Bidirectional Chart Generation of Natural Language Texts

Masahiko Haruno[◇*] Yasuharu Den[†] Yuji Matsumoto[‡] Makoto Nagao[◇]
[◇]Department of Electrical Engineering, Kyoto University
[†]ATR Interpreting Telecommunication Research Laboratories
[‡]Advanced Institute of Science and Technology, Nara
e-mail: haruno@kuee.kyoto-u.ac.jp

Abstract

This paper presents Bidirectional Chart Generation (BCG) algorithm as an uniform control mechanism for sentence generation and text planning. It is an extension of Semantic Head Driven Generation algorithm [Shieber *et al.*, 1989] in that recomputation of partial structures and backtracking are avoided by using a chart table. These properties enable to handle a large scale grammar including text planning and to implement the algorithm in parallel programming languages.

Other merits of the algorithm are to deal with multiple contexts and to keep every partial structure in the chart. It becomes easier for the generator to find a recovery strategy when user cannot understand the generated text.

Introduction

As opposed to traditional naive top-down or bottom-up mechanism [Wedekind, 1988][van Noord, 1989], the Semantic-Head-Driven (SHD) algorithm[Shieber *et al.*, 1989] combines both top-down and bottom-up derivations effectively. However, a straightforward implementation of the algorithm causes intensive backtracking when the scale of the grammar is large.

Bidirectional Chart Generation (BCG) algorithm avoids the inefficiency of backtracking by using a chart table. Like Chart Parsing algorithm[Kay, 1980], BCG algorithm can be implemented as a no-backtracking program in both parallel and sequential programming languages.

The algorithm is used in our explanation system not only for surface sentence generation but also for RST[Mann and Thompson, 1987] based text planning. As pointed out in [Moore and Paris, 1989], a generation facility must be able to determine what portion of text failed to achieve its purpose when follow-up question (user's feedback) arises. BCG algorithm deals with multiple contexts just like ATMS[de Kleer, 1986] and keeps every partial structure in a chart. It is easier for the generator to infer why the explanation fails and to find a recovery strategy.

After reviewing SHD algorithm, we present BCG algorithm comparing with Bottom-up Chart Parsing algorithm. Then, we show an implementation of the algorithm in a parallel logic programming language GHC[Ueda, 1986][1]. Finally, we discuss the application of BCG algorithm to answering user's follow-up questions in a RST based text planning.

Semantic-Head-Driven Algorithm

```
(1) s/Sem --> pp/ga(Sbj),pp/wo(Obj),
        #v(Sbj, Obj)/Sem.
(2) s/Sem -->pp/wo(Obj),pp/ga(Sbj),
        #v(Sbj, Obj)/Sem.
(3) pp/Sem --> np/NP,#p(NP)/Sem.
(4) v(Sbj, Obj)/call(Sbj,Obj) --> [呼ぶ].
(5) np/t --> [太郎].
(6) np/h --> [花子].
(7) p(NP)/ga(NP) --> [が].
(8) p(NP)/wo(NP) --> [を].
```

Figure 1: Sample Grammar

We give a brief outline of SHD algorithm based on the sample Japanese grammar shown in Figure 1. A nonterminal symbol is written in the form of category/semantics. semantic-head (marked by # in the grammar rules) has an important role in the algorithm.

semantic-head When the semantics of a right-hand-side element in a rule is identical to that of the left-hand-side, then the right-hand-side element is called the semantic head of the rule.

Grammar rules are divided into two types: Chain rules that have a semantic-head and non-chain rules[2] that do not. In the sample grammar, (1) through (3) are

*Current affiliation is NTT (Nippon Telegraph and Telephone) corporation.

[1]It is straightforward to transform it into a concurrent program in Prolog.

[2]we consider only lexical rules as non-chain rules for a while.

chain rules and (4) through (8) are non-chain rules. The algorithm proceeds bidirectionally, applying chain rules bottom-up and non-chain rules top-down. Those operations[3] are defined as follows:

Top-down operation A syntactic tree is traversed top-down using non-chain rules. A node that is about to expand is called the *goal*. Select a rule whose left-hand-side semantics is unifiable with that of the goal and make a node (called *pivot*) corresponding to the category of the left-hand-side. Then apply bottom-up operation from the pivot.

