TR-I-0207

単一化に基づく生成の効率改善

Optimization of Unification-based Generation

上田良寛
Yoshihiro Ueda

1991.3

概要

双方向文法を用いる生成のメカニズムとして提案したタイプ付素性構造主導型生成の高速化について述べる。単一化を基礎メカニズムとして用いているシステムでは、単一化それ自体が計算コストのかかるプロセスであるため、生成プロセスのほとんどを単一化によって消費されている。このため、単一化のアルゴリズムを改良する、または、単一化の適用回数を減少させることにより、生成システムの効率を向上させることが出来る。ここでは、後者の方法、すなわち、単一化の適用回数を減らすことによる効率向上を、文法および生成メカニズムの両面から試みた。文法の改良により、最高10倍の効率改善を達成した。

ATR Interpreting Telephony Research Laboratories
ATR 自動翻訳電話研究所

Optimization of Unification-based Generation

Abstract

Here the methods to improve the efficiency of the Typed-Feature-Structure-Directed Generation, a unification-based generation mechanism which is developed for dialogue translation, are described. Unification is a time-consuming process and, in systems that use unification as their basic mechanism, most of the computing time is consumed by unification. Better algorithms for unification and/or reducing the amount of unification can improve the efficiency of such systems. We have adopted the latter approach, experimenting with several methods from both the mechanism side and the grammar side. For the mechanism, delaying surface lexical selection and eliminating disjunctive feature structures in the derivation tree can reduce the generation time up to one-third in some cases. Modification of the grammar to reduce nondeterminism is so effective that it can increase the efficiency up to 10 times.

1. Introduction

Bidirectional grammar is now widely adopted by generation systems because the grammar and lexicon can be developed in a consistent way ([2], [5], [7]). In particular, unification-based systems are desirable as they can handle a variety of information including syntax, semantics, pragmatics, etc. This information is required to reflect the speaker's intention, i. e., "what the speaker wants to say and express" in the translation of spoken dialogues ([9]).

Thus, we have developed the Typed-Feature-Structure-Directed Generation mechanism ([14]), which uses unification-based bidirectional grammar. In this system, constraints described in Typed Feature Structures ([1]) are attached to the grammar rules and used to select the rules to control the generation process.

Unification itself is a time-consuming process and, in systems that use unification as their basic mechanism, most of the computing time is consumed by unification. Better algorithms for unification and/or reducing the number of calls for unification can improve the efficiency of such systems. In this paper, the latter approach is taken both from the mechanism side and the grammar side.

In this paper, we will first give a brief overview of the mechanism and the grammar of the system. Problems with the mechanisms and the grammar are considered and the possible solutions are proposed in section 3. Experiment results are shown in section 4.

2. Brief Overview of the System

2.1 Mechanism: Typed-Feature-Structure-Directed Generation

The basic mechanism of this system is repetitive top-down application of CFG rules. Along with a rule application, a feature structure attached to the mother node is distributed to their daughter nodes according to the specifications attached to the rule. This process constructs the derivation tree.

Simple top-down application of the CFG rules often leads to the derivation of unnecessary phrase structures. Furthermore, this causes termination problems, i.e., infinite application of the same grammar rules ([11]). Selecting appropriate rules is required to avoid these problems ([14]). In particular, declarative rule selection is desirable for unification-based grammars.

Typed-Feature-Structure-Directed Generation is developed to solve this problem. The CFG rules are selected according to the constraints attached to them. The features construct a type hierarchy and the constraints can be represented by the typed feature structures ([1]).

The rule (1), when used in analysis, consumes one element in the subcat list and application of this rule is terminated when the subcat is exhausted.

```
VP =HC*=> (VP XP)                                              (1)*
     (<!m !sem> == <!head-dtr !sem>)
     (<!head-dtr !subcat first> == <!comp-dtr-1 synsem>)
     (<!head-dtr !subcat rest> == <!m !subcat>)
```

The finite length of the input sentence also helps the termination when the sentence is analyzed in a bottom-up way. However, in generation, it infinitely appends the subcat list to the daughter VP and the application never terminates.

We classify verbs into three subtypes (Monadic, Dyadic and Triadic) according to their argument numbers. Then, the following constraint can be attached to the rule to restrict the possible length of the subcat list† and avoid infinite application of this rule. Types are shown here in bold italics.

```
(:or ((<!head-dtr !subcat rest> == [list-end])
      (<!sem reln> == (:or [dyadic] [triadic])))
     ((<!head-dtr !subcat rest rest> == [list-end])
      (<!sem reln> == [triadic])))
```

These constraints can be written in a purely declarative way. The hierarchy (or lattice) constructed by types also gives maximum flexibility in describing the constraints when the grammar grows larger and the feature structures become complex. For example, the type hierarchy for verbs can be reconstructed as in Fig. 1 to allow a new type ***dyadic/triadic*** to represent OR combination of two types.

