# Direct 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

## Abstract

We present a new parsing method using ID/LP rules directly without transforming to context-free grammar rules. The method regards parsing as traversal of the generalized discrimination networks and represents the parsing process as states in the networks. This can yield more compact representation of parser's state sets compared with the previous methods. We also optimize LP rules checking by compiling the LP constraints to a Hasse diagram so that it can be checked efficiently using bitwise operations.

## 1 Introduction

It is widely admitted that variations of word order in natural languages are governed by generalizations that should be expressed by the grammars. Generalized Phrase Structure Grammars(GPSG)[4] provide a method to account for these generalizations by decomposing the grammar rules to Immediate Dominance(ID) rules and Linear Precedence(LP) rules. ID rules just constrain that what constituents may appear as the daughters within a given constituent but, as opposed to conventional rule specification, constrain nothing about the order among them. Instead, the order constraints are specified seperately in LP rules. Using ID/LP formalism, the flexible word order languages can be more easily and concisely described. Given a set of ID/LP rules, one alternative for parsing is to compile them 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 it does not take the language generalizations in the grammars into account while parsing. Moreover, the object grammar may be so huge that will cause parsing to take a lot of time. Another set of approaches[7, 1] tries to parse by using ID/LP rules as they are without transforming to other formalisms. Shieber[7] gives a direct parsing algorithm which is a simple generalization of Earley's algorithm[3]. Barton[2] showed that Shieber's 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.

However, Shieber's algorithm may still suffer from the combinatorial explosion of the number of intermediate states while parsing. Although this cannot be avoided because of the NP-complete characteristic inherent in the problem, it does not preclude algorithms with better average performance. We could search for more efficient algorithm that can (1) further pack the states or (2) reduce the time used in other organizations of parsing process as much as possible. In this paper, we present an efficient method for direct ID/LP rules parsing that provides the desirable properties for both aspects above.

## 2 ID Rules as Generalized Discrimination Networks

In fact, the reason why 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. The multiset representation in Shieber's algorithm saves a lot by keeping many possibilities unexpanded in its generalized dotted rule. In this section, we present the method that can further save more with its compact rule representation as generalized discrimination networks[6].

Given a set of constraints defining a concept, one can build a corresponding discrimination network used to check the concept efficiently. Considering the ID rules with the same LHS element as a set of constraints to construct that constituent structure, we can thus represent such a set as a discrimination network by viewing each element in RHS of an ID rule as an arc in the network. 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. Moreover, since a discrimination network is constructed regarding with the set of ID rules with the same LHS, we can also compact RHS elements of the same form in different rules into one arc in the network. Shieber's representation, in contrast, considers each single ID rule seperately and hence cannot capture this kind of compactness.

However, the discrimination network has a problem in that it cannot be traversed unless constraints are entered in an a priori fixed order, and thus may be in trouble with order-free constraints of ID rules. Okumura et al.[6] proposes the method of generalized discrimination networks(GDN) to solve the problems. GDN is a generalization of a discrimination network

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

Figure 1: An example ID/LP grammar : $G_1$

which can be traversed according to the constraints obtained incrementally during the analytical process, independently of order. Then, parsing process can be viewed as traversal of the GDN incrementally downward to the leaf nodes. For example, the ID rules of $G_1$, shown in Figure 1 ,can be represented as the GDN in Figure 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 catenation of the sequence $S(u)$ and the integer $k$, where $u$ is the immediate predecessor of $v$ and arc $u$-$v$ is the $k$th element in the ordered set of arcs issuing from $u$.[1] 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, which has the same length as the identifier and consists of 1's except the leftmost and rightmost bits, is attached. These identifiers together with their corresponding bit vectors play an important role in the parsing process with GDN, as will be described later.

## 3 LP rules as a Hasse diagram

### 3.1 Constructing the diagram for a set of LP constraints

Given a set of LP constraints, we can build the corresponding so called 'Hasse diagram' [2] to represent the precedence relation between the relevant symbols as follows.

[1] The encoding used here is a little different from the original one in [6].

[2] Hasse diagram is a representation of partially ordered set used in graph theory.