Bottom-up operation A syntactic tree is traversed bottom-up using chain rules. Select a rule whose semantic head is unifiable with the pivot, and then make other categories of the right-hand side as new goals. When all these goals are constituted applying operations recursively, the parent node at the left-hand side is introduced. If the parent node is not unifiable with the goal, then apply the bottom-up operation, regarding the parent node as a new pivot.

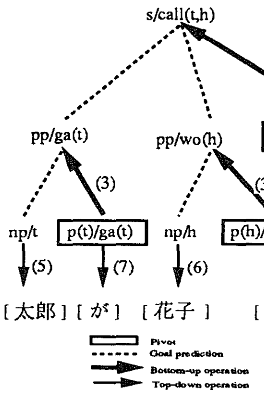

Figure 2: Generation Process

We show a sample generation process starting from semantic representation call(t,h) (Figure 2). First, a pivot v(t,h)/call(t,h) is introduced by applying top-down operation with rule(4). Two bottom-up operations using rules (1) and (2) are applicable to the pivot. Assume that the rule (1) is selected. The new goals pp/ga(t) and pp/wo(h) are introduced from the right-hand side of the rule. Top-down operation introduces new pivots p(t)/ga(t) and p(h)/wo(h) with rules (7) and (8). Going on the same process, a sentence [太郎, が, 花子, を, 呼ぶ] is generated as shown in Figure 2. Another sentence [花子, を, 太郎, が, 呼ぶ] is generated as well applying rule (2) by backtracking. This kind of backtracking causes serious inefficiency when the scale of grammar is large.

As discussed above, SHD algorithm consists of two parts, the top-down operation and the bottom-up operation. Because the bottom-up operation resembles the basic operation of left-corner parsing algorithm, considering the similarity between left-corner categories and semantic heads, SHD algorithm can be realized in the same way as Bottom-up Chart Parsing algorithm. In the next section, we present BCG algorithm, which avoids the inefficiency caused by backtracking.

[3]Top-down operation is augmented afterwards in order to handle general non-chain rules.

BCG Algorithm

Basic Algorithm

Bottom-up Chart Parsing algorithm [Kay, 1980] consists of the following three procedures.

Procedure-1: Let w_i be i-th word. For all rules of the form $b \rightarrow [w_i]$ create new inactive edges between v and w whose term is b provided that v and w are the $(i-1)$-th and i-th vertices.

Procedure-2: Let e_i be an inactive edge of category a incident from vertex v to vertex w. For all rules of the form $b \rightarrow c_1, c_2 \cdots c_n$ in the grammar such that $c_1 = a$, introduce a new edge e_a with the term $[a\ [?]c_2 \cdots [?]c_n]b$, incident from v to w, provided that there is no such edge in the chart already.

Procedure-3: Let e_a and e_i be adjacent active and inactive edges. e_a is incident from vertex v and e_i is incident to vertex w. Let $[?]\alpha$ be the first open box in e_a. If e_i is of category α, create a new edge between v and w whose term is that of e_a with the first open box replaced by the term of e_i.

Procedure-1 looks up lexical rules at the first stage of the algorithm. Procedure-2 predicts phrase structures by making use of the left-corner category. Procedure-3 fills up a prediction. On the other hand, SHD algorithm discussed in the previous section makes use of semantic head in order to predict new goals and the prediction is filled by recursive top-down operations. BCG algorithm is realized from Bottom-up Chart Parsing algorithm by identifying a semantic-head with a left-corner category. But important differences remain to be considered between generation and parsing as follows:

1. In parsing, all initial inactive edges are introduced at the first place by Procedure-1. This process corresponds to introducing pivots from semantic representation in the case of generation. This means that inactive edges must be built dynamically.
2. If Procedure-2 predicts two distinct goal sequences from one pivot by using two different rules, it happens that the pivot has two distinct adjacents because different goals may introduce different pivots.

