A Bottom-up Generation for Principle-based Grammars Using Constraint Propagation

Masato Ishizaki
NTT Communications and Information Processing Laboratories
1-2356 Take Yokosuka-shi Kanagawa-ken, 238-03 Japan
E-mail: ishizaki%nttnly.ntt.jp@relay.cs.net

Abstract

A bottom-up generation algorithm for principle-based grammars is proposed. Bottom-up generation has (1) an inefficiency because of a high degree of nondeterminism, (2) a limitation caused by inability to process logical forms produced by grammar rules, and (3) an identity semantic problem. This paper describes a solution to these problems and implementation issues for the algorithm using a constraint logic programming language.

1 Introduction

The generation of strings from logical forms was studied intensively by [Shi88,89][Cal89]. The study of this problem shed light not only on the efficiency and soundness of generation algorithms, but also on the descriptive appropriateness of grammar itself.

A generation algorithm based on the Early's method has proposed, which is capable of analyzing and generating sentences in a uniform architecture [Shi88]. In this architecture the criterion of "semantic monotonicity" is assumed to reduce fruitless generation. As Shieber has mentioned, this method is still inefficient and limited, and ,instead, has shown an efficient and general top-down generation algorithm using the semantic-head concept[1] [Shi89].

An algorithm for Categorial Unification Grammar(CUG) is mentioned in [Cal89]. As to the important role of lexicon, this grammar shares the same property as principle-based grammars, but differs in that CUG has grammar rules in the lexicon(This enables top-down generation with prediction as discussed in [Shi89]). The identity semantic problem and the lexical indexing problem are some of the same problem noted in principle-based grammars.

Generation based on Lexical Functional Grammar(LFG) is formalized in [Wed88]. He defined the derivability and the generability of f-structure-to-string and semantic-structure-to-string mapping. These concepts seem to be suitable for top-down strategies. His idea is intriguing in that discourse information, such as *topic*, can be reflected on an output string. Bottom-up generation also has the possibility of incorporating this information. This issue will be discussed later.

I propose a bottom-up generation algorithm for principle-based grammars, which makes use of the same grammar as a parser. Bottom-up generation is inefficient due to a high degree of nondeterminism. It is limited by its inability to process semantic expressions created by grammar rules. It also has an identity semantic problem. This paper describes the solution to these problems and the issues concerning implementation of the algorithm using a constraint logic programming language, cu-PROLOG.

First, the problems of bottom-up generation are shown. Next the generation algorithm using constraint propagation and solutions to these problems are considered. Third, some issues concerning implementation and an example of this algorithm are mentioned. Finally, the problem of logical form equivalence, lexical indexing, and controlled search using the discourse information are discussed.

2 Problems of Principle-based Grammar Generation

Principle-based grammars like Head-driven Phrase Structure Grammar(HPSG) or Japanese Phrase Structure Grammar(JPSG) seem to be inadequate for top-down operations. The reasons for this are;
(1) as principle-based grammars have few skeleton rules, top-down operations that are assumed to use rules are not adequate for them,
(2) much information is in lexical items instead of grammar rules in principle-based grammars, and
(3) the construction of semantic expressions is intuitively adequate for bottom-up operations[2].
Therefore, an efficient bottom-up generator for principle-based grammars must be developed.

Bottom-up generation has two advantages; it can use lexical information at an early phase, and can

[1] Precisely his algorithm takes a mixture of top-down and bottom-up strategy. But generation is fundamentally driven in a top-down way.

[2] Unification can be used for bi-directional operations. However, if you need destructive semantic operations('destructive' means that a resultant semantic expressions cannot be constructed from base semantic expressions without these special functions), unification cannot be used for these operations.

avoid the left recursion because of the property similar to bottom-up parsing. However, bottom-up generation has three problems[Shi88,89][Cal89];
(1) inefficiency because of a high degree of nondeterminism,
(2) limitation which is caused by inability to process semantic expressions created by grammar rules, and
(3) vestigial[Shi88], or identity semantic problem[Cal89] (hereafter called identity semantic problem).
(2) and (3) relate to completeness and coherence problems[Wed88].