---

* In this rule, =HC*=> link shows that the first element of the right hand symbols becomes the head daughter and the others the complement daughters. =CH=> link is also supplied for Complement-Head structures. A symbol with an exclamation mark (!) indicates a predefined template. In this rule, !m stands for mother, i.e., left-hand VP.

† This grammar uses Borsley's modification of HPSG ([3]), which uses a separate slot for the subject apart from subcat list. Thus the length of the subcat list is 2 for triadic verbs and 1 for dyadic verbs.

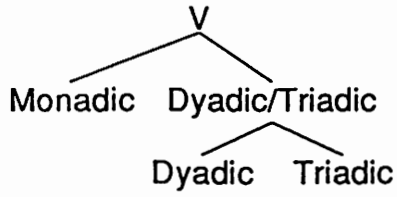

Fig.1 Type hierarchy for verbs

The constraints can thus be rewritten as follows.

```
(:or ((<!head-dtr !subcat rest> == [list-end])
      (<!sem reln> == [dyadic/triadic]))
     ((<!head-dtr !subcat rest rest> == [list-end])
      (<!sem reln> == [triadic]))
```

2.2 Grammar: Revised Analysis of HPSG

The grammar for the generation system is based on a new analysis of HPSG ([13]), an extension of its former version ([12]). Fundamentals of the formalisms such as the head feature principle, the subcat feature principle, etc. remain the same but the formalisms are extended to explain more broad language phenomena.

The most significant extensions are made on its representation structures obtained from the analysis of sentences. The former structure is shown in Fig. 2 (a) and the revised structure in (b). A new feature SYNSEM, which is the combination of syntactic structure and semantic structure, is introduced and it is now used as the element of SUBCAT list instead of the phrase itself. Each element in the structure has a specific role, e. g., a LOCAL structure becomes a SLASH element and a PARAM structure becomes a REL (relative) element.

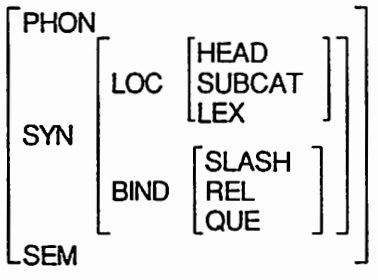

Fig.2 a) Former feature structure

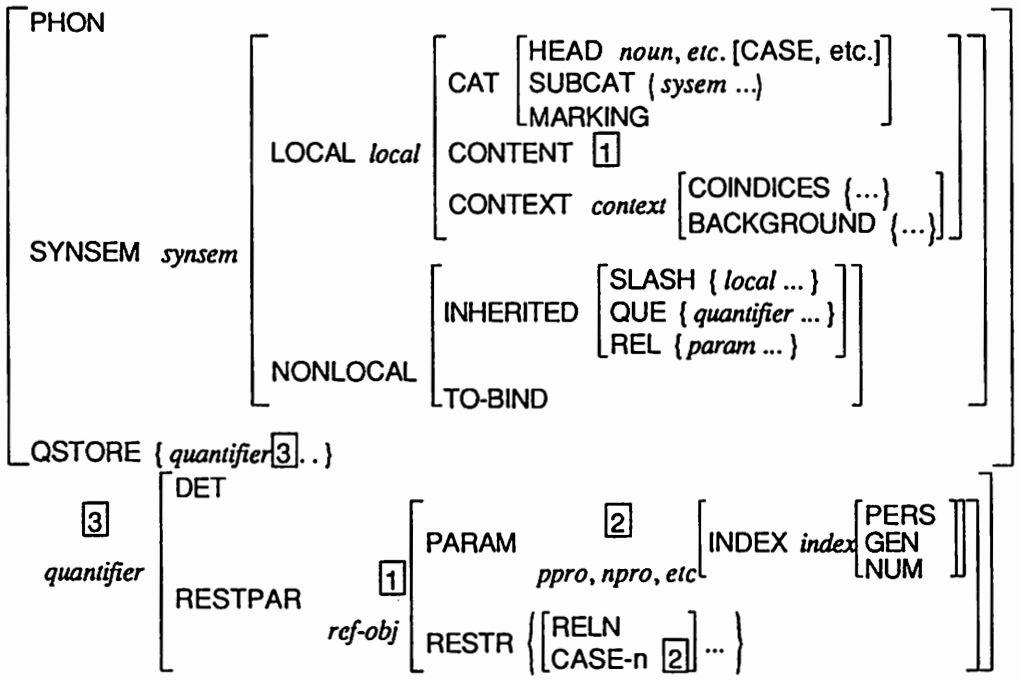

Fig. 2 b) Revised Feature Structure

One of the changes that affects the system is the use of sets in feature structures. Set operations such as UNION are also introduced. For example, modification by an adjective becomes a restriction in the restricted parameter representation of the modified object. Thus, the restrictions construct a set including a restriction of the noun itself. The feature structure shown in Fig. 3 is the semantic representation for a modified noun "big blue book."