The first point demands a dynamic process of introducing pivots. Once a goal is produced, its semantic representation is used to introduce a new pivot. The second point says that adjacent edges in BCG cannot be placed in a linear sequence. We introduce *forward links* to indicate the adjacency relation of edges; that is, when Procedure-1 introduces an inactive edge e_i according to an active edge e_a, it puts a pointer from the tail of e_a to the head of e_i. Two edges are adjacent in generation if there exists a forward link from one to the other. In addition, we must take account of the case where the required pivot has already been introduced before. In such a case, we reuse the previously produced pivot by simply adding a new forward link going to it. Therefore, it occurs that more than one forward link is put to one edge.

BCG algorithm becomes as follows. Procedure-1 realizes the dynamic introduction of pivots. Procedure-2 and Procedure-3 are straightforward augmentations of the bottom-up chart parsing algorithm except for the use of forward links.

Procedure-1: Let e_a be an active edge of category $[a_1 \cdots [?]c_j \cdots [?]c_n]b$ incident from vertex u to v. Let $[?]c_j$ be the first open box in e_a and Sem_j be its semantics. For all rules of the form `b/Sem -->` `[word]` such that Sem and Sem_j are unifiable, create new inactive edges between vertex w and vertex w' whose term is b/Sem and put a forward link from vertex v to vertex w. If the same inactive edge ever exists from vertex x to vertex y put a forward link from vertex v to vertex x instead.

Procedure-2: Let e_i be an inactive edge of category a incident from vertex v to vertex w. For all rules of the form `b --> c1, ..., #ch,...,cn` in the grammar such that Sem_h and Sem are unifiable, introduce a new active edge e_a with the term $[[?]c_1 \cdots a \cdots [?]c_n]b$, incident from v to w, provided that there is no such edge in the chart already. Sem and Sem_h are semantics of a and c_h.

Procedure-3: Let e_a be an active edge with the term $[a_1 \cdots [?]c_j \cdots [?]c_n]b$ incident from vertex u to vertex v and e_i be an inactive edge with the term a incident from vertex w to vertex x. Let $[?]c_j$ be the first open box in e_a. If a forward link exists from vertex v to vertex w such that $c_j = a$, create a new edge between u and x whose term is $[a_1 \cdots a\ [?]c_{j+1} \cdots [?]c_n]b$.

An example starting from semantic representation call(t,h) is explained in the rest of this section.
The chart constructed in the process is shown in Figure 3 and in Table 1. The first inactive edge `v(t,h)/call(t,h)` is introduced from rule(4) by Procedure-1 and the process proceeds as shown in Figure 3. The inactive edge4 `p(t)/ga(t)` is produced from the goal `pp/ga(t)` of active edge2 and rule(7) by Procedure-1. Then the forward link A is put from the tail of active edge2 to the head of inactive edge4. Inactive edge5 `p(h)/wo(h)` is produced in the same way from active edge3. After the inactive edge10 `pp(t)/ga(t)` and 11 `pp(h)/wo(h)` are generated, which have the same head as inactive edge4 and 5, active edge12 [`pp/ga(t)` [?]`pp/wo(h)` `v(t,h)/call(t,h)`] `s/call(t,h)` is introduced from active edge2 and inactive edge10 by Procedure-3. Although inactive edge `p(h)/wo(h)` is introduced from goal `pp/wo(h)` of active edge12 and rule(8) by Procedure-1, it is the same as inactive edge5. Then forward link E is put from the tail of active edge12 to the head of inactive edge5 instead of generating a new inactive edge. At the end of the process, inactive edges 14 and 15 are produced, each of which corresponds to a sentence[4]. They are generated with no backtracking.

