A Chart-based Method of ID/LP Parsing with Generalized Discrimination Networks

Surapant Meknavin Manabu Okumura Hozumi Tanaka

Department of Computer Science,
Tokyo Institute of Technology
2-12-1, O-okayama, Meguro-ku, Tokyo 152, Japan
e-mail surapan@cs.titech.ac.jp

1 Introduction

Variations of word order are among the most well-known phenomena of natural languages. From a well represented sample of world languages, Steele[13] 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 Grammar (GPSG)[7] provides a method to account for these generalizations by decomposing the grammar rules to Immediate Dominance(ID) rules and Linear Precedence(LP) rules. Using ID/LP formalism, the flexible word order languages can be concisely and more easily described. However, designing an efficient algorithm to put the seperated components back in real parsing is a difficult problem.

Given a set of ID/LP rules, one alternative method for parsing is to compile it into another grammar description language, e.g. Context-Free Grammar(CFG), for which there exist some parsing algorithms. However, the received object grammar tends to be so huge and can slow down the parsing time dramatically. Also, the method losts the modularity of ID/LP formalism.

Another set of approaches[11, 4, 1] tries to keep ID and LP rules as they are, without expanding them out to other formalisms. Shieber[11] has proposed an interesting algorithm for direct ID/LP parsing by generalizing Earley's algorithm[6] to use the constraints of ID/LP rules directly. Despite of its possibility of blowing up in the worst case, Barton[3] has shown 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 is likely to be an appealing candidate for parsing with ID/LP rules from the computational point of view.

In this paper, we present a new approach to direct ID/LP rules parsing that outperforms the previous methods. Besides of the direct parsing property, three features contribute to its efficiency. First, ID rules are precompiled to generalized discrimination networks[9] 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 order legality check at run time. And, third, its bottom-up depth-first parsing strategy minimizes the work of edge check and therefore saves a lot of processing time.

We will first describe briefly each feature of our parser. Then, we will show the parsing algorithm and an example of parsing. The comparisons of our approach with other related works are also described. Finally, we give a conclusion and our future works.

$$\begin{array}{rcll} s & \rightarrow_{ID} & a,b,c,d & (1) \\ s & \rightarrow_{ID} & a,b,e,f & (2) \\ a,b,c & < & d & (3) \\ b & < & c & (4) \\ a,e & < & f & (5) \end{array}$$

Figure 1: An example ID/LP grammar : G_1

2 The Principles of the Parser

2.1 Bottom-up Depth-first Strategy

Chart parsing is one of the most well-known and efficient techniques for parsing general context-free grammars. The chart serves as a book-keeping storage for all parses generated while parsing. In general, to avoid redoing the same tasks, the chart has to be checked every time a new edge is proposed to see whether the identical edge was already generated. Also, when an edge is entered into the chart, it must be checked with other edges to see if it can be merged together to create new edges. In practice, these checks can occupy the majority of parsing time.

In order to build an efficient parser, it is apparent to minimize the checks above. Many different strategies of chart parsers has been developed. Most of them try to minimize the number of useless edges to reduce the checking time.

Our parsing strategy is based on the Word Incorporation (WI) algorithm[12] with some modifications to accommodate ID/LP formalism. We follow WI algorithm by restricting the parsing strategy to be solely bottom-up and depth-first. This makes the parsing proceed along the input in an orderly fashion (left to right or right to left) and keep processing at a vertex until no more new edges ending at the vertex can be generated. Once the parsing go beyond a vertex, the processing will never be redone at that vertex again. As a consequence, the duplicated edge check can be completely omitted. Moreover, once a complete edge is used (for creating new active edges), we can delete it out of the storage since it cannot affect other edges anymore. This reduces the number of edges and hence the checking time.

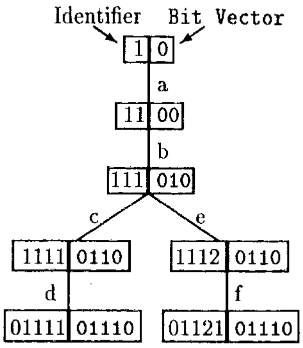

Figure 2: Generalized discrimination network representation of ID rules

2.2 Generalized Discrimination Networks as ID rules compilation

In conventional chart parsing for context-free grammars, a method for reducing the number of edges is precompiling the grammars into discrimination trees. Assume two CFG rules, S → ABCD and S → ABEF. The RHS of the two rules have the common left part AB and therefore can be merged together into a single combined rule: S → AB(CD,EF). In parsing, the common part can then be represented by a single active edge.

