ICOT Technical Memorandum: TM-0679

TM-0679

Logical Dependency Grammar and Its Constraint Analysis

by
R. Sugimura

February, 1989

© 1989, ICOT

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

Logical Dependency Grammar and its Constraint Analysis

Ryôichi Sugimura
Institute for New Generation Computer Technology (ICOT)
4-28, Mita 1-chome, Minato-ku, Tokyo 108 Japan

AREA : C1. Natural Language
Multiple Submission : NONE

Abstract

This paper introduces logical dependency grammar which includes type-0 rewriting rule for modifying relations, and constraint analysis of dependency structures of sentence analysis.

It is well known that idiosyncratic gap between dependency structures and phrase structures reflects not only syntactic idiosyncracy but also semantic idiosyncracy.

Dependency structures in which one phrase modifies more than two phrases, can not be directly written in context free rules.

Therefore, in order to obtain the dependency structures, usual machine translation systems have been applying procedural methods. A defect of these systems is lack of declarative rules. In usual declarative phrase structure grammars for dependency structures, resultant syntactic trees do not reflect the resultant semantic structures.

In this paper, we propose declarative type-0 grammar rules for dependency structures. As the matter of fact, termination of rewriting process with proposed rules is guaranteed. Different from phrase structural approach, resultant dependency graphs reflects its semantic dependency structures.

More over, in our grammar, we can define constraints between ambiguous dependency structures of sentence analysis, and we can solve these constraints in Constraint Logic Programming framework.

As practical tool, we will briefly introduce an effective logic based parallel parser SAX-0 for type-0 language. Results of experiment will be reported.

1 Introduction

This paper introduces a tiny set of logic grammar rules, with which we can get dependency structures from input sentence. Dependency structure is constructed from constituents of a sentence, and *modifying relations* between them.

We will show that modifying relations in dependency structures, can be written in a set of simple rewriting rules including a type-0 and a type-1 rules. As the matter of fact, termination of rewriting process with proposed rules is guaranteed.

It is well known that there is syntactic and semantic idiosyncratic gap between Japanese and English [6]. Japanese has dependency structure in which one phrase modifies other phrases. On the other hand English has phrase structure.

The idiosyncratic gap between dependency structures and phrase structures, reflects not only syntactic idiosyncracy but also semantic idiosyncracy.

As to rewriting rules, we can not write context free rules for dependency structures in which one phrase modifies more than two phrases. Machine translations systems usually utilize procedures in order to get dependency structures [5] of agglutinative language such as Japanese or Korean language.

As the main feature of our system, we can analyse modifying relations in dependency structure in declarative rules with graphs which correspond to semantic structures.

As to the disambiguation of the result of sentence analysius, we propose a constraint approach[7]. In our approach, logical constraints between results of parsing can be propagated into context analysis.

Section 2 will summarize dependency grammars. Then, simple rules for dependency structure will be introduced. Section 3 will briefly introduce parallel parser which is the extended version of parallel parser SAX [4]. Section 4 describes about constraint analysis of the resultant dependency structures from parsing.

As the conclusion, we will consider the context sensitiveness in pragmatic

parsing.

2 Dependency Grammar

2.1 Dependency Structure

Agglutinative language such as Japanese or Korean language has dependency structures. A sentence in dependency structure can be segmented into a set of special grammatical phrases (called "bunsetsu" in Japanese). This paper uses the term "PHRASE" to mean minimal constituents of dependency structure. A PHRASE consists of some words and can be regarded as a minimal semantic elements to present various kinds of cases in a sentence. Any PHRASE has a dependent relation with other PHRASEs, or more exactly, *modifying relations* (MRs for short).

Formally, dependency structures are defined as follows.

DR1 All PHRASE except the last one should *modify* at least one PHRASE to the right of themselves. If a PHRASE modifies a PHRASE to the right of themselves, draw *arc* between them.

DR2 The last PHRASE modifies no other PHRASES.

DR3 No two *arcs* should cross each other.

With definition DR1, it is clear that dependency structure is not a tree but a graph. From a sequence "PHRASE_1, PHRASE_2", we can get the sequence "PHRASE_1, arc, PHRASE_2" when there holds modifying relation between PHRASE_1 and PHRASE_2. It is clear that rewriting rule for this transformation can be written in type-0 rule. Therefore, context free grammars can not generate and analyse dependency structure.

For example, the following figure [1] is the correct dependency structure. The following figure [2] is illegal because it breaks the rule R3.

2.2 Grammar Rules for Dependency Structure

Now, we will introduce a tiny set of grammar rules for dependency structure.

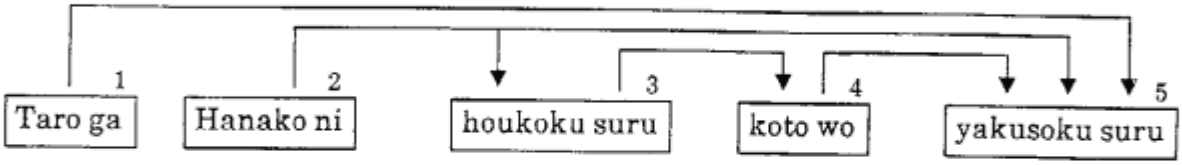

Figure 1: Dependency structure

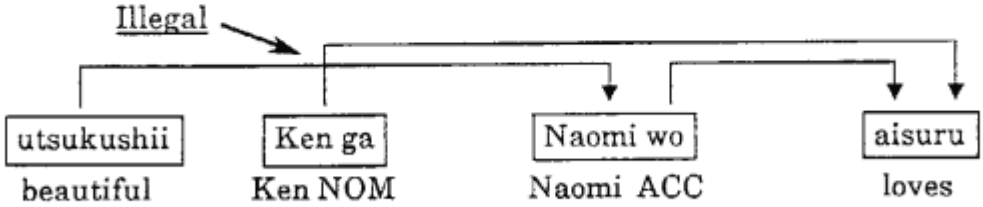

Figure 2: Illegal dependency structure

Following rules (1) and (2) correspond to the definition DR1. We will use symbol "ph" to note PHRASE. Rule (2) defines the *gap* between *modifying* PHRASE ph_1, and *modified* PHRASE ph_2.