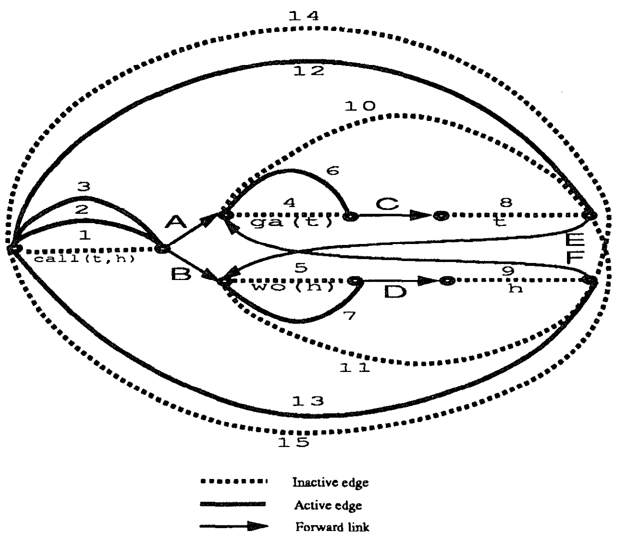

Figure 3: Graph Representation of the Chart

General Non-Chain Rule

We show in this section how general non-chain rules are handled in BCG algorithm, though we have considered only lexical rules as non-chain rules. General non-chain rules are necessary for handling a large scale grammar, particularly for text planning. Consider the following non-chain rule which describes a Japanese relative clause:

```
np/ind(X,[R|Rstr]) -->
        s_rel(X)/R,
        np/ind(X,Rstr).
```

First, we extend the top-down operation defined before:

top-down operation A syntactic tree is traversed top-down using non-chain rules. A node that is about to expand is called the *goal*. Select a non-chain rule whose left-hand-side semantic representation is unifiable with that of the goal and make a node called *pivot* corresponding to the category

[4]Note that the order of edges in the chart doesn't mean the surface word order. It is shown explicitly by difference lists as discussed in the next section.

Edge	Term	Procedure	Rule
1	v(t,h)/call(t,h)	1	(4)
2	[[?]pp/ga(t),[?]pp/wo(h),v(t,h)/call(t,h)]s/call(t,h)	2	(1)
3	[[?]pp/wo(h),[?]pp/ga(t),v(t,h)/call(t,h)]s/call(t,h)	2	(2)
4	p(t)/ga(t)	1	(7)
5	p(h)/wo(h)	1	(8)
6	[[?]np/t,p(t)/ga(t)]pp/ga(t)	2	(3)
7	[[?]np/h,p(h)/wo(h)]pp/wo(h)	2	(3)
8	np/t	1	(5)
9	np/h	1	(6)
10	pp/ga(t)	3	-
11	pp/wo(h)	3	-
12	[pp/ga(t),[?]pp/wo(h),v(t,h)/call(t,h)]s/call(t,h)	3	-
13	[pp/wo(h),[?]pp/ga(t),v(t,h)/call(t,h)]s/call(t,h)	3	-
14	s/call(t,h)	3	-
15	s/call(t,h)	3	-

Table 1: Table Representation of the Chart

of the left-hand-side. **In addition, make categories of right-hand side as new goals and apply top-down operation to them recursively.** If the pivot is not unifiable with the goal, then apply bottom-up operation from the pivot.

The bold-face part is supplementary to the original top-down operation. It expands the categories at right-hand side after unifying the goal with left-hand side. Note that this part is almost same as top-down derivation of a syntactic tree. The procedure for the general non-chain rules is formalized in the same way as Top-down Chart Parsing algorithm [Kay, 1980]. The definition of the operation is the following Procedure-1'.

Procedure-1' Let e_a be an active edge with the term $[a_1 \cdots [?]c_j \cdots c_n]d$ incident from vertex u to v. Let $[?]c_j$ be the first open box in e_a and Sem_j be its semantic representation. For every rule of the form b/Sem --> $c_1, \ldots, c_n$ such that $Sem_j andSem$ are unifiable, create a new active edge with the term $[[?]c_1 \cdots [?]c_n]b$ looping at vertex w, and put a forward link from v to w. If the same inactive edge ever exists from y, simply put a forward link from v to y instead.

Implementation

Previous sections show that BCG algorithm is formalized in the similar way to Chart Parsing algorithm. PAX parsing system[Matsumoto, 1986] is an implementation of Bottom-up Chart Parsing algorithm in a parallel logic programming language GHC[5]. We show in this section a GHC implementation of BCG algorithm in the similar way to PAX system. The implemented system consists of the following two parts.