Inefficiency because of a high degree of nondeterminism means that a naive bottom-up generation algorithm cannot use semantic information properly, and many semantic-irrelevant subexpressions that are syntactically correct will result.

One of the reasons that Shieber adopted a top-down strategy is that he insisted on the existence of logical forms produced by grammar rules. This assumption means that sub-semantic expressions cannot be derived from resultant semantic ones that are different from normal ones. It is shown later that if we permit this kind of destructive semantic operation, we cannot obtain an efficient algorithm.

An identity semantic problem is common among generation algorithms using lexical-based grammars. The semantic expressions are classified into substantial elements that contribute to the whole semantic expressions, and functional elements(these are identity semantic expressions) without influencing the whole ones. Examples of the functional elements are complementisers in English, or case-marking postposition in Japanese. Simply speaking, the solution is to introduce these items at some time, but this also produces high inefficiency.

3 Principle-based Grammar Generation

3.1 Generation Algorithm Using Constraint Propagation

Natural language processing, such as sentence comprehension and production, is thought of as constraint satisfaction problems[Has86]. Constraint propagation techniques are very effective in these problems[Din86]. Constraint propagation techniques are classified into two methodologies: active constraint, which transforms constraints into more efficient ones, and passive constraint, which is realized by the function such as *freeze* in Metalog. Passive constraint is similar to data-driven control, so if data does not arrive, the constraints are unsolved. As active constraint solving transforms constraints independent of data, it does not incur such a problem.

Hasida has shown that sentence analysis and synthesis can be described following a simple program with a constraint, *constituent* [3].

struct(*Category*,X,Y); constituent(*Category*,X,Y).

If active constraint solving techniques are applied to the problem, can this program be executed efficiently? Active constraint solving is equivalent to fold/unfold transformation [Tud89]. If the constraint clause is simply unfolded, then the number of clauses created will be the same as the number of lexical items. Presently this is not an efficient way[4].

If passive constraint solving techniques are used, then how is data obtained? The answer is to predict the base lexical item, which is the core of a sentence, using a top-down prediction analogous to a bottom-up parsing technique.

```
gen(cat(P,F,Aa,Au,Sc,Sem)) :-
  pred(cat(P,F,Aa,Au,Sc,Sem),Sem,BaseLex),
  gen1(Sem,BaseLex,NewBaseLex,
                    cat(P,F,Aa,Au,Sc,Sem)).

gen1(Sem,BaseLex,NewBaseLex,Cat) :-
  verify(NewBaseLex,Cat).
gen1(Sem,BaseLex,NewBaseLex,Cat) :-
  getLex(Sem,BaseLex,BaseLex1),
  gen1(Sem,BaseLex1,NewBaseLex).

getLex(Sem,BaseLex,NewBaseLex) :-
  getLex1(Sem,Lex);
    psr(Lex,BaseLex,NewBaseLex).
getLex(Sem,BaseLex,NewBaseLex) :-
  getLex1(Sem,Lex), introduceFLex(FLex);
    psr(Lex,FLex,Lex1),
    psr(Lex1,BaseLex,NewBaseLex).
```

Figure 1: The generation algorithm.

This generation algorithm is sketched by the cu-PROLOG, or prologIII notation in figure 1. The predicate *gen* produces a sentence string from the term *cat(P,F,Aa,Au,Sc,Sem)*[5]. The term *cat* represents a set of features: *P* is the feature for part-of-speech, *F* is form such as verb inflection, *Aa* is adjacent node specification, *Au* is adjunct node specification, *Sc* is subcategorization information, and *Sem* is semantic information. The predicate *pred* anticipates *BaseLex* that is the core of a sentence(normally head verb), using part-of-speech and semantic information. *BaseLex* has lexical and feature information. The predicate *gen1* gets a lexical item, and applies principles to the item and the base item until producing a sentence. *getLex* extracts a lexical item that is

[3] This is rather misleading. An efficiency problem of generation and theoretical consideration about the problem are also mentioned in [Has86].