However, to apply the method to ID/LP formalism, the case is different. Suppose we have a ID/LP grammar G_1 as shown in Fig. 1. If we view parsing as discrimination tree traversal, the parsing has to proceed in the fixed order from the root to leaf nodes. Because of the order-free characteristic of ID rules, we can no longer just simply combine the ID rules (1) and (2) together as for the two CFG rules above.

To achieve the same merit of discrimination network in the case of CFG rules, we use generalized discrimination network (GDN) for representing ID rules. GDN is a generalization of a discrimination tree that can be traversed according to the order in which constraints are obtained incrementally during the analytical process, independently of the order of the network's arcs. The technique has been first proposed in [9] to be used in incremental semantic disambiguation model but its characteristic also matches our purpose. The technique of GDN is to assign each node in the network a unique identifier and a bit vector. For example, the ID rules of G_1, shown in Fig. 1 ,can be represented as the discrimination network in Fig. 2, of which each node is assigned a unique identifier. 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 k is the numerical number of the arcs issuing from u.[1] Note that the identifier of the root node r has only the first leftmost digit($S(r)$ is null).

As shown in Fig. 2, to each node identifier, we attached a bit vector that has the same length as the identifier and consists of 1's except the leftmost and rightmost bits. These identifiers together with their corresponding bit vectors play an important role in the parsing process with GDN, as will be described later.

Note that representing ID rules by GDN can combine the common parts of different ID rules into the same arcs in the network. Shieber's representation, in contrast, considers each single ID rule seperately and thus cannot achieve this kind of compactness.

2.3 Representing LP rules as a Hasse diagram

Hasse diagram is a representation of partially ordered set used in graph theory[8]. Since a set of LP rules also defines a partially ordered set on a grammar's categories, we can construct its corresponding Hasse diagram. Fig. 3 shows a Hasse diagram for LP rules of G_1. To each node we assign a unique flag and construct a bit vector by setting the flag to '1' and the others to '0'. As for this Hasse diagram, we assign flag(a) the first bit, flag(b) the second bit, ..., and flag(f) the sixth bit. The bit vectors of nodes a, b, c, d, e and f are then 000001, 000010, 000100, 001000, 010000 and 100000, respectively. The precedence vector of each node is the bitwise disjunction between its bit vector and all bit vectors of its subordinate nodes. For example, the precedence vector of f is the disjunction between bit vectors of f, a and e; $100000 \vee 000001 \vee 010000 = 110001$. The resultant precedence vectors are shown in Fig. 3 with 0's in their left parts omitted.

[1]The encoding used here is a little different from the original one in [9].

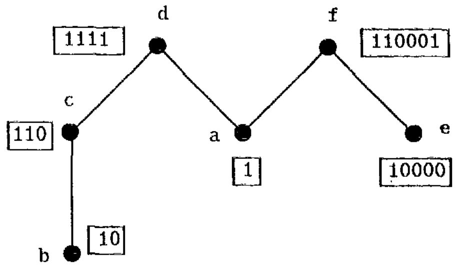

Figure 3: Hasse diagram with the precedence vector assigned to each node

Using the above technique, the order legality check with respect to a given set of LP rules can be efficiently done by the algorithm below:

Algorithm: CheckOrder

Input : Two symbols, A and B with the precedence vector Pre_A and Pre_B respectively, where A precedes B in the input.

1. Take the bitwise disjunction between Pre_A and Pre_B.

2. Check equality: if the result is equal to Pre_A, fail. Otherwise, return the result as the precedence vector of the string AB.

Note that, by the encoding algorithm described in the previous subsection, the precedence vector of a symbol A that must precede a symbol B always be 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 check the legality of next input symbol with respect to all preceded symbols easily by checking with the resultant string's precedence vector only. In real implementation, we can represent a precedence vector by an integer and the order legality can thus be checked efficiently by using Boolean bitwise operations between integers provided in most machines.

