Parsing Ill-Formed Input with ID/LP rules

Surapant Meknavin Thanaruk Theeramunkong 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

In this paper, we describe our new ill-formed input parsing strategy which uses ID/LP grammar formalism directly. The strategy can process errorful input, hypothesize the errors, and provide possible alternative interpretations for the input. The error recovery process is triggered by failure of finding a parse of goal category in the normal parsing process. The previous methods of parsing ill-formed input on CFGs and their problems are first described. Then we go on to suggest our new methods and show how it can be represented by a logic program. We also show that our method can be more efficient than the alternative of parsing on CFGs for flexible word order languages.

1 introduction

In recent years, works in natural language processing, particularly within syntactical analysis, have been developed rapidly to a satisfactory level that can be used in real applications. Many works use context-free grammars (hereafter CFG) to describe natural languages because of their simplicity and tractability. By using a suitable parsing algorithm, the languages which can be described by CFGs are proved to be recognizable in polynomial time. However, using a single kind of CFG rules to describe various surface characteristics of a natural language, such as word order variation in a free word order language, can lead to a huge grammar which does not give useful information or intuition of the language.

Recent grammar formalisms have changed their ways of describing a language by using several kinds of independent constraints instead of one single kind of rules as in a CFG [3, 4, 5]. Using these formalisms can result in shorter and clearer language descriptions. Moreover, some of these formalisms can account for some kinds of context-sensitiveness that are out of the power of CFGs.

To use any formalism in real applications of natural language parsing, however, ill-formed input handling is an important issue. The way people use language in their daily life is so different from what is told in grammar textbooks. For example, they often omit some words,

change order of phrases, or cause some careless errors such as tense errors, misspellings or extra words. For a particular natural language processing system, users may use some vocabularies or patterns that are beyond the coverage of the lexicon or grammar of the system. Encountering such ill-formed input, a robust system should not just reject the input, but should process them reasonably.

Recently, many methods that deal with ill-formed input have been presented in the literature [8, 6]. However, almost all of them are based on only CFGs. In this paper, we put our focus on parsing ill-formed input with ID/LP rules. By using ID/LP rules, flexible word order languages can be stated more systematically and compactly than by using CFG rules. Many works concerning ID/LP parsing on well-formed input [7, 9] have been done, but none of them has touched the problem of parsing ill-formed input. In the rest of this paper, we will first describe the previous methods of parsing ill-formed input with ordinary CFGs and their problems. Then we will describe our parsing method which uses ID/LP rules directly in parsing ill-formed input and show how it can be represented as a logic program. The comparison of our method with the previous works is also discussed.

2 Chart parsing of ill-formed input

In recent years, the application of a chart parser to deal with ill-formed input have been interested by many researchers[8, 6]. The most attractive point of using a chart parser is that the information of partial parses generated by the parser can be used for hypothesizing errors in many flexible ways. Mellish [8] shows how to combine the advantages of bottom-up and top-down chart parsers in parsing ill-formed input using CFGs. The processing of his method can be divided into two phases: bottom-up phase and top-down phase. The bottom-up phase is done first by using a left corner chart parser without top-down filtering to look for a complete parse of the goal category. The parsing will successfully finish in this phase if the input is well-formed. Otherwise, the parsing will continue to the top-down phase where top-down parsing is performed to hypothesize the minimal errors that, if corrected, can complete possible parses. Edges in the chart are assessed by a number of parameters to decide which edges should play a role in error recovery. This method has the advantage that parsing of well-formed sentence is not affected by the added extra mechanism for the recovery process and the recovery process itself will never repeat the same work done before by the bottom-up parsing. Moreover, the use of a declarative grammar formalism (e.g. CF-PSG) allows constructing partial parses that do not start at the beginning of the input. In this way, in contrast to ATN parser [10], it can use both left and right context in the determination of the best parse for an ill-formed sentence.