[4] If other techniques are developed, this consideration is not always correct.

[5] Although this top predicate is a recognizer, constructing a tree or a string is easily realized with little effort.

constrained by principles[6] using semantic information. *introduceFLex* extracts an identity semantic item that is constrained by principles.

3.2 Counterarguments to Bottom-up Generation Deficiency

3.2.1 Inefficiency Due to High Degree of Nondeterminism

Constraint propagation techniques in the previous section remedies nondeterminism of a bottom-up generation problem. An example of nondeterminism is the noun phrase generation in [Shi89]. If a NP occurs before a verb, different case NPs will be generated nondeterministically in figure 2.

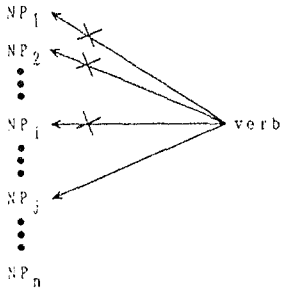

Figure 2: Nondeterminism in NP generation.

psr(Daughter,Head,Mother);
member(Daughter,Head.subcategorization).

psr represents a Mother → Daughter Head rule. The predicate *member* is used as a constraint, which says the Daughter node is a member of a subcategorization of the Head node. The constraint propagation process is shown in figure 3.

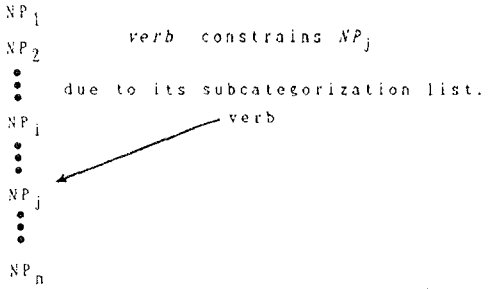

Figure 3: Reduction of nondeterminism in NP generation.

Constraints can also eliminate the irrelevant phonological expressions shown in [Cal89], and compactly express the order-variant subcategorization list in JPSG.

3.2.2 Limitation Owing to Special Semantic Expressions

Suppose a Mother → Daughter Head rule, where

Mother.sem = f(Daughter.sem, Head.sem)

and semantic function f destructively creates Mother.sem from Daughter.sem and Head.sem. That is, Daughter.sem and Head.sem cannot be predicted from Mother.sem. To get the value of Daughter.sem or Head.sem from Mother.sem, the inverse function f^{-1} is needed. The implicit assumption of inversability of the function[Shi89] is very severe, and a rather tricky feature structure must be constructed to escape the completeness problem. Therefore, it seems better to use another function in a semantic-monotonous framework instead of this one.

However, we can easily modify the predicate *predict* to get another head using the semantic information to which the function is applied, if this inverse function is obtained.

3.2.3 An Identity Semantic Problem

When using constraints to access functional lexical items, an exhaustive search is not required. The insertion of Non-null constituents, such as case markers and functional nouns, can be restricted using various constraints (syntactic, semantic information). For example, case markers that indicate the relationships between verbs and nouns are demanded by subcategorization information of the verbs. By using the constraint solving techniques, efficiency can be improved equal to or more than that of the top-down algorithm. Of course since such occurrences(e.g. null constituents, or gap) cannot be restrained, this convenient mechanism cannot be used. However, this situation is the same as top-down algorithms.

4 Implementation

4.1 Grammar Formalism and Implementation Language

Our algorithm exploits JPSG as phrase structure grammar formalism. The concept of JPSG inherits the fundamental property of HPSG. That is, JPSG makes use of a set of feature-value pairs, feature constraints and unification to stipulate Japanese grammar instead of rewriting rules for terminal and nonterminal symbols. Subcategorization information is stored in lexical entries, instead of being stored in grammar rules.