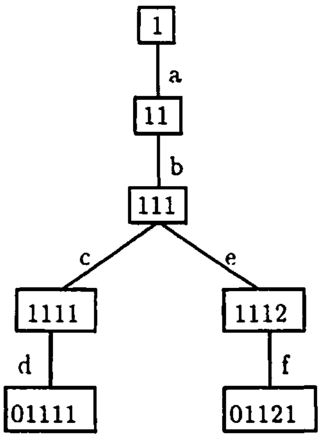

Figure 2: discrimination network representation of ID rules

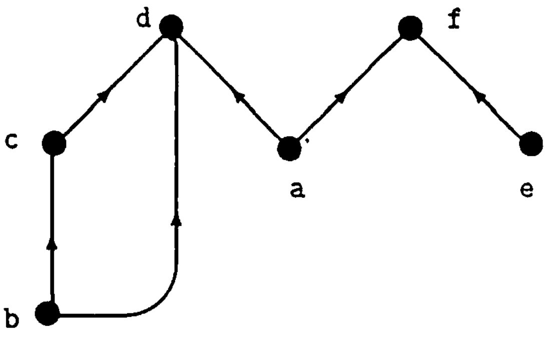

Figure 3: The directed graph representing the precedence between symbols

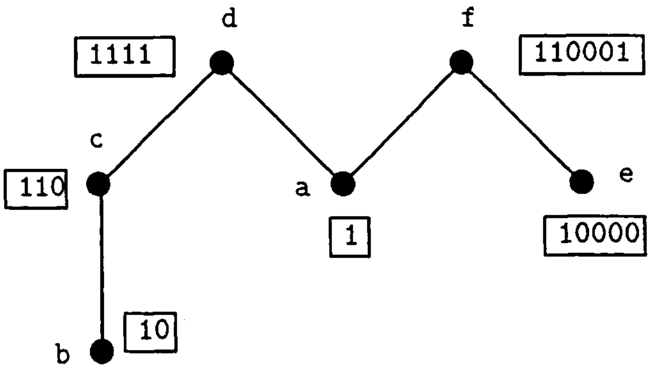

Figure 4: Hasse diagram with the precedence vector assigned to each node

First, we build a directed graph representing these relations where a node represents a symbol and an arc between two nodes represents the precedence between them, with the direction of an arc goes from lower precedence node to higher one, as shown in Figure 3. Next, we can simplifiy this graph into Hasse diagram by deleting unnecessary arcs and omitting the arrowheads representing direction. As for Figure 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. Finally, assign each node a unique flag and compute the precedence vector of each node from the disjunction of that node's flag with all of the flags of nodes below it in the diagram.

As for this example, the set of all nodes is $\{a, b, c, d, e, f\}$ and we assign each node a unique flag represented as an integer which has one bit set to '1' and the rest reset to '0',i.e., flag($a$) = 1, flag($b$) = 10, ..., flag($f$) = 100000. These flags are then used to compute the precedence vectors of all nodes. The resultant diagram is shown in Figure 4.

## 3.2 Order legality checking

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

### Algorithm: CheckOrder

*Input* : Two symbols, $A$ and $B$ with the precedence vector $Pre_A$ and $Pre_B$ respectively, where $A$ is before $B$ 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 $AB$.

By the reason that the precedence vector of symbol $A$ that must precede symbol $B$ always be included in $B$'s precedence vector, so, 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. In real implementation, we can represent a precedence vector by an integer and the order legality can thus be checked efficiently in constant time by using Boolean bitwise operations between integers provided in most machines.

## 4 Compiling the grammar into the table

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. But, as described, GDN is first proposed to handle the *completely* order-free constraint system. So, in order to apply the model to natural language parsing where 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 to a 4-tuple (*Cat,Id,Pre,BitV*) where each element is defined as the following:

*Cat* : the mother category the state belongs to;
*Id* : the identifier of the state;

```
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)).
reduce($s$,$goal$).
```

Figure 5: Category-state table generated from ID/LP rules : $G_1$

*Pre* : the precedence vector of the state;
*BitV* : the bit vector of the state.

Next, the GDN's 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 when a category is complete (e.g. the current state is at a leaf node and all bits of *BitV* are set to 0) to reduce it to higher level constituent states.

Figure 5 shows the generated table after compiling $G_1$. Note that the special symbol *goal* is introduced for representing the state that the category in the top level will be reduced to.

## 5 Direct ID/LP Parsing Algorithm

Using the table generated from the ID/LP grammar as above, we can parse the input by the algorithm below. The algorithm is based on chart parsing method[5], with slightly modification.