Kato [6] proposed an improved version of Mellish's parsing method. The method is similar to [8] in that it also employs a combination of bottom-up and top-down parsing strategies. However, different from [8], Kato extends the process of edge completion to use the information received from bottom-up phase more to prune search space before starting top-down search. A special phase called the edge completion phase is added between bottom-up phase and top-

down phase. The newly added phase is for constructing all active edges that are not generated in the initial bottom-up phase due to the left to right characteristic of the left corner parser. These edges can help reducing search space of top-down phase and thus make finding errors done more efficiently. Nevertheless, the method still has the following problems.

- Edge overgeneration

  In top-down phase, Mellish's parser can overgenerate many edges that are not relevant to errors in input. Kato tries to suppress this by exploiting the edges generated bottom up to reduce search space before starting to search top down. The edge completion process which does this, however, can also take a lot of time and overgenerates many active edges, especially when the length of RHS of grammar rules is long and the number of rules describing a LHS category is large. We will discuss this in more detail in the following sections.

- Parsing flexible word order language

  As described earlier, a language which has flexible word order can need a large number of CFG rules to describe its grammar. For such a language, the CFG-based methods discussed above may be unsuitable and suffer with huge search space. Also, the methods cannot handle sentences with some errors in word order, though the errors are common in natural input. Even for languages which have fixed word order in general, like English, the occurrences of non-standard word order are often found [2].

3 ID/LP rules and ill-formed sentences

Here, we propose a new strategy in parsing ill-formed input by using ID/LP rules directly. There are a number of reasons why we make this proposal:

1. Using ID/LP rules in parsing ill-formed input of flexible word order language may have an advantage over using CFG rules in the same way as in parsing well-formed (grammatical) input. This is because the number of edges or states generated would be much smaller than when CFG grammars are used.

2. As ID/LP rules formalism is proposed to capture the generalization of word order in a language, it is reasonable to think of it as a suitable tool for handling word order errors. The use of non-standard word order can be handled simply by allowing violations of some LP rules.

3. By allowing violations of some ID rules, other kinds of errors can also be handled straightforwardly. For instance, ellipsis of constituents can be handled by allowing the absence of some daughters in ID rules and extra word errors can be handled by allowing addition of some constituents into ID rules.

3.1 Our strategy

Our basic strategy is essentially similar to the strategy of Kato : to run a bottom-up parser over the input (BU phase) and then, if this fails to find a complete parse, to generate all edges that were blocked in the bottom-up phase due to the left-to-right characteristic of the parser (EC phase), and finally to run a modified top-down parser over the resulting chart to hypothesize possible complete parses (TD phase). Our parser can handle four kinds of primitive errors: extra word error, omitted word error, unknown word error and word order error. For simplicity, we will first focus on the first three kinds of errors. Handling word order errors will be discussed later in section 3.2.

BU phase

To accommodate ID/LP grammar formalism, the data structure of edges has to be modified. We use the following notation to represent an edge in our parser:

$< C$ from S_1 to E_n
found C_1 from S_1 to E_1, ..., C_n from S_n to E_n needs $Cs >$

where C is a LHS category, C_i are a set of RHS categories found already, Cs is an *unordered* needs set of RHS categories remains to be found, S_i, E_i are positions in the chart. An edge of this form indicates that the parser is attempting to find a phrase of category C, already found the subphrases of $C_1, C_2, \ldots, C_n$ in the range starting from S_1 to E_n, but in order to succeed it must still satisfy all categories in the needs set Cs. Note that E_i is not necessarily equal to S_{i+1}, and the unfound categories possibly exist between them. Also, Cs represents an unordered set instead of an ordered sequence of categories. For instance, $<$ S from 0 to 0 found {} needs {NP,VP}> represents the edge of category S that spans from position 0 to 0 (found nothing yet), and needs NP and VP (in any order) to complete the constituent. Similarly, $<$ S from 0 to 2 found {VP} from 0 to 2 needs {NP}> represents the edge of category S that found one of its subconstituent VP from position 0 to 2, and still needs NP to complete the constituent. To construct edges like the latter whose an element in the needs set is found in the input string, the order legality between the element and the other remaining elements in the needs set must be checked with respect to LP rules. In this case, there must be no LP rule like NP $\prec$ VP, otherwise such an edge cannot be generated. This check plays a role like the left corner check in standard chart parsing. Given these replacements of notation and operation on edges, our BU phase can be done analogous to ordinary bottom-up chart parsing.