Logical forms are expressed by the semantic representation language proposed by [Hob85]. The distinctive feature of this language is its simplicity of the form for discourse processing. The reason for this simple form is; (1)a conjunction of atomic predicates, (2)all variables are existentially quantified with the widest possible scope, and (3)there is no functions, functionals, nested quantifiers, disjunctions, negations, or modal or intensional operators.

The algorithm is implemented using the cu-PROLOG developed in ICOT[Tud89]. The main fea-

[6] A few phrase structure rules that observe principles, such as the head feature principle, are used in the program.

ture of this language is to adopt constraint unification instead of normal unification. This gives it more descriptive power and more declarativeness than normal prologs. The clause of cu-PROLOG is represented as;

$$h \leftarrow b1,b2,...,bn;c1,c2,...,cn$$

where h is a head, b1,b2,...,bn bodies, and c1,c2,...,cn constraints.

```
psr(cat(P, F, [], [], Sc, Sem),
    cat(Pos, Form, [cat(P, F, [], [], Sc, Sem)], Au, SubCat, SEM),
    cat(Pos, Form, [], Au, SubCat, SEM),[adjacent_p]).

psr(cat(P, F, [], Au, Sc, Sem),
    cat(Pos, Form, [], Adj, [cat(P, F, [], Au, Sc, Sem)|Rest], SEM),
    cat(Pos, Form, [], Adj, Rest, SEM),[subcat_p] );sc_cond(P,Pos).
sc_cond(p,v).
sc_cond(adn,n).
sc_cond(n,p).

dict(shonen,cat(n,n,[],[],[],[boy,X])).
dict(tama,cat(n,n,[],[],[],[ball,X])).
dict(ker,
     cat(v,vcr,[],[],S,[[kick,E,Sbj,Obj],[R1,Sbj],[R2,Obj]]));
     pp_wo_ga(S,Obj,Sbj,R2,R1).
dict(ga,cat(p,ga,cat(n,F,Aa,Au,Sc,Sem),[],[],Sem)).
dict(wo,cat(p,wo,cat(n,F,Aa,Au,Sc,Sem),[],[],Sem)).
```

This grammar is shared with the parser mentioned in [Tud89]

Figure 4: Part of grammar rules and a lexicon.

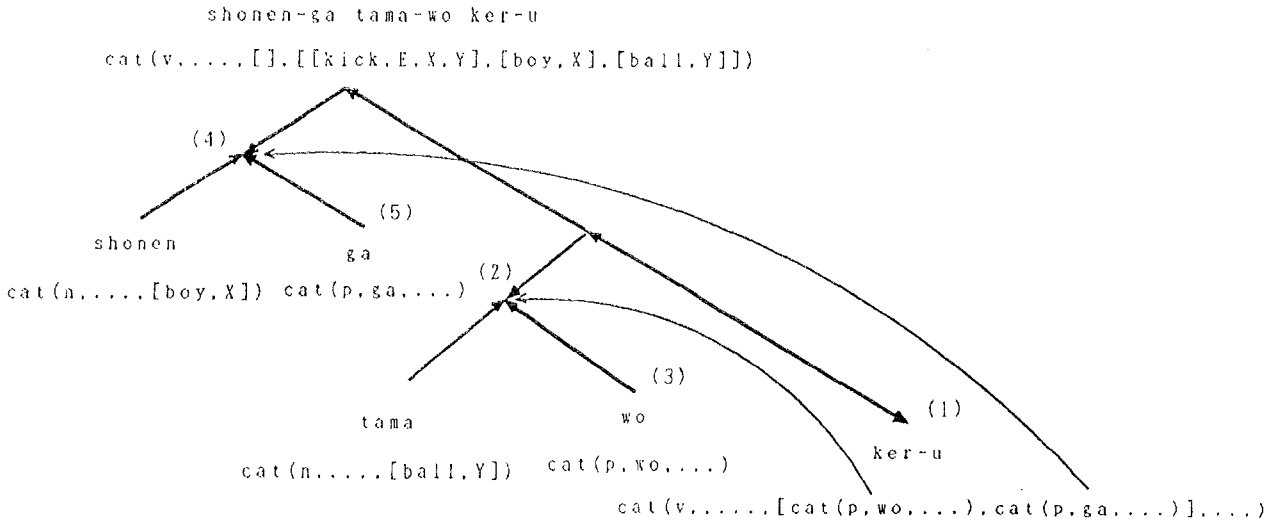

Figure 5: The generation process example.