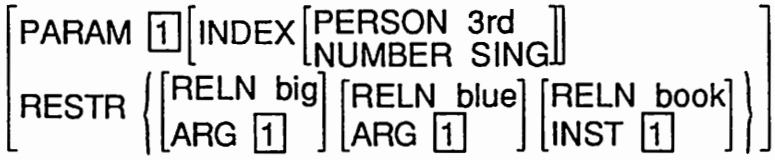

Fig.3 Feature Structure containing a set

Sets and set operations cannot be well formulated by ordinary unification. An extension of the unification by adding a function-calling mechanism makes the bidirectional use of grammar and lexicon impossible. The reverse operation of UNION, which is necessary in generation, is especially difficult to define.

In our implementation of the grammar, differential lists are used instead of sets. Lists preserve the order of elements, and the order in the semantic representation reflects the surface constituents' order. In other words, the order in the semantic representation given as the generation input defines the result of generation. This leads to an undesirable

consequence in that the generation system loses its ability to define the order of the results, e.g., the appropriate order of adjectives.

This can be solved by representing all the possibilities of the order using disjunction and filtering out those that are ungrammatical by restrictions in the grammar. Yoshimoto described such restrictions for the sequence of post-verbs in unification grammar for Japanese analysis ([15]). We experimented with making this Japanese grammar bidirectional and such restrictions helped considerably.

3. Problems and Possible Solutions

Here we consider some problems with the efficiency of the generation and methods to improve it by reducing the amount of unification. The approach is made both from the mechanism side and the grammar side.

3.1 Constraint Check Mechanism

Unification-based systems in general use unification in two ways: as a constraint that restricts the inappropriate rules to be applied by failure of the unification, and propagating the result of unification to the next process or step.

Unification also has two roles in the unification grammars. Constraints are for checking the grammar rule applicability and the unification results are syntactic/semantic representation of phrases that are propagated from node to node (in the generation case, from the mother node to its daughter nodes). In this generation system, the constraints are described as the equations attached to the CFG rules as well as the specifications of feature structures for the nodes. The equations construct one feature structure that contains both the constraint portion and the propagation portion.

The advantage of such embedding of the constraint portion in the propagation portion is that embedding helps to describe both the constraints and the specification for transferring feature structures in a uniform way. Furthermore, this requires no special mechanism for constraint checking.

However, it can be a disadvantage from the viewpoint of efficiency. The constraint portion of the feature structure may be applied after the resulting feature structure is almost completed. In such a case, the process of creating the resulting feature structure is totally abandoned. The current constraint check mechanism that uses uniform unification can create such incomplete feature structures and make the system inefficient.

Thus, efficiency can be increased by unifying the constraint checking portion first and then unifying the transferring portion only to those feature structures that have survived the constraint check.

Furthermore, the unification result of the constraint check portion is not necessary and can be discarded. Thus, unification can be replaced by a lighter process that only checks the unifiability.

To achieve this, the constraint portion is explicitly separated from the transferring portion in the grammar rules. For example, rule (1) can be rewritten as follows.

```
VP =HC*=> (VP XP)                                              (1')
  (<!m !sem> == <!head-dtr !sem>)
  (<!head-dtr !subcat first> == <!comp-dtr-1>)
  (<!head-dtr !subcat rest> == <!m !subcat>)
  (:info :gen
    (:or ((<!head-dtr !subcat rest> == [list-end])
          (<!sem reln> == (:or [dyadic] [triadic])))
         ((<!head-dtr !subcat rest rest> == [list-end])
          (<!sem reln> == [triadic]))))
```

To make the constraints independent of other parts has another advantage. The constraints are required only by generation and the grammar compiler can remove them from the analysis grammar. This reduces an unnecessary load on the unification during analysis.

Kogure proposed "early failure finding strategy" in his unification algorithm from the same motivation ([10]). His algorithm uses the statistical information of unification success and failure and such information can be inaccurate. The method described here avoids this problem by forcing the constraints to be explicitly described.