To see more vividly how our method works, let us consider a simple ID/LP grammar G_1 of a flexible word order language that has only one ID rule, one LP rule and four lexicon rules.

G_1:

$$S \quad \rightarrow_{ID} \quad A, B, C, D$$
$$A \quad \prec \quad C$$

$$A \rightarrow a$$
$$B \rightarrow b$$
$$C \rightarrow c$$
$$D \rightarrow d$$

If the input string $axcd$ is parsed by using our method directly on G_1, the following edges will be generated in BU phase:

(1) $< A$ from 0 to 1 found $\{a\}$ from 0 to 1 needs $\{\}>$
(2) $< C$ from 2 to 3 found $\{c\}$ from 2 to 3 needs $\{\} >$
(3) $< D$ from 3 to 4 found $\{d\}$ from 3 to 4 needs $\{\} >$
(4) $< S$ from 0 to 1 found $\{A\}$ from 0 to 1 needs $\{B, C, D\} >$
(5) $< S$ from 3 to 4 found $\{D\}$ from 3 to 4 needs $\{A, B, C\} >$

Note that the active edge of category S, $< S$ from 2 to 3 found $\{C\}$ from 2 to 3 needs $\{A, B, D\} >$, is not generated because it is blocked by the LP rule $A \prec C$ in G_1.

In contrast, consider what happens if the same string is parsed using ordinary bottom-up chart parsing on the equivalent CFG G_1'. G_1' has 12 rules spelling out all possible strings formed by A, B, C and D, whose A is prior to C.

G_1':

$$S \rightarrow A,B,C,D \qquad S \rightarrow A,B,D,C$$
$$S \rightarrow A,C,B,D \qquad S \rightarrow A,C,D,B$$
$$S \rightarrow A,D,B,C \qquad S \rightarrow A,D,C,B$$
$$S \rightarrow B,A,C,D \qquad S \rightarrow B,A,D,C$$
$$S \rightarrow B,D,A,C \qquad S \rightarrow D,B,A,C$$
$$S \rightarrow D,A,B,C \qquad S \rightarrow D,A,C,B$$

As illustrated below, for this example the advantage of parsing on ID/LP grammar directly starts from BU phase, since the number of edges generated in this phase by ordinary bottom-up chart parser is much larger than the one generated by our method. This is similar to the advantage of using Shieber's algorithm on ID/LP grammar to parse well-formed input over using Earley's algorithm on the corresponding CFG [1]. Here, the form of the generalized edge [8] is used:

$<$ C from S to E

needs Cs_1 from S_1 to E_1, ..., Cs_n from S_n to $E_n >$

where C is a category, Cs_i are lists of categories, and S, E, S_i, E_i are positions in the chart. The found portion is omitted in this form. Note that S and E indicate the range of a whole phrase of category C, not only of the found portion as in our form.

(6) $< A$ from 0 to 1 needs $\{\} >$
(7) $< C$ from 2 to 3 needs $\{\} >$
(8) $< D$ from 3 to 4 needs $\{\} >$
(9) $< S$ from 0 to $*$ needs $\{B, C, D\}$ from 1 to $* >$
(10) $< S$ from 0 to $*$ needs $\{B, D, C\}$ from 1 to $* >$
(11) $< S$ from 0 to $*$ needs $\{C, B, D\}$ from 1 to $* >$
(12) $< S$ from 0 to $*$ needs $\{C, D, B\}$ from 1 to $* >$
(13) $< S$ from 0 to $*$ needs $\{D, B, C\}$ from 1 to $* >$
(14) $< S$ from 0 to $*$ needs $\{D, C, B\}$ from 1 to $* >$
(15) $< S$ from $*$ to 4 needs $\{A, B, C\}$ from $*$ to 3 $>$
(16) $< S$ from $*$ to 4 needs $\{A, C, B\}$ from $*$ to 3 $>$
(17) $< S$ from $*$ to 4 needs $\{B, A, C\}$ from $*$ to 3 $>$

