1600	1539	126129	71820	8380

LangLAB in every case, though LangLAB seems to outperform the other two ID/LP parsers. This is because the size of expanded CFG grammar is not much larger than that of grammar X and thus the advantage of using ID/LP rules is obscured by parsing overhead in the two parsers.

7.2 Experiment II

To experiment with a more realistic grammar, we constructed a Thai grammar "Thai_1" consisting of 97 ID rules and 38 LP rules, which is equivalent to 556 CFG rules after translation. The grammar is an extension of ID/LP rules written by Vorasucha.[18] It covers almost all patterns of sentences appeared in Ref. 12). We use the grammar to test the performance of GHW, Shieber's parser and SDG+metarules parser on sentences with various lengths and patterns. For LangLAB, we use CFG grammar generated from Thai_1 testing on the same set of sentences. All tests run on NEWS3860 workstation. However, we found during the experimentation that the SDG+metarules method requires so much time to analyze the test sentences, so we stopped testing with it and show here in **Table 3** only the results of the other three parsers.

The figure shows that our parser GHW is better than the other two parsers even for a quite large grammar like Thai_1. LangLAB does quite well on rather short sentences but tends to get worse dramatically when sentences are long and have high ambiguities. Shieber's parser is slowest on processing short sentences but eventually surpasses LangLAB on long sentences. GHW is the best in finding all parses of every case and is about 10 times faster than the other two parsers in average.

8. Conclusion

A new method for parsing with ID/LP rules is described. The method improves the performance of parsing by keeping the parsing states set as small as possible, reducing the time used by the LP rules checking operation and cutting away the overhead of duplicated edge checking. These are accomplished by integrating the merits of GDN, Hasse diagram and WI algorithm in parsing. The method is shown to be superior to the previous methods on tested grammars. However, we realize that more explorations have to be done with diverse grammars and sentences to confirm our idea. This is left as one of our further works.

Acknowledgements The authors would like to thank Prof. Harvey Abramson for providing his system and Craig Hunter for checking English in this paper.

References

1) Abramson, H.: Metarules and an Approach to Conjunction in Definite Clause Translation Grammars, *Proceedings of the Fifth Intenational Conference and Symposium on Logic Programming* (Kowalski, R. and Bowen, K. eds.), MIT Press (1988).
2) Abramson, H.: Metarules for Efficient Top-down ID-LP Parsing in Logic Grammars, Technical Report Technical Report TR-89-11, University of Bristol, Department of Computer Science (1989).
3) Abramson, H. and Dahl, V.: On Top-down ID-LP Parsing with Logic Grammars, submitted for publication (1992).
4) Barton, E.: On the Complexity of ID/LP Parsing, *Computational Linguistics,* October-Decem-

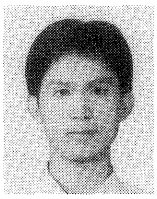
ber, pp. 205-218 (1985).
5) Blache, P. and Morin, J.: Bottom-up Filtering: a Parsing Strategy for GPSG, *Coling-90,* Vol. 2, pp. 19-23 (1990).
6) Dahl, V. and Popowich, F.: Parsing and Generation with Static Discontinuity Grammars, *New Generation Computing,* Vol. 8, No. 3, pp. 245-274 (1990).
7) Earley, J.: An Efficient Context-Free Parsing Algorithm, *Comm. ACM,* Vol. 13, No. 2, pp. 94-102 (1970).
8) Gazdar, G., Klein, E., Pullum, S. and Sag, I.: *Generalized Phrase Structure Grammar,* Blackwell Publishing (1985).
9) Kilbury, J.: Parsing with Category Cooccurence Restrictions, *Coling-88,* Vol. 1, pp. 324-327 (1988).
10) Liu, L., C.: *Elements of Discrete Mathematics,* McGraw-Hill International Editions (1986).
11) Okumura, M. and Tanaka, H.: Towards Incremental Disambiguation with a Generalized Discrimination Network, *Proceedings Eighth National Conference on Artificial Intelligence,* pp. 990-995 (1990).
12) Panupong, V.: *The Structure of Thai: Grammatical System,* Ramkamhaeng University (1984) (in Thai).
13) Popowich, F.: Unrestricted gapping grammars, *Computational Intelligence,* Vol. 2, pp. 28-53 (1986).
14) Shieber, S. M.: Direct Parsing of ID/LP Grammars, *Linguistics and Philosophy,* Vol. 7, pp. 135-154 (1984).
15) Simpkins, N. and Hancox, P.: Chart Parsing in Prolog, *New Generation Computing,* Vol. 8, No. 2, pp. 113-138 (1990).
16) Steel, S.: Word Order Variation: A Typological Study, *Universals of Language* (Greenberg, J. ed.), Vol. 4, Stanford University Press (1981).
17) Tokunaga, T., Iwayama, M., Tanaka, H. and Kamiwaki, T.: LangLAB: a Natural Language Analysis System, *Coling-88,* pp. 655-660 (1988).
18) Vorasucha, V.: Thai Syntax Analysis Based on GPSG, Master's thesis, Tokyo Institute of Technology (1986).
19) Winograd, T.: *Language as a Cognitive Process,* Vol. 1, Syntax, Addison-Wesley (1983).

Appendix

The grammar used in comparison test

$s \rightarrow_{ID} np, vp$	$n \rightarrow [n]$
$vp \rightarrow_{ID} v$	$v \rightarrow [v]$
$vp \rightarrow_{ID} v, np$	$p \rightarrow [p]$
$vp \rightarrow_{ID} v, np, pp$	$np < vp$
$vp \rightarrow_{ID} v, s$	$p < vp$
$vp \rightarrow_{ID} v, np, s$	$p < np$
$np \rightarrow_{ID} n$	$v < s$
$pp \rightarrow_{ID} p, np$	

(Received November 21, 1991)
(Accepted April 21, 1994)



Surapant Meknavin was born on September 21, 1966. He received the B. S. degree in Computer Engineering from the Faculty of Engineering of Chulalongkorn University in 1987. After that, he went to Tokyo Institute of Technology and received his M. S. degree and Doctor of Engineering in 1990 and 1993 respectively. Now he works as a researcher for the National Electronics and Computer Technology Center, Thailand. His research interests are in natural language processing, machine learning, and knowledge science.

Manabu Okumura was born in 1962. He received B. E., M. E. and Dr. Eng. from Tokyo Institute of Technology in 1984, 1986 and 1989 respectively. He has been an assistant at the Department of Computer Science, Tokyo Institute of Technology from 1989 to 1992. He is currently an associate professor at the School of Information Science, Japan Advanced Institute of Science and Technology. His current research interests include natural language understanding and knowledge representation. He is a member of the Information Processing Society of Japan

Hozumi Tanaka received the B. S., and M. S. degrees in faculty of science and engineering from Tokyo Institute of Technology in 1964 and 1966 respectively. In 1966 he joined in the Electro Technical Laboratories, Tsukuba. He received his Doctor of Engineering in 1980. In 1983 he joined as an associate professor in the faculty of Information Engineering in Tokyo Institute of Technology and he became professor in 1986. He has been engaged in Artificial Intelligence and Natural Language Processing research. He is member of the Information Processing Society of Japan, etc.