This language has the ability to solve constraints in an active or passive way, but we use it for passive constraint solving.

4.2 An Example

Part of the grammar rules and a lexicon, and the process of generation from the logical form,

[[kick,E,X,Y] , [boy,X] , [ball,Y]]

are shown in figure 4 and figure 5.

The flow of the algorithm is described as follows; First, base lexical item (1)(a top down prediction) is predicted.This prediction instanciates the packed subcategorization list, and constrains counterpart conditions(2). Next, a lexical item is used to satisfy the constraints(2). Since this goal cannot be achieved by only the existing lexical item, a functional element is inserted that observes the restriction of the subcategorization list(3). As for (4) and (5), a similar process continues until a sentence is

produced. Finally we get an output

shonen-ga	tama-wo	ker-u.
a boy	a ball	kicks

from the semantic expression.

Figure 6 shows the result tree of real cu-PROLOG execution.

```
v[vcr]:[[kick,E_2516,Y_2517,Y_2518],[boy,Y_2517],[ball,Y_2518]]---[subcat_p]
 |
 |--p[ga]:[boy,Y_2517]---[adjacent_p]
 |    |
 |    |--n[n]:[boy,Y_2517]---[shonen]
 |    |
 |    |__p[ga,AJA{n[n]}]:[boy,Y_2517]---[ga]
 |
 |__v[vcr,SC{p[wo]}]:[[kick,E_2516,Y_2517,Y_2518],[boy,Y_2517],[ball,Y_2518]]---[subcat_p]
      |
      |--p[a]:[ball,Y_2518]---[adjacent_p]
      |    |
      |    |--n[n]:[ball,Y_2518]---[tama]
      |    |
      |    |__p[wo,AJA{n[n]}]:[ball,Y_2518]---[wo]
      |
      |__v[vcr,
           SC{p[ga],p[wo]}]:[[kick,E_2516,Y_2517,Y_2518],[boy,Y_2517],[ball,Y_2518]]---[ker]

 E = E_2516  X = Y_2517  Y = Y_2518  S = [shonen,ga,tama,wo,keru]
```

Figure 6: The real execution result of the example using the cu-PROLOG.

5 Remaining Problems

5.1 The Problem of Logical Form Equivalence

The problem of logical form equivalence has been discussed in [App87]. This problem concerns generation algorithms which are sensitive to logical forms. Namely, if an input semantic expression is converted by meaning postulates, different expressions with the same meaning are produced by different procedures.This problem occurs in the generation of expressions that have quantificational ambiguities.

A conversion using meaning postulates does not need syntactic or semantic information, but needs discourse information. Since generation strategies such as [Shi89], [Cal89] as well as ours use syntactic and semantic information, it is reasonable not to consider those operations. However, algorithms must have enough extendibility to reflect discourse information.

5.2 Lexical Indexing Strategy

Searching lexical items is very important for efficient algorithms. Metaknowledge about semantic expressions is necessary for this purpose, especially in the case of complex ones[Cal89].

This problem is not peculiar to generation. For example, discourse processing in which various inferences are executed by using the semantic expressions has the same problem. To cope with this problem, [Hob85] has proposed more simple logical forms. I also consider this a good idea for generation.

5.3 Controlling Search Using Discourse Information

Many sentences corresponding to one meaning can be generated by our algorithm(or other algorithms). Idealistically all sentences are distinctively produced according to other information such as discourse information. Our algorithm has the possibility for easily realizing this mechanism.

Suppose that an information unit agrees with a predicate in the semantic information. Control of the element position is realized by solving the constraint of the older information in turn[7]. Passivisation in Japanese is achieved by controlling the insertion of functional elements.