1. The program translated from grammar rules.

[5]A GHC clause can be understood just like a Prolog clause if the commit operator '|' is replaced by '!'.

2. The meta-process that introduces inactive edges dynamically. It absorbs the difference between parsing and generation.

Basic Transformation of Grammar Rules

In our implementation, each terminal and non-terminal symbol is realized as a parallel process that communicate with each other for building up larger structures. The communication channel is called a stream. Let us take the following grammar rule.

```
s/Sem -->
        pp/ga(Sbj),
        pp/wo(Obj),
       #v(Sbj,Obj)/Sem.
```

Three non-terminal symbols at right-hand-side are realized as parallel processes and each of them receives a stream from the left and passes an output stream to the right. For transformation, the following modification is done to the grammar rule: Identifiers standing for intemediate positions in a grammar rule are inserted in the rule and the semantic head of the rule is moved to the top of the right-hand-side to be associated with left-corner parsing. Moreover, in order to keep the surface order information, difference lists representing words are added to each symbol. The example rule results in the following form:

```
s(S0-S3)/Sem -->
        v(Sbj, Obj, S2-S3)/Sem,
        id1,
        pp(S0-S1)/ga(Sbj),
        id2,
        pp(S1-S2)/wo(Obj).
```

By translating this rule into following GHC clauses, we can achieve SHD generation in parallel. The behavior of the grammar rule is depicted in Figure 4.
First, `v(Sbj,Obj,S2-S3)` is translated into the program below.

```
v(In,Sbj,Obj,S2-S3,Out) :- true |
    Out = [id1(ga(Sbj),In,Sbj,Obj,S2-S3)].
```

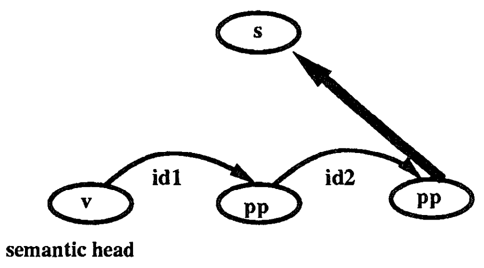

Figure 4: Behavior of Processes

When v(Sbj,Obj,S2-S3) is produced, tree traverse proceeds to the position of id1. It corresponds to the Procedure-2 of BCG algorithm that selects a rule with a semantic head whose semantic representation is unifiable with that of the inactive edge (pivot), and introduces a new active edge (goal). The process v(In,Sbj,Obj,S2-S3,Out) generates id1, which corresponds to the active edge. In general, processes perform inactive edges and data in streams stands for active edges. The first open box of the new active edge is pp(S0-S1), whose semantics ga(Sbj) is passed along with id1 and used afterwards in the meta-process (Procedure-1 of BCG algorithm).
Secondly, pp(S0-S1) is translated as below.

```
pp([id1(_,In,Sbj,Obj,S2-S3)|Tail],S0-S1,Out) :-
   true |
   Out = [id2(wo(Obj),In,Obj,S0-S1,S2-S3)|Out1],
   pp(Tail,S0-S1,Out1).
```

Because pp(S0-S1) is to the right of id1, tree traverse proceeds to the position of id2 when pp(S0-S1) receives id1. This corresponds to the Procedure-3 of BCG algorithm that derives a new active edge from an active edge and an inactive edge. The first open box of the new active edge is pp(S1-S2), the semantics of which is wo(Obj) is inserted as the first argument of id2. In the same way, pp(S1-S2) is translated as below.

```
pp([id2(_,In,Obj,S0-S1,S2-S3)|Tail],S1-S2,Out) :-
   true | s(In, S0-S3,Out1),
   pp(Tail,S1-S2,Out2),merge(Out1,Out2,Out).
```

When pp(S1-S2)/wo(Obj) is generated, the parent node s(S0-S3)/Sem is generated. The final definition of process pp is the collection of all of such clauses each of which corresponds to an occurrence of pp in the right-hand-side of grammar rules. The following clauses are necessary to handle exceptional situations:

```
pp([],_,Out) :- true | Out = [].
pp([_|Tail],String,Out):- otherwise |
    pp(Tail,String,Out).
```