3.2 Inefficiency Caused by Disjunctive Feature Structures

This system uses disjunctive feature structure to handle the multiple surface forms in a single lexical entry (called a lexical unit). This avoids making a copy of the derivation tree for each candidate of the surface form. For example, the lexical unit for the verb "be" can be described as follows.

```
(DEFLEX-UNIT |be-Unit| DYADIC
 (:or
 (!finite-form
  (:or (!present-tense
        (:or ((<word> == "am") !1sg-subj-agr)
             ((<word> == "are")
              (:or ((!2sg-subj-agr) (!pl-subj-agr)))
```

```
                ((<word> == "is") !3sg-subj-agr)
...   )         ;; for past form, particles, etc.
```

When the subject gets its semantics including NUMBER and PERSON, the in the lexical unit are simultaneously resolved by the subject-verb agreement[1]. The semantics of the subject are given by the subcat frame of the verb and, in most cases, the semantics of the subject and the surface form of the verb are simultaneously determined when the derivation reaches the leaf and the lexical unit is selected.

However, some disjunctions survive until the subject is determined. We use the following label notation to represent the speaker.

[SYNSEM|LOCAL|CONTENT [LABEL *SPEAKER*]]

This is usually generated as "I" (first-person, singular pronoun). However, it must be rendered as "this" in sentences such as "this is the conference office" in telephone dialogues. Selection of these two surface realizations of the subject affects the selection of the main verb "be" ("am" or "is"). Until the selection is completed, the disjunction remains unresolved.

Kasper's algorithm is used for the unification of disjunctive feature structures ([8]). However, Carter pointed out that disjunctive unification in general is a very time-consuming process ([4]). Keeping disjunction in the feature structures during the course of unification decreases the efficiency of the generation.

Shieber et al. proposed "postponing lexical choice" for a similar problem ([11]). We adopt the same approach, which adds an identifier to the lexical unit instead of attaching each disjunctive component to the derivation tree. Unifying the disjunctive components with the derivation tree and resolving them is delayed until the whole tree is constructed.

3.3 Distribution of Quantifiers

We consider the problem of distributing the Quantifier Storage as one of the problems of the grammar.

---

[1] Agreements are represented as follows using a template.

```
(deffstemp !3sg-subj-agr ()
     (<!subj-1 !cont param index pers> == 3rd)
     (<!subj-1 !cont param index num>  == sing))
```

The new analysis of HPSG adopted the Quantifier Storage (Cooper Storage), and the information on quantifiers is analyzed as a QSTORE feature apart from semantic content. A sentence that contains "every student" and "the book" will be analyzed as Fig. 4. Here each element of the QSTORE points to some part of the semantic content.

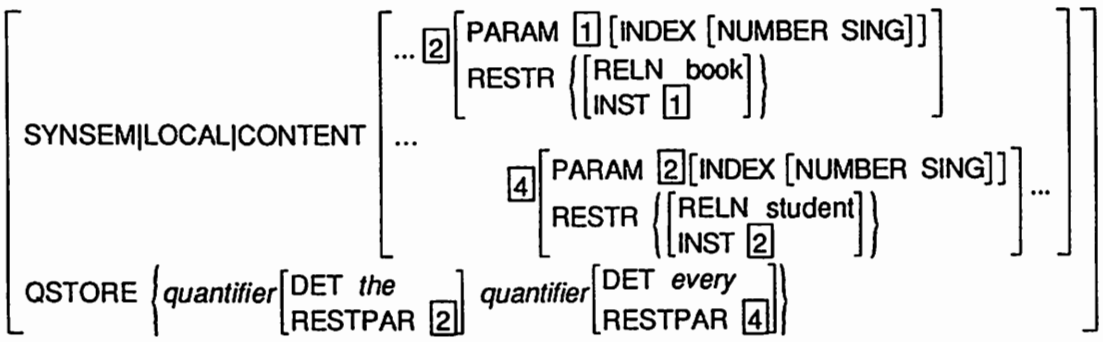

Fig. 4 Representation that uses Quantifier Storage

QSTORE is also represented using a differential list in our implementation of the grammar. One of the shortcomings of lists including differential lists is that they can be traversed only from their first element. No problem arises if the elements are generated according to the order of occurrence in the list as in the case of adjective modification.

However, the QSTORE is not such a case. Until the semantic content for the first QSTORE element is determined, quantifier information for other constituents is left undetermined and every quantifier element possibility is tried. Shieber et al. adopted a "shuffling" operation for this purpose ([11]). Such nondeterminism increases the possible number of derivation trees and decreases the efficiency of the generation process.

To solve the nondeterminism problem caused by the quantifier storage distribution, we consider the following approaches:

1) Improving unification of differential lists
2) Attaching procedures to check rule applicability
3) Modifying the grammar.

One way to improve the unification of the differential lists (dlists) is to make them accessible from both ends of the list. When the generation process doesn't know from which end the quantifier storage is consumed, this approach can help to reduce the nondeterminism. We call this the "double dlist" approach.

However, this approach also has a limitation. There are cases in which the quantifier storage has more than two quantifier elements and the generation process starts from the quantifier element in the middle of the quantifier storage. In such cases, the appropriate quantifier cannot be determined.