For general grammars, the advantage of using ID/LP rules depends on the number of LP rules and the length of RHSs of ID rules. When the number of LP rules is small and the lengths of RHSs of ID rules are long, the large advantage can be obtained.

EC phase

Some subparses were not generated in BU phase due to the left to right characteristic of (left corner) bottom up parsing. For instance, in the above example, no parse covering C is generated because C cannot appear as the leftmost element of S. However, for ill-formed input parsing this is not desirable since it will block many subparses useful for hypothesizing errors. In EC phase, we attempt to generate the edges corresponding to those blocked subparses. In order to do this, the bottom up rule and the fundamental rule of chart parsing have to be modified to allow operating from arbitrary positions in the RHS of a grammar rule or in the undeterminated portion of an active edge, not just from the leftmost position. The modified bottom-up rule provokes new active edges by matching inactive edges in the chart with arbitrary RHS elements other than the leftmost ones of grammar rules (since the edges for the leftmost elements would be already generated in BU phase). The modified fundamental rule provokes new active edges by making the completion between inactive edges and arbitrary positions of undeterminated parts in active edges. These two rules play an important role in reducing search space in TD phase [6].

The modified bottom-up rule can be realized in our method by simply omitting the order check of the category of RHS categories whose constituent is found in input with the remaining categories. In a similar way, the modified fundamental rule can be realized by omitting the order check of the element of the needs set whose constituent is found in input with the remaining elements. Instead, the order check is performed between the element in question and all found categories. In the case of the above example, the following active edge will be generated by the bottom-up rule as the checks between C and the categories A, B and D are omitted.

(18) $< S$ from 2 to 3 found $\{C\}$ from 2 to 3 needs $\{A, B, D\} >$

The edge is then matched with inactive edges in the chart like the one of D from 3 to 4 by the fundamental rule. In this case, because C can precede D according to the LP rule in G_1, the active edge $< S$ from 2 to 4 found $\{C\}$ from 2 to 3 $\{D\}$ from 3 to 4 needs $\{A, B\} >$ is generated.

However, applying the fundamental rule to all edges in the chart as in Kato's method is not quite a good method, since the parser must spend time attempting to match a great number of active edges and inactive edges pairs, as shown in Fig. 1. Moreover, many useless edges which will never contribute in finding a complete parse for the goal category are also generated. Our method thus applies only the bottom-up rule in EC phase, and for this example generates only one active edge above. The fundamental rule is applied later upon requests in TD phase to guarantee that the edges generated will contribute in parsing for the goal category and only edges in the specified range are needed to be matched. This will be discussed shortly.

TD phase

In this phase, the data structure of a special type of edges used in searching, which we will call "searcher", is almost the same as in [6] except that Cs_i denotes an unordered multiset instead of an ordered list of categories.

$<$ hole: N err: M Cs_1 from S_1 to E_1, ..., Cs_n from S_n to $E_n >$

where the value N of "hole" portion is the number of categories remained to be satisfied, M of "err" portion is the number of errors accumulated so far in a searcher, and S_i, E_i are positions in the chart.