Finally, let us take the following non-chain (lexical) rule.

```
v(Sbj, Obj)/call(Sbj,Obj) --> [呼ぶ].
```

This rule is translated into the program below, which generates a process corresponding to v(Sbj,Obj) from the semantic representation call(Sbj,Obj).

```
pivot(call(Sbj,Obj),In,Out) :- true |
    v(In,Sbj,Obj,[呼ぶ|S0]-S0,Out).
```

The pivot process is generated dynamically by the meta-process corresponding to the Procedure-1 of BCG algorithm.

Meta-Process

The meta-process monitors the data in all streams and controls the whole generation process. It checks the semantic representation in each identifier (semantics(Id,Sem)), and generates or reuses an inactive edges according to the semantic representation, then passes the identifier to the inactive edges. It is attained by calling the pivot process described below. Here, streams perform forward links of BCG algorithm. Forward links are introduced dynamically and a stream is realized by an open list to receive identifiers incrementally. The meta-process maintains the table that consists of pairs like wait(Sem,Str), where Sem is the semantic representation of an ever produced inactive edge and Str is the tail of its input stream. When the meta-process derives Sem_j from an identifier and is about to produce a pivot process, it checks whether Sem_j is already registered in the table. The meta-process generates a new pivot process only if the pair wait(Sem_j,Str) is not registered. The following program realizes the task.

```
meta_proc([],_) :- true | true.
meta_proc([Id|Tail],Table) :-
   semantics(Id,Sem),
   get(wait(Sem,StrTail),Table,Table1) |
   StrTail = [Id|NewStrTail],
   put(wait(Sem,NewStrTail),Table1,NewTable),
   meta_proc(Tail,NewTable).
meta_proc([Id|Tail],Table) :- otherwise |
   semantics(Id,Sem),
   pivot(Sem,[Id|StrTail],Out),
   put(wait(Sem,StrTail),Table,NewTable),
   merge(Out,Tail,Next),
   meta_proc(Next,NewTable).
```

The second clause of meta_proc corresponds to the case of reusing the existing process and the third to the case of generating a new process. In the second clause, get(wait(Sem,StrTail),Table,Table1) looks up if wait(Sem,StrTail) is previously registered in the table. When the table includes the element, meta_proc reuses it by instantiating the top of the open list with StrTail. Otherwise meta_proc generates a new process by calling pivot(Sem,[Id|StrTail],Out) in the third clause, and register the process in the table by put(wait(Sem,StrTail),Table,NewTable).

The pivot process introducing new processes is derived by transforming lexical (non-chain) rules as described in the previous section. The transformation of general non-chain rules are described in the next section.

Transformation of Non-Chain Rules

Let us consider the following rule.

```
np/ind(X,[R|Rstr]) -->
   s_rel(X)/R,
   np/ind(X,Rstr).
```

General non-chain rules are treated by Procedure-1' whose central part is the same as Procedure-1 . The only difference is that Procedure-1' introduces a new active edge, from a semantic representation of a predicted goal. The process is also realized by the pivot process as below:

```
pivot(ind(X,[R|Rstr]),In,Out) :- true |
   Out = [id3(R,In,X,R,Rstr)].
```

The identifier **id3** is inserted just before the leftmost category **s_rel(X)** for top-down traversal of a syntactic tree. The **pivot** process corresponding to the semantic representation **ind(X,[R|Rstr])** generates this identifier. This kind of identifier corresponds to the active edge of Top-down Chart Parsing algorithm. When all categories at right-hand side are constituted, then a new process corresponding to **np** at left-hand side is produced.

Applying BCG Algorithm to RST Based Text Planning

This section examines the applicability of BCG algorithm to text planning. The depth-first search strategy has been used mainly in text planning, in which it is difficult for a generator to select the relevant operator at every choice point. On the other hand, BCG algorithm deals with more than one candidate in parallel until enough information is obtained.