Next, we will consider attaching procedures to check the rule applicability. We will call them "procedural constraints." The procedural constraint attached to the NP rule must traverse the quantifier store and identify the one which quantifies the semantic content of the NP.

Procedural constraints are powerful and flexible but they can make the grammar difficult to read and write, particularly for grammar writers.

The third approach is to treat the problem within the grammar level. One possible solution according to this approach is to put the quantifier information somewhere in the semantic content. Here we put the DET feature under the PARAM feature in the semantic content. Fig. 5 shows the feature structure given to the phrase "every student."

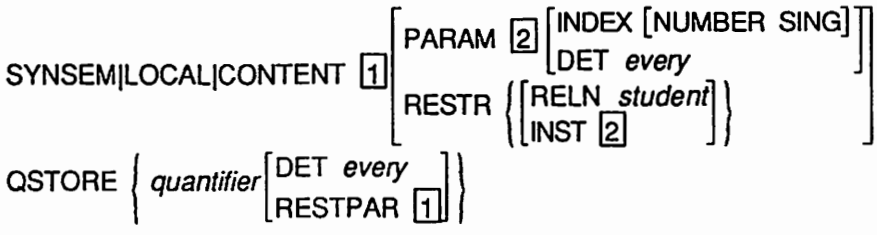

Fig. 5 Quantifier information in semantic features

This modification allows the generation process to identify the quantifiers, yet does no harm to the quantifier storage analysis.

4. Experiment[2]

4.1 Method

We have conducted an experiment to determine the effects of these possible solutions.

The sentences to be generated are the following:

---

[2] Details of the experiment method and the result are shown in the appendix (in Japanese).

S-1) My boss attends every big conference.
(two quantifiers)
S-2) The conference office sent the professor a registration form.
(three quantifiers)
S-3) This is the conference office.
(to see the effect of delayed surface selection)

We prepared 5 separate versions of the grammar that uses the different approaches to the problems. Grammars from G-2 thru G-4 adopt the solutions for the quantifier distribution problem. Grammar G-5 also includes the delayed surface selection for the disjunctive feature structure problem.

G-1) Original untouched version
G-2) uses double dlist
G-3) checks procedural constraints
G-4) places the determiner in the semantic content
G-5) delays the surface lexical selection

For each grammar, three constraint checking methods were tried.

a) unification of feature structures which embed the constraint
b) independent unification of the constraints
c) check of the unifiability of the constraints

This experiment was performed in Sun Common Lisp on a SPARCStation 1+.

4.2 The Result

The result is shown in Table 1 - 3. Every combination of the parameter (grammar and constraint check method) was tried 5 times. The generation time figure used here is their average value excluding those distorted by GC.

check method	G-1	G-2	G-3	G-4	G-5
a	7.87	3.49	2.18	2.36	2.39
b	7.36	3.64	2.23	2.26	2.23
c	6.96	3.09	1.97	2.03	2.16

Table 1 Generation Time (sec) for Sentence 1

check method	G-1	G-2	G-3	G-4	G-5
a	28.91	9.83	2.40	2.61	2.54
b	31.71	9.75	2.26	2.58	2.39
c	27.52	8.24	2.40	2.56	2.34

Table 2 Generation Time (sec) for Sentence 2

check method	G-1	G-2	G-3	G-4	G-5
a	17.71	7.52	7.54	5.63	1.76
b	18.26	7.81	7.62	5.80	1.69
c	17.82	7.60	7.54	5.55	1.60

Table 3 Generation Time (sec) for Sentence 3

Fig. 6 summarizes the generation time of the different versions of the grammar. Here the constraint check method (c) is used.

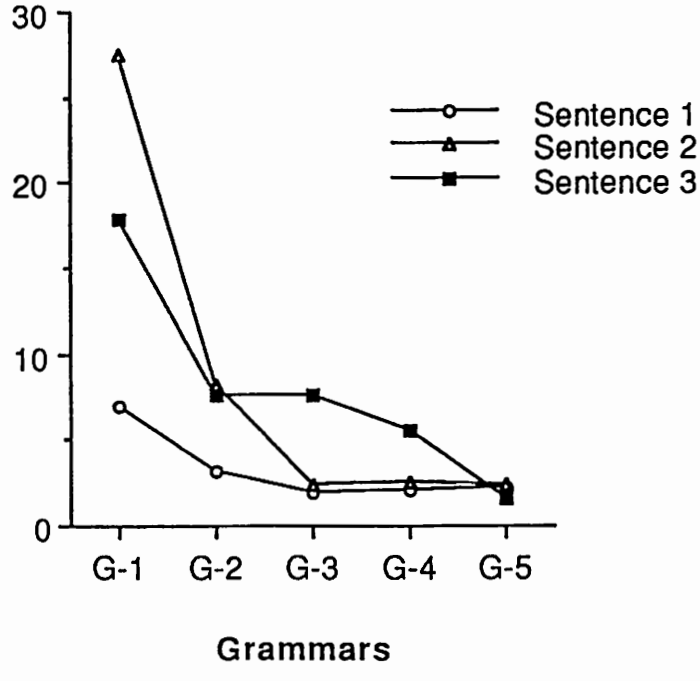

Fig. 6 The effect of changing grammars

The effect of the solutions to the quantifier storage distribution problem can be seen in this figure. The grammar G-2 which uses a double dlist effectively reduces the generation time. However, we can see that the effect is not sufficient by comparing the result with those of G-3 and G-4. Procedural constraints (G-3) and the determiner in the