We perform top-down search in this phase according to the search rules in Fig. 2 and Fig. 3. The search starts off with the searcher $<$ hole: 1 err: 0 $[S]$ from 0 to L $>$, where S is the start symbol (goal category), L is the length of input, and find a result when a searcher of the form $<$ hole: 0 err: M [] $>$ is derived. The first two rules, the active fundamental rule and the top-down rule, are for tracing unsatisfied portions, and the remaining three rules are for determining 3 kinds of primitive errors: extra word error, omitted word error and unknown word error. The predicates *select/3* and *valid_order/2* in the body of several rules are used to select a category from the leftmost part of the needs and to check that the selected category can precede the rest categories. The active fundamental rule performs the completion between searchers and previously generated active edges to narrow the undecided portions, and is used as the main rule in search process. The top-down rule refines an expectation of a category into more detailed constituents according to all ID rules whose LHS category is the expected category. At any point, all rules are applied to generate new searchers which will be kept with previously generated searchers. Then the searcher which has the least summation of hole and error value will be selected to do searching in the next time. If the summations are equal, the one with less value of hole will be selected. If there are still many candidates left, one of them will be selected.

Edges spawned from A with bottom up rule:

(19) $< S$ from $*$ to $*$ needs $\{B\}$ from $*$ to 0 $\{C, D\}$ from 1 to $* >$
(20) $< S$ from $*$ to $*$ needs $\{B\}$ from $*$ to 0 $\{D, C\}$ from 1 to $* >$
(21) $< S$ from $*$ to $*$ needs $\{D\}$ from $*$ to 0 $\{B, C\}$ from 1 to $* >$
(22) $< S$ from $*$ to $*$ needs $\{D\}$ from $*$ to 0 $\{C, B\}$ from 1 to $* >$
(23) $< S$ from $*$ to $*$ needs $\{B, D\}$ from $*$ to 0 $\{C\}$ from 1 to $* >$
(24) $< S$ from $*$ to $*$ needs $\{D, B\}$ from $*$ to 0 $\{C\}$ from 1 to $* >$

Edges spawned from D with bottom up rule:

(25) $< S$ from $*$ to $*$ needs $\{A\}$ from $*$ to 3 $\{B, C\}$ from 4 to $* >$
(26) $< S$ from $*$ to $*$ needs $\{A\}$ from $*$ to 3 $\{C, B\}$ from 4 to $* >$
(27) $< S$ from $*$ to $*$ needs $\{B\}$ from $*$ to 3 $\{A, C\}$ from 4 to $* >$
(28) $< S$ from $*$ to $*$ needs $\{A, B\}$ from $*$ to 3 $\{C\}$ from 4 to $* >$
(29) $< S$ from $*$ to $*$ needs $\{A, C\}$ from $*$ to 3 $\{B\}$ from 4 to $* >$
(30) $< S$ from $*$ to $*$ needs $\{B, A\}$ from $*$ to 3 $\{C\}$ from 4 to $* >$
(31) $< S$ from $*$ to $*$ needs $\{A, B, C\}$ from $*$ to 3 $>$
(32) $< S$ from $*$ to $*$ needs $\{A, C, B\}$ from $*$ to 3 $>$
(33) $< S$ from $*$ to $*$ needs $\{B, A, C\}$ from $*$ to 3 $>$

Edges spawned from C with bottom up rule:

(34) $< S$ from $*$ to $*$ needs $\{A\}$ from $*$ to 2 $\{B, D\}$ from 3 to $* >$
(35) $< S$ from $*$ to $*$ needs $\{A\}$ from $*$ to 2 $\{D, B\}$ from 3 to $* >$
(36) $< S$ from $*$ to $*$ needs $\{A, B\}$ from $*$ to 2 $\{D\}$ from 3 to $* >$
(37) $< S$ from $*$ to $*$ needs $\{B, A\}$ from $*$ to 2 $\{D\}$ from 3 to $* >$
(38) $< S$ from $*$ to 3 needs $\{A, B, D\}$ from $*$ to 2 $>$
(39) $< S$ from $*$ to 3 needs $\{A, D, B\}$ from $*$ to 2 $>$
(40) $< S$ from $*$ to 3 needs $\{B, A, D\}$ from $*$ to 2 $>$
(41) $< S$ from $*$ to 3 needs $\{B, D, A\}$ from $*$ to 2 $>$
(42) $< S$ from $*$ to 3 needs $\{D, A, B\}$ from $*$ to 2 $>$
(43) $< S$ from $*$ to 3 needs $\{D, B, A\}$ from $*$ to 2 $>$