Moreover, in explanation dialogue systems, users often ask follow-up questions when he or she cannot fully understand the explanation. The generator must infer why its explanation has failed to achieve the communicative goal; an error in user model, ambiguity of the meaning and so on. In BCG algorithm, it is easier for a generator to find a recovering strategy because all partial structures are preserved.

Plan Language

Our plan language is based on Rhetorical Structure Theory(RST)[Mann and Thompson, 1987]. Explanation dialogue requests a plan language to express both intentional and rhetorical structures of the text once produced to answer follow-up questions. We adopt the similar representation of RST to Moore's operators [Moore and Paris, 1989], one of which is shown below.

```
EFFECT:(BMB S H ?x)
CONSTRAINTS:nil
NUCLEUS:(INFORM S H ?x)
SATELLITES:(PERSUADE S H ?x)
```

In order to apply BCG algorithm to text planning, such operators are represented in DCG rules, where CONSTRAINTS are inserted as extra conditions. (1) corresponds to the above example.

```
(1) bmb/bmb(Speaker,Hearer,X) -->
       inf/inform(Speaker,Hearer,X),
       psd/persuade(Speaker,Hearer,X).
(2) bmb/bmb(Speaker,Hearer,X) -->
       explain/explain(Speaker,Hearer,X),
       inf/inform(Speaker,Hearer,X).
```

Let the speaker's goal be **bmb(Speaker,Hearer,X)**, there are alternative rules (1) and (2) applicable to this situation. A naive top-down planner recomputes **inf/inform(Speaker,Hearer,X)** due to backtracking. On the other hand, BCG algorithm proceeds in parallel reusing the structures ever constructed. Because most rules are applied top-down in text planning, the behavior of BCG algorithm, in this case, is almost identical to that of Top-down Chart algorithm [Kay, 1980].

Answering Follow-up Questions

BCG algorithm can select the best recovering strategy by comparing multiple contexts when receiving follow-up questions. Suppose user model contains concepts that the user does not actually know, the generator must change the user model and select the proper strategy for explaining the concept. The generator employs partial information in the chart particularly incomplete active edges, which stand for suspended plans. Let us consider the following plan operators and general knowledge.

```
          % plan operator
goal/goal(Hearer,do(Hearer,Act)) -->
 recommend/recommend(Speaker,Hearer,Act),
 psd/persuaded(Hearer,goal(Hearer,do(Hearer,Act))).
psd/persuaded(Hearer,goal(Hearer,do(Hearer,Act)))
 --> {step(Act,Goal)},
 motivation/motivation(Act,Goal).
psd/persuaded(Hearer,goal(Hearer,do(Hearer,Act)))
 --> {step(Act,Goal),
 bel(Hearer,benefit(Act,Hearer))},
 inf/inform(benefit(Act,Hearer)).
motivation/motivation(Act,Goal) -->
 {step(Act,Goal)}
 bel/bel(Hearer,step(Act,Goal)).
bel/bel(Hearer,step(Act,Goal)) -->
 {know(Hearer,Goal)},
 inf/inform(Speaker,Hearer,step(Act,Goal)).
bel/bel(Hearer,step(Act,Goal)) -->
 inf/inform(Speaker,Hearer,step(Act,Goal)),
 {\+ know(Hearer,Goal)},
 elaboration/elaboration(Goal).

          % general knowledge
step(insert,optimization).
know(user,optimization).
```

The domain of the dialogue is Prolog programming. The system's goal is **goal(user(do(user,insert))**, to recommend the user to insert a '!' before a recursive call. **insert** means inserting ! before the recursive call and **optimization** means the tail recursion optimization. The system generates the following texts as the first text (Figure 5).

system: 再帰呼び出しの前にカットを入れて下さい。

```
        末尾最適化に必要です。
        (Insert a cut symbol before recursive call.
         It is necessary for tail recursive optimi
         zation.)
```

The user cannot understand the explanation and poses the follow-up question.

```
user: 良く分かりません。
      (I don't understand very well.)
```