2.4 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 a branch and its immediate subordinate node. However, GDN has been proposed to handle the *completely* order-free constraint system. 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 $< Id, BitV >$ to a 4-tuple $< Cat, Id, Pre, BitV >$ where each element is defined as the following:

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 several networks for all nonterminal categories in grammar, *Cat* is added to indicate which networks the state belongs to. Moreover, in addition to the elements used to check ID rules, the precedence vector *Pre* is added for the check of LP rules.

```
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)).
```

Figure 4: Category–state table generated from ID/LP rules : G_1

Next, the constraint-identifier table is replaced by the category-state table, notated as *reduce(category, state)*, viewing each category as a constraint. This table will be used to reduce a constituent to 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. Fig. 4 shows the table derived from G_1.

3 The Parsing Algorithm

Using the table generated from the ID/LP grammar, we can parse by the following algorithm.

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 $w = w_1 w_2 \dots w_n$, where w_i is a terminal in G, we construct the chart as follows:

$k \leftarrow 0$;

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

1. Look up D for the entry of w_{k+1}. Span the inactive(complete) edges corresponding to every possible category 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 of state 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 input string w is accepted if and only if there exists some complete edge of category S from vertex 0 to vertex n in the chart.

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

operation between *Cats* **:** If $Cat_1 = Cat_2$ then return Cat_1. Otherwise, fail;

operation between *Ids* **:** Ignoring the leftmost bit, if Id_1 is a prefix-numerical string of Id_2, return Id_2. Otherwise, fail;

operation between *Pres* **:** As described in CheckOrder algorithm;

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

The operation between *Cats* first checks whether the two states are in the same network. The operation between *Ids* then checks whether one node can be reached from the other in the network. The operation between *Pres* tests whether the catenation of the edges violates LP constraints and return the precedence vector of the successful combined edge as described in section 4. The operation between *BitVs* allows us to cope with the free order of constraints. The bit vector represents all the constraints that must be satisfied between the root node and the reached node. A bit of 0 and 1 means that the corresponding constraint is satisfied and unsatisfied, respectively. For example, the bit vector '0110' in reduce(e, <s,1112,10000,0110>) of the table in Fig. 4 means that by receiving

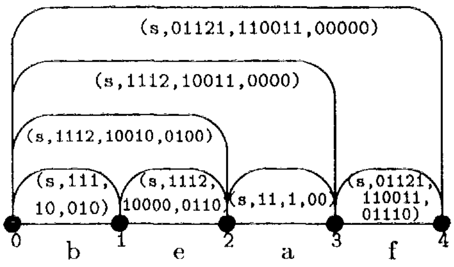

Figure 5: Chart of parsing *beaf*.

e as the input, the constraint e is satisfied and its corresponding rightmost bit in the bit vector will become '0'. In addition, the two 1's mean that we can traverse to the node with the identifier 1112 but another two constraints, a and b, has to be satisfied before. The leftmost bit just makes the vector length the same as that of the identifier and has no corresponding constraint. By taking the conjunction of bits of these vectors, bits of the resultant vector are incrementally changed to 0. Because the bit conjunction operation is executable in any order, it is possible to cope with an arbitrary order of constraints.

Note that one may adopt other mechanisms used in conventional chart parsing to improve the efficiency of the above algorithm.

Example. Suppose we are given the string of categories *b,e,a,f* to parse, using grammar in Fig. 1. First, the edge <s,111,10,010> 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 <s,1112,10000,0110> is then operated with the state <s,111,10,010>. Operation between categories succeeds because both states have the same category s. Operation between identifiers 111 and 1112 succeeds because 111 is a prefix of 1112, thus 1112 is returned. 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 <s,1112,10010,0100> as the edge spanned between vertices 0 and 2.

Continuing in this manner, we will get <s,1112,10011,0000> between vertices 0 and 3, and <s,11121,110011,00000> between vertices 0 and 4. Because the latter is a complete edge of goal category 's', the input string is thus accepted. The chart is shown in Fig. 5.

4 Comparison with Related Works

Other than Shieber's work, there are many works in the past concerning ID/LP parsing. Popowich's FIGG[10] treats ID/LP rules by compiling them into Discontinuos Grammar rules. The different approach of top-down ID/LP parsing using logic grammars is presented by Abramson[1]. This approach is based on using metarules and 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)[5] 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 [2], the approach appears to be considerably faster than Popowich's FIGG.

Another approach of Bottom-up filtering strategy[4] attempts to reduce the nondeterminism in parsing. Different ID rules are constrained to have at most one category in common and the knowledge of the leftmost constituent is used for phrase level initialization.