semantics (G-4) are equally responsible for further reduction. A ten-fold increase in efficiency is made in the case of sentence 2.

The generation time for sentence 3 is reduced by the grammar G-5 because of the delayed lexical choice. The reduction factor is almost 1/3 (5.55 sec in G-4 to 1.60 sec in G-5).

The effect of changing constraint check methods is summarized in Fig. 7. Here the data is taken from the grammar G-5. A drastic improvement cannot be achieved by this method.

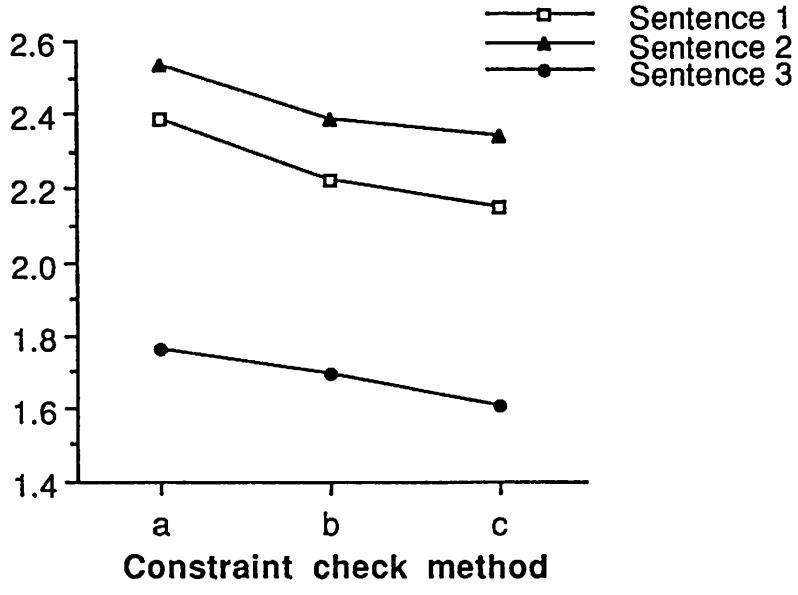

Fig.7 The effect of changing constraint check method

5. Conclusion

In this paper, we have proposed and experimented with several methods to improve the efficiency of a unification-based generation system. The greatest improvement was made by reducing nondeterminism in the grammar. For the mechanism, delaying surface lexical selection and eliminating disjunctive feature structures in the derivation tree can reduce the generation time to about one-third in some cases. Improvement of the constraint check method doesn't give such drastic results nevertheless, it is also important because the effect is applicable to any implementation of grammar and to any sentences.

The grammar used here must be extended to handle a variety of speech acts in the telephone dialogue. We have learned that it is most important to reduce nondeterminism in order to make the grammar

efficient. Typed-Feature-Structure-Directed Generation provides a way of reducing such nondeterminism using constraints represented by the type hierarchy.

The mechanism itself is also a target of improvement. We have combined Semantic-Head-Driven generation ([11]) and our method, and the combined mechanism is now under evaluation.

We have not discussed improvement of the unification algorithm but this also plays a very important role. Unification occupies a significant portion of computation time in unification-based parsing and generation. Several algorithms have been proposed ([10], [6]) and we can expect them to help increase the efficiency of the generation.

References

[1] Aït-Kaci, H.: An Algebraic Semantics Approach to the Effective Resolution of Type Equations, *Theoretical Computer Science* 45, pp. 293 - 351, North-Holland (1986).

[2] Appelt, D. E.: Bidirectional Grammars and the Design of Natural Language Generation Systems, *Theoretical Issues in Natural Language Processing* - 3, pp. 185 - 191, Las Cruses (1987).

[3] Borsley, R. D.: Subjects and Complements in HPSG, *Report No. CSLI-87-107*, CSLI, Stanford (1987).

[4] Carter, D.: Efficient Disjunctive Unification for Bottom-Up Parsing, in *Coling-90*, Vol. 3, pp. 70 - 75, Helsinki (1990).