The system accepts the follow-up question and searches the suspended active edges. Now, there are two suspended active edges (1) and (2):

```
(1)[[?]inf/inform(benefit(insert,user))]
      psd/persuaded(user,goal(user,do(user,insert)))
(2)[inf/inform(system,user,step(insert,optimization)),
      [?]elaboration/elaboration(optimization)]
        bel/bel(user,step(insert,optimization)
```

Each of the edges is suspended because of the contradiction to the user model; `bel(user,benefit(insert,user))` and \+ `know(user,insert)`. The generator selects the active edges that require few hypotheses to expand, and gives the relaxation to the user model. In this case, the generator assumes that \+ `know(user,insert)` holds and produces the following additional explanation as shown in Figure 5.

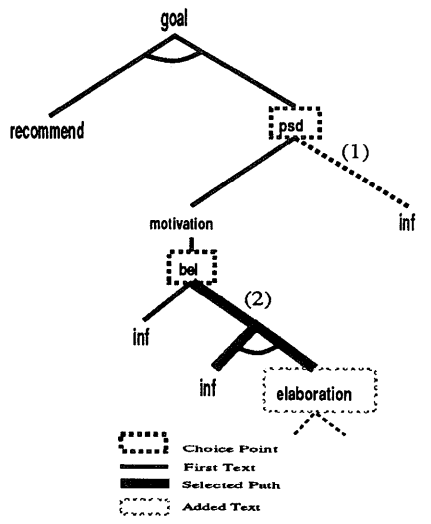

Figure 5: The Explanation Tree

```
system: 末尾最適化はコンパイラにバックトラックが必要無
        いことを示し、記憶容量を節約する手法です。
        (Tail recursive optimization saves memory
         by showing compilers that no backtracking
         is necessary.)
```

The generator can reproduce the explanation by relaxing the user model according to actual state of user's knowledge. This is the similar situation to relaxation based parsing of ill-formed inputs in which chart-based method is powerful[Mellish, 1989] because it maintains all partial structures.

Concluding Remarks

We have presented BCG algorithm as a basic control mechanism of generation system. In contrast to Shieber's SHD algorithm, BCG algorithm deals with multiple contexts at a time. This property resolves two problems: First, the efficiency is remarkably improved in the case of a large scale grammar. Secondly, the comparison between multiple contexts becomes possible. Hence, revision like answering follow-up questions is performed easier by referring to the contexts in the chart. To sum up, we obtain the efficiency and robustness by adopting the BCG algorithm.

We are now studying the patterns of the follow-up questions and investigating the recovery heuristics based on the chart.

References

Johan de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28:127–162, 1986.

Martin Kay. Algorithm schemata and data structure in syntactic processing. Technical Report CLS-80-12, Xerox PARC, 1980.

W. C. Mann and S. A. Thompson. Rhetorical structure theory: Description and construction of text structures. In *Natural Language Generation*, chapter 7, pages 85–96. Martinus Nijhoff Publishers, 1987.

Yuji Matsumoto. A parallel parsing system for natural language analysis. In *Proc. 3rd ICLP*, Lecture Notes in Computer Science 225, pages 396–409. Springer-Verlag, 1986.

Chris Mellish. Some chart-based techniques for parsing ill-formed input. In *Proc. 27th ACL*, pages 102–109, 1989.

Johanna Moore and Cecile Paris. Planning text for advisory dialogues. In *Proc. 27th ACL*, pages 203–211, 1989.

S. M. Shieber, van Noord, R. C. Moore, and F. C. N. Pereira. A semantic-head-driven generation algorithm for unification-based formalisms. In *Proc. 27th ACL*, pages 7–17, 1989.

K Ueda. Guarded Horn Clauses. In E. Wada, editor, *Logic Programming '85*, Lecture Notes in Computer Science 221, pages 168–179. Springer-Verlag, 1986.

van Noord. BUG: A directed bottom-up generator for unification-based formalisms. Working Paper 4, Katholieke Universiteit Leuven Stiching Taaltechnologie Utrecht, the Netherlands, 1989.

J Wedekind. Generation as structure driven derivation. In *Proc. 11th COLING*, pages 732–737, 1988.