Edges generated by fundamental rule:

(44) $< S$ from $*$ to $*$ needs $\{A\}$ from $*$ to 2 $\{B\}$ from 4 to $* >$ (C+D)
(45) $< S$ from $*$ to 4 needs $\{A, B\}$ from $*$ to 2 $>$ (C+D)
(46) $< S$ from $*$ to 4 needs $\{B, A\}$ from $*$ to 2 $>$ (C+D)
(47) $< S$ from $*$ to 4 needs $\{A\}$ from $*$ to 2 $\{B\}$ from 3 to 3 $>$ (C+D)
(48) $< S$ from $*$ to 4 needs $\{B\}$ from $*$ to 2 $\{A\}$ from 3 to 3 $>$ (C+D)
(49) $< S$ from 0 to $*$ needs $\{\}$ from 1 to 2 $\{B\}$ from 4 to $* >$ (A+C+D))
(50) $< S$ from 0 to 4 needs $\{B\}$ from 1 to 2 $>$
(51) $< S$ from $*$ to 4 needs $\{B\}$ from $*$ to 0 $\{\}$ from 1 to 2 $>$
(52) $< S$ from 0 to 4 needs $\{\}$ from 1 to 2 $\{B\}$ from 3 to 3 $>$ (A+C+D)

Figure 1: Edges generated in EC phase of Kato's method

```
active_fund_rule(Searcher,ActEdges,InactEdges,NewActEdges,NewSearchers) :-
    Searcher = searcher(Hole,Err,[(Start,End,Cs)|_NeedList]),
    select(C,Cs,RestCs), /* select one category C from Cs */
    valid_order(C,RestCs), /* C can precede all categories in RestCs */
    /* check whether there exists a record of C in the range of Start to End */
    ( not_exist_in_record(C,Start,End) ->
      /* do the fundamental rule if there is no such a record */
      fundamental_rule(ActEdges,InactEdges,Start,End,NewActEdges),
      keep_record(C,Start,End)
    ; NewActEdges = ActEdges
    ),
    /* make completion between the searcher and active edges */
    mk_completion(Searcher,NewActEdges,NewSearchers).

top_down_rule(Searcher,NewSearchers) :-
    Searcher = searcher(Hole,Err,[(S,E,Cs)|NeedList]),
    select(C,Cs,RestCs), /* select one category C from Cs */
    valid_order(C,RestCs), /* C can precede all categories in RestCs */
    findall( searcher(NewHole,NewErr,NewNeeds)
             , ( id_rule(C,Rhs),
               NewNeeds = [(S,E1,Rhs),(S1,E,RestCs)|NeedList])
             , NewSearchers ).
```

Figure 2: Definition of search rules (1)

A main difference between search rules of our method and the ones of [6] is at the active fundamental rule. In EC phase, we completely avoid applying the fundamental rule because , in addition to what we need, it may generate many useless edges. Consequently, the works have to be done in this phase instead when there are requests by some searchers. Our active fundamental rule first checks the record whether the needed category in the specified range is ever requested before. If never, the fundamental rule will be invoked to make the completion between an active edge of category needed by the searcher and an inactive edge in the range covered by the start and the end positions of the searcher. The rule is applied until no new edge can be generated. The resultant edges are in turn matched with the searcher to further generate new searchers. Once all of these have been done in a specified range, the requested category and the range are kept on record. Next time when the category is requested by some searchers within the range again, we do not have to apply the rule anymore since all edges we

```
/* rule for handling extra word error */
garbage_rule(Searcher,NewSearcher) :-
    Searcher = searcher(Hole,Err,[(S,E,[])|NeedList]),!,
    NewErr is Err+E-S,
    NewSearcher = [searcher(Hole,NewErr,NeedList)].
garbage_rule(_,[]).

/* rule for handling omitted word error */
empty_category_rule(Searcher,NewSearcher) :-
    Searcher = searcher(Hole,Err,[(S,S,Cs)|NeedList]),!,
    length(Cs,LengthofCs),
    NewHole is Err-LengthofCs,
    NewErr is Err+LengthofCs,
    NewSearcher = [searcher(NewHole,NewErr,NeedList)].
empty_category_rule(_,[]).

/* rule for handling unknown word error */
unknown_word_rule(Searcher,NewSearcher) :-
    Searcher = searcher(Hole,Err,[(S,E,Cs)|NeedList]),
    select(C1,Cs,RestCs),
    valid_order(C1,RestCs),
    isa_lexical_category(C1),   /* C1 is a lexical category */
     /* no inactive edge of C1 at position S */
    C1 inactive_edge_is_not_at S,!,
    NewHole is Hole-1,
    NewErr is Err+1,
    S1 is S+1,
    NewSearcher = [searcher(NewHole,NewErr,[(S1,E,RestCs)|NeedList)].
unknown_word_rule(_,[]).
```