[5] Dymetman, D. and Isabelle, P.: Reversible Logic Grammars for Machine Translation, in *2nd International Conference on Theoretical and Methodological Issues in Machine Translation*, Pittsburgh (1988)

[6] Godden, K.: Lazy Unification, in *28th Annual Meeting of the Association for Computational Linguistics*, pp. 180 - 187, Pittsburgh (1990).

[7] Jacobs, P.S.: PHRED: A Generator for Natural Language Interfaces, *Computational Linguistics*, Vol. 11, No. 4, pp. 219 - 242, MIT Press (1985).

[8] Kasper, R. T.: A Unification Method for Disjunctive Feature Descriptions, in *25th Annual Meeting of the Association for Computational Linguistics*, pp. 235 - 242, Stanford (1987).

[9] Kogure, K.: A Method of Analyzing Japanese Speech Act Types, in *2nd International Conference on Theoretical and Methodological Issues in Machine Translation*, Pittsburgh (1988),.

[10] Kogure, K.: Strategic Lazy Incremental Copy Graph Unification, in *Coling-90*, Vol. 2, pp. 223 - 228, Helsinki (1990).

[11] Shieber, S. M., van Noord, G., Moore, R. C., and Pereira, F. C. N.: Semantic-Head-Driven Generation, in *Computational Linguistics,* Vol. 16, No. 1, pp. 30 - 42, MIT Press (1990).

[12] Pollard, C. and Sag, I. A.: *An Information-Based Syntax and Semantics, Volume 1, Fundamentals*, CSLI Lecture Notes Number 13, Center for the Study of Language and Information, Stanford (1987).
[13] Pollard, C. and Sag, I. A.: *An Information-Based Syntax and Semantics, Volume 2, Topics in Binding and Control* , CSLI Lecture Notes (to appear), Center for the Study of Language and Information, Stanford (1991).
[14] Ueda, Y. and Kogure K.: Generation for Dialogue Translation Using Typed Feature Structure Unification, in *Coling 90*, Vol. 1, pp. 64 - 66, Helsinki (1990).
[15] Yoshimoto, K. and Kogure, K.: Phrase Structure Grammar for Inter-Terminal Dialog Analysis, in 37th Meeting of Japan Information Processing Society, 5C-5 (1988).

付録: 生成効率改善実験

文法ieg2のシリーズを用いて生成効率改善の実験を行った。その実験方法と実験データを示す。

実験方法

例文
1) my boss attends every big conference
2) this is the conference office
3) the conference office sent the professor a registration form

文法
ieg2 Grammar 1。特に対策をたてていないもの。
ieg2-2 Grammar 2。双方向DListをもちいたもの。
ieg2-3 Grammar 3。手続き的制約を用いたもの。
ieg2-5 Grammar 4。determinerをsynsem素性の下に入れたもの。
ieg2-4 Grammar 5。Grammar 3に表層選択の遅延を加えたもの。

制約条件のチェック方法
:total チェック方法a。制約条件と素性構造伝播部分を同時に単一化。
:desc-check チェック方法b。制約条件部分を独立に単一化してチェック。
:desc-filtered チェック方法bの結果を、素性構造伝播部分の入力に使用。
:spec-check チェック方法c。制約条件部分の単一化可能性のみをチェック。

実験環境 Sun Common Lisp / SPARCstation 1+

実験の再現方法
コネクトディレクトリは/as06/ryu/

```
1 (load "achart/acp-gen")
2 (load "grammar/ieg2")
3 (setq *dribble-file* "grammar/test-gen-ieg2-1.log")
4 (load "grammar/test-ieg-seq.lisp" :print t)
```

実験結果

実験結果は、下記のファイルに残されている。

```
grammar/test-gen-ieg2-1.log
...
grammar/test-gen-ieg2-5.log
```

ここで得られたデータを次に示す。5回の生成を行い、その平均を左端に出している。測定データのうち、ガーベジコレクション("Dynamic Byte Consed")のため信頼できないもの

には＊をつけて、平均値の算出からは除いている。

ieg2　特に対策をたてていないもの(Grammar 1)。

S-1	:total	7.8	8.47	7.04	7.64	8.33	7.856
	:desc-check	6.82	7.51	8.81	6.35	7.3	7.358
	:desc-filtered	7.47	9.43	8.03	9.52	7.46	8.382
	:spec-check	7.09	6.38	7.81	6.3	7.23	6.962
S-2	:total	19.38	16.77	17.86	17.58	16.97	17.712
	:desc-check	18.84	18.26	18.17	18.64	17.37	18.256
	:desc-filtered	21.4	21.97	21.98	21.19	21.75	21.658
	:spec-check	17.43	18.42	18.03	17.05	18.13	17.812
S-3	:total	27.58	28.71	27.86	29.49	30.92	28.912
	:desc-check	30.95	32.95	32.81	37.14*	30.11	31.705
	:desc-filtered	35	36.83	33.89	35.85	36.09	35.532
	:spec-check	26.25	30.37	25.65	27.22	28.13	27.524