As an investigation of our approach, we have implemented a small parser, called GHW, using SICStus prolog on a Sun 3-60 workstation. To reduce spurious parses, the parser adopts the technique of the left-corner parsing method to detect the edges that cannot start a constituent in the bottom-up rules invoking stage. The technique is similar to the one used in [4]. GHW is compared with the SDG+metarules and Shieber's parsers running on the same environments. In experimentation, we use a toy grammar taken from [2] that was used to compare SDG+metarules approach with FIGG. The grammar contains 11 ID rules and 4 LP rules. A set of artificial sentences whose lengths are ranged from 2 to 6 is tested on. The timings are averaged over 100 runs using compiled fastcode and reflect the average amount of CPU time in milliseconds required to parse the sentences of several lengths. The result is shown in Fig. 6. Because Shieber's and our parser develop all parses in parallel and thus the time used to find the 1st and all parses are about the same, only the all parses time is shown.

Comparing GHW with Shieber's parser, as expected, GHW outperforms the other for all lengths of the input. When comparing with SDG+metarules parser, for short sentences SDG+metarules wins over our approach in finding the 1st parse, but for longer sentences our approach surpasses it in all cases. This can be explained that because our method needs to do more works of 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. Thus our method may be more suited for using in real applications that concern rather long and complicated sentences. However, this experiment is just a first step of investigating the behaviour of our approach. It remains to be seen how the performance will be for a realistic grammar.

5 Conclusion

A new method for ID/LP rules parsing 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 in-

Length of sentences		n=2	n=3	n=4	n=5	n=6
SDG+metarules	1st	1.2	3.3	6.8	17.	103.
	total	38.	63.	104.	97.	310.
Shieber	total	120.	230.	300.	280.	580.
GHW	total	2.4	4.2	6.8	8.3	12.6

Figure 6: The result of comparison test

tegrating the merits of GDN, Hasse diagram and WI algorithm in parsing.The method is shown to be superior to the previous methods on the tested grammar. However, more explorations have to be done with diverse grammars and sentences to confirm the effectiveness of our method. This is left as one of our further works. Also, extending the parser to handle ill-formed input is under investigation.

Acknowledgements

The authors would like to thank Prof. Harvey Abramson for providing his system and Suresh Katare Gopalrao for checking English in this paper. This work was partly supported by the Telecommunications Advancement Foundation(TAF).

References

[1] Abramson, H. Metarules for Efficient Top-down ID-LP Parsing in Logic Grammars, Technical Report TR-89-11, University of Bristol, Department of Computer Science, 1989.

[2] Abramson, H. and Dahl, V. On Top-down ID-LP Parsing With Logic Grammars, submitted for publication.

[3] Barton, E. On the Complexity of ID/LP Parsing. In *Computational Linguistics*, (October-December 1985), 205-218.

[4] Blache, P. and Morin J. Bottom-Up Filtering: a Parsing Strategy for GPSG. In *Proceedings of the 13th International Conference on Computational Linguistics*, vol. 2, pp. 19-23, 1990

[5] Dahl, V. and Popowich, F. Parsing and Generation with Static Discontinuity Grammars. *New Generation Computing*, vol. 8, no. 3, pp. 245-274, 1990.

[6] Earley, J. An Efficient Context-Free Parsing Algorithm, *Comm. ACM* 13.2:94-102. 1970.

[7] Gazdar, G., E. Klein, G.K. Pullum and I.A. Sag. Generalized Phrase Structure Grammar. 1985.

[8] Liu, C.L. Elements of Discrete Mathematics. McGraw-Hill International Editions. 1986.

[9] Okumura M. and Tanaka H. Towards Incremental Disambiguation with a Generalized Discrimination Network. In *Proceedings Eighth National Conference on Artificial Intelligence*, pp. 990-995, 1990.

[10] Popowich, F.P. Unrestricted gapping grammars. *Computational Intelligence*, vol. 2, pp. 28-53, 1986.

[11] Shieber, Stuart M. Direct Parsing of ID/LP Grammars. *Linguistics and Philosophy* 7(1984), pp. 135-154. 1984.

[12] Simpkins, N.K. and Hancox, P. Chart Parsing in Prolog. *New Generation Computing*, vol. 8, no. 2, pp. 113-138. 1990.

[13] Steel, S. Word Order Variation: A typological Study. In *J. Greenberg(ed.) Universals of Language*, vol. 4. Stanford, CA: Stanford Universit y Press. 1981.

[14] Winograd T. *Language as a Cognitive Process*, vol. 1, Syntax, Addison-Wesley. 1983.