### Algorithm

Given a table $T$ generated from ID/LP grammar $G$ and an input string $s = w_1 w_2 \ldots w_n$, where $w_i$ is a terminal category in $G$, we construct chart as follows:

```
$k \leftarrow 0$;
while $k < n$ do begin
```

1. For all $\alpha$ of *reduce*($w_{k+1}$, $\alpha$), span the edge $\alpha$ between vertices $k$ and $k+1$.

Now perform steps (2) and (3) until no new edges can be added.

2. For each inactive(complete) edge of category $\beta$ spanned between vertices $j$ and $k+1$ ($j < k+1$), if *reduce*($\beta$, $\phi$) is an entry in $T$, span the edge $\phi$ between vertices $j$ and $k+1$.

3. For each active(incomplete) edge of category $\beta$ 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 exist some *goal* edges 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** *Cats* : If *Cats* are the same then return *Cat*. Otherwise, fail;

**operation between** *Ids* : If one *Id* includes the other as a prefix-numerical string, return the longer string. Otherwise, fail;

**operation between** *Pres* : As described in Check-Order 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 corresponding 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 concatenation of the edges violates LP constraints and returns 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.[3] For example, state ($s$,1112,10000,0110) of table in Figure 5 represents the satisfaction of the constraint corresponding to the rightmost bit, which is $e$. 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.

The above algorithm is based on the plain bottom-up chart parsing method by the reason of clarity, though one can add other strategies, e.g. lookahead, to improve the efficiency of the algorithm.

## 6 An Example

Suppose we are given the string of categories $b,e,a,f$ to parse, using grammar in Figure 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 $s$ category. 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 the edges ($s$,1112,10011,0000) between vertices 0 and 3, and ($s$,01121,110011,00000) between vertices 0 and 4. Because the latter is a complete edge and then can be reduced to *goal* state, the input string is thus accepted.

[3] The leftmost bit has no corresponding constraint and makes the vector length the same as that of the identifier.

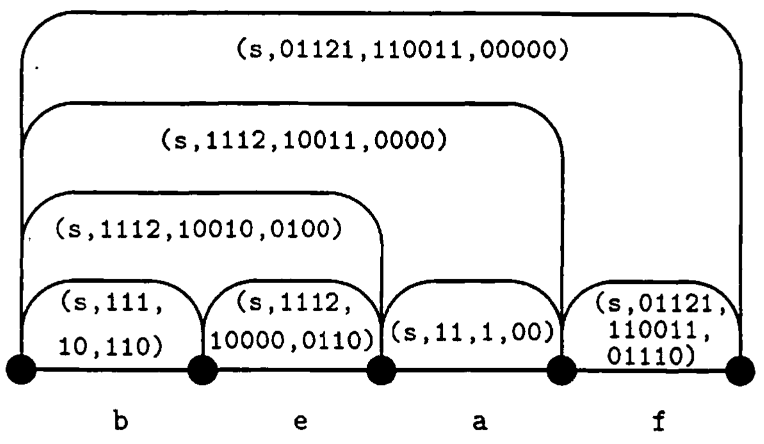

Figure 6: Chart of parsing *beaf*.

The chart is shown in Figure 6.

## 7 Conclusion

The direct parsing method presented here tackles two points: data compactness and efficient LP constraints checking. The former is handled by representing ID rules as GDN and the state of the GDN is then used to track the parsing process. This can reduce the number of data structures generated while parsing because we can keep unexpanded until needed many possibilities of the states at the nodes below a state. This is similar to the merit of data structure used by Shieber. However, unlike Shieber's, common elements of different rules can be merged into one arc in the network. This yields more compact rules representation and thus more wasteful recomputation can be avoided. LP rules checking is also optimized by compiling to a Hasse diagram and computing each category's precedence vector. The LP constraints are embedded in these vectors and also propagated with the precedence vectors of the resultant strings. These vectors are then used to check order legality efficiently using machine instruction of bitwise operations.

Our future works include investigating how the method works in parsing general grammars and then improving the deficiencies found.

## 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] Barton, E. On the Complexity of ID/LP Parsing. In *Computational Linguistics*, pp. 205-218, October-December 1985.

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

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

[5] Kay, M., Algorithm Schemata and DataStructures in Syntactic Processing. *Readings in Natural Language Processing*, B. J. Grossz et al. eds, pp. 35-70, 1986.

[6] 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.

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