Transformational Grammar(the antecedent of Parameter and Principle theory) indicates the interesting piece of data that have many sentences with the same meaning. This concerns the position of the elements and the introduction of functional elements. As mentioned above, our algorithm is capable of reflecting discourse information on surface structures because of constraints.

[7]This assumption is problematic because correspondence is not guaranteed and if a semantic element is uniquely mapped to a lexical entry, the position is directly designated by the order of the semantic elements.

The algorithm proposed in [Wed88] can generate sentences that reflect discourse phenomena such as topicalization in LFG, but the formalization of topic greatly helps to simplify the algorithm.

6 Conclusion

In this paper an efficient bottom-up generation algorithm for principle-based grammars using constraint propagation is proposed, and a solution to bottom-up generation problems is mentioned. Issues about implementation and an example processed by the algorithm are also shown. Both the parser[Tud89] and the generator use the same grammar, that is, the grammar is reversible.

Since problems not inherent in bottom-up generation are connected to the logical form problem, or the knowledge representation problem, they should be discussed more deliberately from the viewpoint of generation.

Acknowledgement

The author wishes to extend his sincere gratitude to Yoshihiko Hayashi, Tsuneaki Kato, and Gen-ichiro Kikui for their comments on the initial idea of this generation algorithm. The author also wishes to express his indebtness to Tuda Hirosi, and Hasida Koiti of ICOT for permitting him to use the cu-PROLOG. Thanks are also due to Dr. Terashima , Mr. Sakama, Mr. Higashida, Mr. Shimazaki and all the members of the Natural Language Processing Division for their cooperation.

References

[App87] Douglas E. Appelt. 1987. "Bidirectional grammars and the design of natural language generation systems", Theoretical Issues in Natural Language Processing - 3, New Mexico, 185-191.

[Cal89] Jonathan Calder, Mike Reape, and Henk Zeevat. 1989. "An Algorithm for Generation in Unification Categorial Grammar", Proceedings of the 4th Conference of the European Chapter of the Association for Computational Linguistics, Manchester, 10-12 April, 233-240.

[Din86] Dincbas, M. 1986. "Constraints, Logic Programming and Deductive Databases", Proceedings of France-Japan Artificial Intelligence and Computer Science Symposium 86, Tokyo, 30 November-3 December, 1-27.

[Has87] Hasida Koichi and Isizaki Shun. 1987. "Dependency Propagation: a Unified Theory of Sentence Comprehension and Generation", Proceedings of the 11th International Conference on Computational Linguistics, Milan, 23-28 August, 664-670.

[Hob85] Hobbs, R. 1985. "Ontological Promiscuity", 23rd Annual Meeting of the Association for Computational Linguistics, New Jersey, 61-69.

[Pol87] Carl Pollard and Ivan A.Sag. 1987. "An Information-based Syntax And Semantics", Vol 1 Fundamentals, CSLI Lecture Notes Number 13.

[Gun87] Takao Gunji. 1987. "Japanese Phrase Structure Grammar", D.Reidel Publishing Company.

[Shi88] Stuart M. Shieber. 1988. "A Uniform Architecture for Parsing and Generation", Proceedings of the 12th International Conference on Computational Linguistics, Budapest, 22-27 August, 614-619.

[Shi89] Stuart M. Shieber, Gertjan van Noord, Robert C. Moore, and Fernando C.N.Pereira. 1989. "A Semantic-Head-Driven Generation Algorithm for Unification-Based Formalisms", 27th Annual Meeting of the Association for Computational Linguistics, British Columbia, 26-29 June, 7-17.

[Tud89] Tuda Hirosi, Hasida Koiti, and Sirai Hidetosi. 1989. "JPSG Parser on Constraint Logic Programming", Proceedings of the 4th Conference of the European chapter of the Association for Computational Linguistics, Manchester, 10-12 April, 95-102.

[Wed88] Jurgen Wedekind. 1988. "Generation as Structure Driven Derivation", Proceedings of the 12th International Conference on Computational Linguistics, Budapest, 22-27 August, 732-737.