Figure 3: Definition of search rules (2)

need have been already provided.

The process of top-down search in our method is shown below. Although this example illustrates only handling of unknown word errors, other two kinds of errors can also be done in similar ways. TD phase of Kato's method can be done similarly, except that the application of the fundamental rule is not done.

start

(a) < hole: 1 err: 0 [S] from 0 to 4 >

apply active fundamental rule

from (4),(2) by fundamental rule

(b) < S from 0 to 3 found $\{A\}$ from 0 to 1 $\{C\}$ from 2 to 3
needs $\{B, D\}$ >

from (b),(3) by fundamental rule

(c) < S from 0 to 4 found $\{A\}$ from 0 to 1 $\{C\}$ from 2 to 3
$\{D\}$ from 3 to 4 needs $\{B\}$ >

make completion between (a) and (c)

(d) < hole: 1 err: 0 [B] from 1 to 2 >

apply unknown word rule to (d)

(e) < hole: 0 err: 1 [] >

Considering the total number of edges generated while parsing, it is apparent that our method generates less edges than Kato's one. Because parsing time varies to the number of edges, our method is more efficient in this case. As described earlier, however, the advantage of our method depends on the characteristic of languages. In general, the more flexible the language is, the more advantage our method will provide.

3.2 Handling word order errors

We have described how to cope with three kinds of errors in input by using ID/LP formalism. However, another important issue of handling word order errors remains to be discussed. In order to do this, we must generalize the form of an edge as follow.

< err: Err C from S_1 to E_n found C_1 from S_1 to E_1, ..., C_n from S_n to E_n
needs Cs >

of accumulated errors of an edge since we of word strings containing some word order errors. with the new format of edges defined above. we introduce another special rule called the *transposition rule* for handling word order errors in input, shown in Fig. 4. In our method, word order errors are detected by failure to make the completion between a pair of edges in the process of the fundamental rule due to the LP constraints violation. If the error is detected, the transposition rule would be triggered to form a new edge with an extra error, in addition to the err values of both edges, added to the err part of the resulting edge. Consequently, the transposition rule will not be applied wildly but used only when it is triggered by failure of the operations in the fundamental rule. This is to prevent overgeneration of order-violated merged

```
/* rule for handling word order error */
transposition_rule(ActEdge,InactEdge,NewEdge) :-
        ActEdge = edge(Err1,Cat,Start,End,Found,Need),
        InactEdge = edge(Err2,C1,S1,E1,Found1,[]),
        select(C1,Need,RestNeed),  /* remove C1 from Need list */
        /* insert C1 into Found list */
        insert((S1,E1,C1),Found,NewFound),
        Err3 is Err1+Err2+1,    /* add an extra error */
        NewEdge = edge(Err3,Cat,NewFound,RestNeed).
```

Figure 4: Definition of the transposition rule

edges. The resulting edges are then subsequently used by the fundamental rule to further generate new edges.

