1 1 − 4

Direct ID/LP Parsing with a Generalized Discrimination Network

Surapant Meknavin* Manabu Okumura Hozumi Tanaka

Tokyo Institute of Technology

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. We also optimize LP rules checking so that it can be checked efficiently in constant time.

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)[3] provide a method to account for these generalizations by decomposing the grammar rules to Immediate Dominance(ID) rules and Linear Precedence(LP) rules. One alternative for parsing is to compile ID/LP rules into another grammar description language, e.g. Context-Free Grammars, for which there exist parsing algorithms. These approaches lacks of the naturalness, especially for "free word order language", in the sense that they do not take the languages' generalizations in the grammars into account while parsing. Another set of approaches try to parse by using ID/LP rules as they are without transforming to other formalisms. Barton[1] showed that Shieber's direct parsing algorithm[5] usually does have a time advantage over the use of Earley's algorithm[2] 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 viewpoint.

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 the 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[4].

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, common elements of different rules can be merged into one arc in the network. This yields more compact rules representation and the wasteful recomputation can be avoided.

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(1990) proposes the method of generalized discrimination networks, or GDN, to solve the problems. GDN is a generalization of a discrimination

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

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

Figure 1: An example ID/LP grammar : G_1

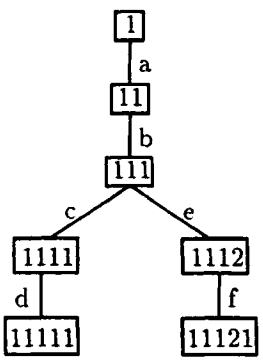

Figure 2: discrimination network representation of ID rules

network which can be traversed according to the order in which constraints are 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 2 ,can be represented as the GDN in Figure 2, of which each node is assigned a unique identifier. The identifier of a node v is 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 kth element in the ordered set of arcs issuing from u. To each node identifier, a bit vector is attached which has the same length as the identifier and consist 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.

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' to represent

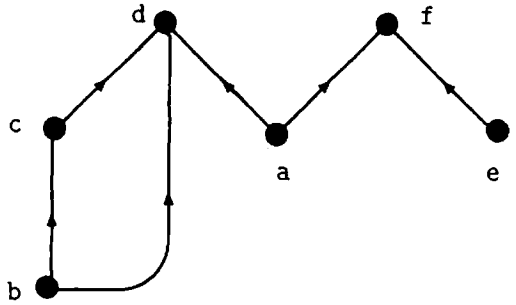

Figure 3: The directed graph representing the precedence between items

the precedence relation between the relevant items as follows.

First, we build a directed graph representing these relations where a node represents an item 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 arrow representing direction. In Figure 3, we can delete arc b-d because we know that "$<$" is a transitive relation, consequently, the existence of arcs b-c and c-d is enough to imply the arc b-d. In addition, if we restrict that the direction of the arrow between nodes always go upward, then we can omit all the arrows. 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 to each node as a unique flag 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

Algorithm: CheckOrder

Input : Two items, 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, success and return the result as the precedence vector of the string AB.

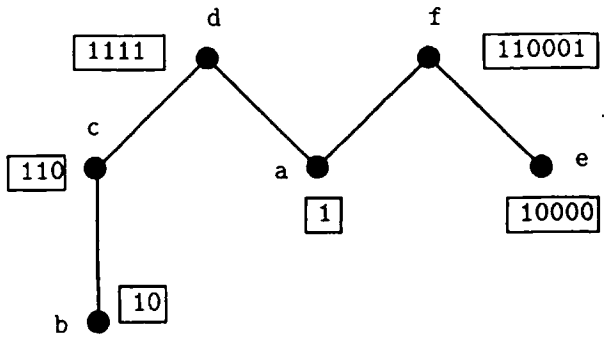

Figure 4: Hasse diagram with the precedence vector assigned to each node

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;

Pre : the precedence vector of the state;

BitV : the bit vector indicating those categories required to reach the state.

Next, the constraint-identifier table is replaced by the category-state table, notated as $table(category, state)$, viewing each category as a constraint. Third, for every nonterminal categories appearing in LHS of some ID rules, we have to know when the categories are complete and can be reduced to higher level constituent states. So the state-reduced_state table, notated as $reduce(state1, state2)$, is also built to serve for this purpose. A mother category is completed(eg. can be reduced) just in case the current state is at a leaf node and all bits of *BitV* are set to 0.

```
table(a,(s,11,1,00)).
table(b,(s,111,10,010)).
table(c,(s,1111,110,0110)).
table(d,(s,11111,1111,01110)).
table(e,(s,1112,10000,0110)).
table(f,(s,11121,110001,01110)).
reduce((s,11111,_,00000),goal).
reduce((s,11121,_,00000),goal).
```

Figure 5: Table generated from ID/LP rules : G_1

Figure 5 shows the generated table after compiling G_1, written in Prolog syntax. Note that the special symbol *goal* is used to represent the state that the category in the top level will be reduced to.

5 Direct ID/LP Parsing 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 $table(C, \alpha)$, span the inactive edge α between vertices k and $k+1$.

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

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

3. For each active edge 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 success, 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 *Cat* : If *Cat*s are the same, successes and return *Cat*. Otherwise, fail;

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

operation between *Pre* : As described in Check-Order algorithm;

operation between *BitV* : After adjusting the length of *BitV*s 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.

Example: Suppose we are given the string of categories *b,e,a,f* to parse, using grammar in Figure 2. 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 (s,1112,10011,0000) between vertices 0 and 3, and (s,11121,110011,00000) between vertices 0 and 4. Because the latter can be reduced to goal state, the input string is thus accepted. The chart is shown in Figure 6.

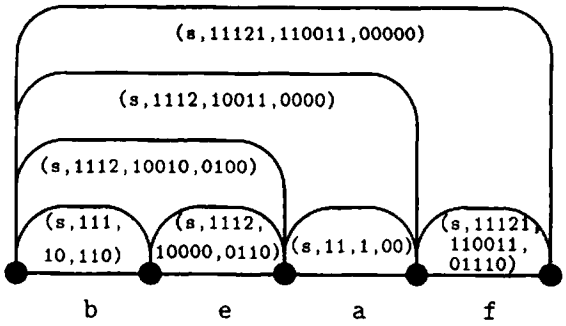

Figure 6: Chart of parsing *beaf*.

6 Conclusion

The direct parsing method presented here attacks 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. Moreover, a GDN is constructed regarding with the set of ID rules with the same LHS, so it can 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. LP rules checking is also optimized by compiling to a Hasse diagram and computing each category's precedence vector. The LP constraints are embeded in these vectors and also propagated with the precedence vector of the resultant string. These vectors are then used to check order legality efficiently using machine instruction of bitwise operations.

References

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

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

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

[4] 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.

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