ieg2-2. DoubleDListを用いたもの(Grammar 2)。かなり効果が現れている。文３ではじゅうぶんな結果は得られない。

1	:total	3.28	4.44*	3.24	3.94	4.66*	3.487
	:desc-check	3.33	3.88	4.75*	3.28	4.07	3.64
	:desc-filtered	5.11*	4.38	3.62	4.72*	4.24	4.08
	:spec-check	3.11	4.59*	4.81*	2.66	3.5	3.09
2	:total	7.53	8.65*	7.78	8.22*	7.26	7.523
	:desc-check	7.93	8.61*	8.04	7.46	8.53*	7.81
	:desc-filtered	9.94	9.91	10.55*	10.14	10.58*	9.997
	:spec-check	7.31	7.77	7.96	8.33*	7.35	7.5975
3	:total	9.98	10.26	8.59	9.87	10.44	9.828
	:desc-check	9.62	10.45	10.66	8.44	9.6	9.754
	:desc-filtered	11.55	13.85	11.66	11.5	11.06	11.924
	:spec-check	7.82	8.62	8.45	9.11	7.2	8.24

ieg2-3. チェック手続きを用いたもの(Grammar 3)。文3のようにqstoreのリストが３個以上で先頭からも末尾からも生成がなされないものにも有効である。

1	:total	2.53	1.98	1.93	2.94*	2.26	2.175
	:desc-check	2.57	2.13	2	2.95*	2.23	2.2325
	:desc-filtered	2.54	1.96	1.92	3.30*	2.04	2.115
	:spec-check	2.5	1.76	1.8	3.40*	1.83	1.9725
2	:total	7.86	8.42*	7.65	7.16	7.47	7.535
	:desc-check	8.41*	7.38	7.69	7.8	8.03*	7.623
	:desc-filtered	8.54	8.84	9.02	9.54*	8.98	8.845
	:spec-check	7.27	7.81	8.01*	7.7	7.37	7.5375

3	:total	2.96*	2.42	2.53	2.26	3.48*	2.403
	:desc-check	2.04	2.05	2.05	2.03	3.11	2.256
	:desc-filtered	2.2	3.22	2.22	3.98*	2.17	2.4525
	:spec-check	3.96*	1.95	2.66	2.6	3.32*	2.403

ieg2-4. 表層形の選択を同時に行わず、最後まで遅らせたもの(Grammar 5)。文2のように主語が動詞によって決定され、それまで表層形が決定されないものにも有効である。しかし、その他の文に対してはかえって遅くなっている。

1	:total	2.44	3.79*	2.09	2.88	2.13	2.385
	:desc-check	2	3.51*	2.05	2.83	2.01	2.2225
	:desc-filtered	3.20*	2.46	2.73	2.48	3.34*	2.557
	:spec-check	1.94	2.73	1.95	1.99	3.25*	2.1525
2	:total	1.67	2.15	1.61	1.61	2.51*	1.76
	:desc-check	1.62	1.62	1.95	1.65	1.62	1.692
	:desc-filtered	2.06	1.68	1.7	2.26*	1.63	1.7675
	:spec-check	1.55	1.53	1.91	1.52	1.51	1.604
3	:total	4.00*	2.23	3.2	2.18	3.77*	2.537
	:desc-check	2.16	3.09	2.16	3.58*	2.15	2.39
	:desc-filtered	3.22	2.82	3.49*	2.71	3.36	3.0275
	:spec-check	2.09	4.12*	2.1	3.08	2.09	2.34

ieg2-5. determinerをsynsem素性の下に入れることによって非決定性をなくしたもの(Grammar 4)。表層形の選択の遅延は行っていない。

1	:total	2.39	3.82*	2.01	3.01	2.04	2.3625
	:desc-check	2.07	3.30*	2.41	2.54	2.02	2.26
	:desc-filtered	2.1	3.7	2.72	2.07	3.68*	2.6475
	:spec-check	2	1.92	2.3	1.91	3.31*	2.0325
2	:total	5.5	5.37	5.78	5.96	5.54	5.63
	:desc-check	6.56*	5.58	5.96	5.66	6	5.8
	:desc-filtered	7.52*	6.65	6.88	6.63	7.04	6.8
	:spec-check	6.54*	5.53	5.8	5.49	5.39	5.5525
3	:total	2.72	4.13*	2.27	3.14	2.32	2.6125
	:desc-check	4.18*	2.24	3.18	2.31	3.61*	2.577
	:desc-filtered	2.51	3.32	2.82	3.61*	2.39	2.76
	:spec-check	4.19*	2.23	3.29	2.15	3.56*	2.557