For example, consider how our method handles a word order error in the string *cbad*. Ignoring irrelevant edges, the parsing process looks as shown below. For simplicity, the err portions of edges that include no error are omitted. Since A must precede C as stated in the LP rule of G_1, the edge (h) cannot merge with the inactive edge of A. So, the transposition rule is employed to relax the constraint and form the edge (i). The edge is then used to continue searching for a complete parse.

EC phase:

(f) < S from 0 to 1 found{C}from 0 to 1 needs {A, B, D} >

TD phase:

the start searcher

(g) < hole: 1 err: 0 [S] from 0 to 4 >

active fundamental rule

from (f) and the inactive edge of B, by fundamental rule

(h) < S from 0 to 2 found {C} from 0 to 1 {B} from 1 to 2 needs {A, D}>

from (h) and the inactive edge of A, by transposition rule

(i) < err: 1 S from 0 to 3 found {C} from 0 to 1 {B} from 1 to 2
{A} from 2 to 3 needs {D} >

from (i) and the inactive edge of D, by fundamental rule

(j) < err: 1 S from 0 to 4 found {C} from 0 to 1 {B} from 1 to 2
{A} from 2 to 3 {D} from 3 to 4 needs {} >

match (g) and (j)

(l) < hole: 0 err: 1 [] >

4 Conclusion

We proposed a new approach of using ID/LP rules directly for ill-formed input parsing. We also showed that the approach we have taken can be better than previous CFG-based methods in parsing flexible word order languages. We believe that the approach can also work with reasonable performance for the languages which have more strict word order, though more experiments and analysis have to be done to verify this claim. The technique of applying the fundamental rule upon requests of top-down search may be used in the CFG-based methods as well to improve the parser's performance. A parser based on the method presented in this paper has been implemented and tested on a Thai grammar with 66 ID rules and 28 LP rules. Preliminary experiments show that our parser is more efficient than a CFG-based parser by a factor of 10 to 50. Nevertheless, we realize that more optimized parsers are needed for practical applications. We plan to test the parser with more variations of grammars and sentences to understand its behavior in general, and to exploit the knowledge to improve the parser's performance. The parallel implementation of the parser is another interesting topic that we would like to do in the near future.

References

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

[2] Jaime G. Carbonell and Philip J. Hayes. Recovery Strategies for Parsing Extragrammatical Language. *American Journal of Computational Linguistics*, Vol. 9, No. 3–4, pp. 123–146, 1983.

[3] Noam A. Chomsky. *Lectures on Government and Binding*. Foris Publications, 1981.

[4] Gerald Gazdar, Ewan Klein, Geoffrey K. Pullum, and Ivan Sag. *Generalized Phrase Structure Grammar*. Blackwell Publishing, 1985.

[5] Aravind K. Joshi. Tree Adjoining Grammars: How much context-sensitivity is required to provide reasonable structural descriptions? In David R. Dowty et al., editor, *Natural Language Parsing: Psychological, computational, and theoretical perspectives*, pp. 206–250. Cambridge University Press, 1985.

[6] Tsuneaki Kato. Yet Another Chart-Based Technique for Parsing Ill-Formed Input. Natural Language Processing 83–10, Information Processing Society of Japan, 1991.

[7] Surapant Meknavin. A Chart-based Method of ID/LP Parsing with Generalized Discrimination Network. In *Proceedings of the fifteenth International Conference on Computational Linguistics*, Vol. 1, pp. 401–407, 1992.

[8] Chris S. Mellish. Some Chart-Based Techniques for Parsing Ill-Formed Input. In *27th ACL*, pp. 102–109, 1989.

[9] Stuart M. Shieber. Direct Parsing of ID/LP Grammars. *Linguistics and Philosophy*, Vol. 7, pp. 135–154, 1984.

[10] Ralph M. Weischedel and Norman K. Sondheimer. Meta-rules as a Basis for Processing Ill-Formed Input. *American Journal of Computational Linguistics*, Vol. 9, No. 3–4, pp. 161–177, 1983.