$$\begin{aligned} & \mathrm{ph}_1, \mathrm{arc}, \mathrm{ph}_2 \\ & \quad --> \quad \mathrm{ph}_1, \mathrm{ph}_2 \qquad (1) \\ & \mathrm{ph}_1, \mathrm{arc}, \mathrm{ph}_2 \\ & \quad --> \quad \mathrm{ph}_1, (\mathrm{arc};[]), \mathrm{ph\_arc\_array}, \mathrm{ph}_2.\mathrm{ph\_arc\_array} \\ & \quad --> \quad \mathrm{ph\_arc\_array}, \mathrm{ph}, \mathrm{array}. \qquad (2) \end{aligned}$$

Analysis begins from a sequence "$\mathrm{ph}_1, \ldots, \mathrm{ph}_n$".

Rule 1 is type-0 rule. This rule is activated if there exists dependent relation between two adjacent phrases. "arc" which represents modifying relation is generated in this rule, if there holds modifying relation between PHRASEs.

It is very important to understand that this rule can be applied to a pair of *adjacent* PHRASEs, and if this rule applied, the result array of symbols can not be applied this rule. No other grammar rule generates adjacent PHRASES, therefore this rule always stops.

Rule 2 is type-1 rule. This rule is activated if there exists dependent relation between two *separated* PHRASES like "ph_1,...,ph_k" where ph_1 modifies ph_k. "(arc;[])" allows multiple dependent relations. "ph_arc_array" correspond to more than one array of "PHRASE , arc".

For practical use, we modify the above rules as follows.

```
ph(W1,Syn1,Sem1),                                          (3)
    arc([(W1,W2)]),                                        (4)
ph(W2,Syn2,Sem12)                                          (5)
              --> ph(W1,Syn1,Sem1),                        (6)
                  ph(W2,Syn2,Sem2),                        (7)
                  {const(Syn1,Sem1,Syn2,Sem2,Sem12)}.      (8)

ph(W1,Syn1,Sem1),                                          (9)
arc([(W1,W2),J—I]),                                        (10)
ph(W2,Syn2,Sem12)                                          (11)
              --> ph(W1,Syn1,Sem1),                        (12)
                  (arc(J);[]),                             (13)
                  ph_arc_array(I,_),                       (14)
                  ph(W2,Syn2,Sem2),                        (15)
                  {const(Syn1,Sem1,Syn2,Sem2,Sem12)}.      (16)
```

Rule in (1) is extended into formula from (3) to (8). Rule in 2 is extended into formula from 9 to 16. "W1" and "W2" is a logical variable which stand for syntactic category and that is used to draw dependency graph. If dependency relation holds between phrase in 6 and phrase in 7, "arc" in 4 will be generated. "arc" carries the pair of names of these two PHRASEs. This information is used to draw dependency graph.

Syn1 stands for syntactic features of modifying phrase. Syn2 stands for syntactic features of modified phrase.

"construct" is the predicate which checks whether dependency relation is holds or not. If there holds dependency relation, this makes new semantic information "Sem12". "Sem12" in (5) is made by "const" in (8), "Sem12" in (11) is made by "const" in (16). The main work for grammar writer is to

define this predicate "const". There is no need to consider another factor of the grammar for grammar writer.

3 Parallel parsing for Type-0 Language

As stated in previous section, grammar rules for dependency structure is in type-0 and type-1 language. Therefore there should be effective parser which can parse sentence with context sensitive type-0 and type-1 rewriting rules.

For this purpose, we extended our parallel parser SAX (of AX) [4]. SAX is a layered stream based chart parser. It executes analysis of a sentence bottom-up breadth-first. Restricted Gapping Grammar [1] [2] [8], is already impremented in it. Extention is on the point that instead of generating gaps, SAX asserts the programmed nonterminals. For example, if the rule 2 is holds, parser gemerates a sequence of head symbols.

In fact parsing will be carried out as follows.

From this parsing, we can get resultant dependency structure as follows.

4 Disambiguation with CLP

Lext consider that after parsing we got 5 ambiguities as follows.

First, lets define dependency graph.

$$Dgraph = (\Sigma, M) \tag{17}$$

$$\Sigma \stackrel{def}{=} \{X | X = \text{PHRASE (ph for short)}\} \tag{18}$$

$$M \stackrel{def}{=} \{arc(ph_a, ph_b) | ph_a \text{ modifies } ph_b\} \tag{19}$$

In the resultant dependency structure, Σ is the same, but M is different. Between M in resultant ambiguities there holds the following axioms.

$$M_1 \vee M_2 \vee \ldots \vee M_n = true \tag{20}$$

$$\text{where } M_i = arc(P1_a1, P1_b1) \wedge \ldots arc(P1_am, P1_bm) \tag{21}$$

From the resultant dependency structures, we can get the following formulae and get the conclusions after applying the above constraint. The

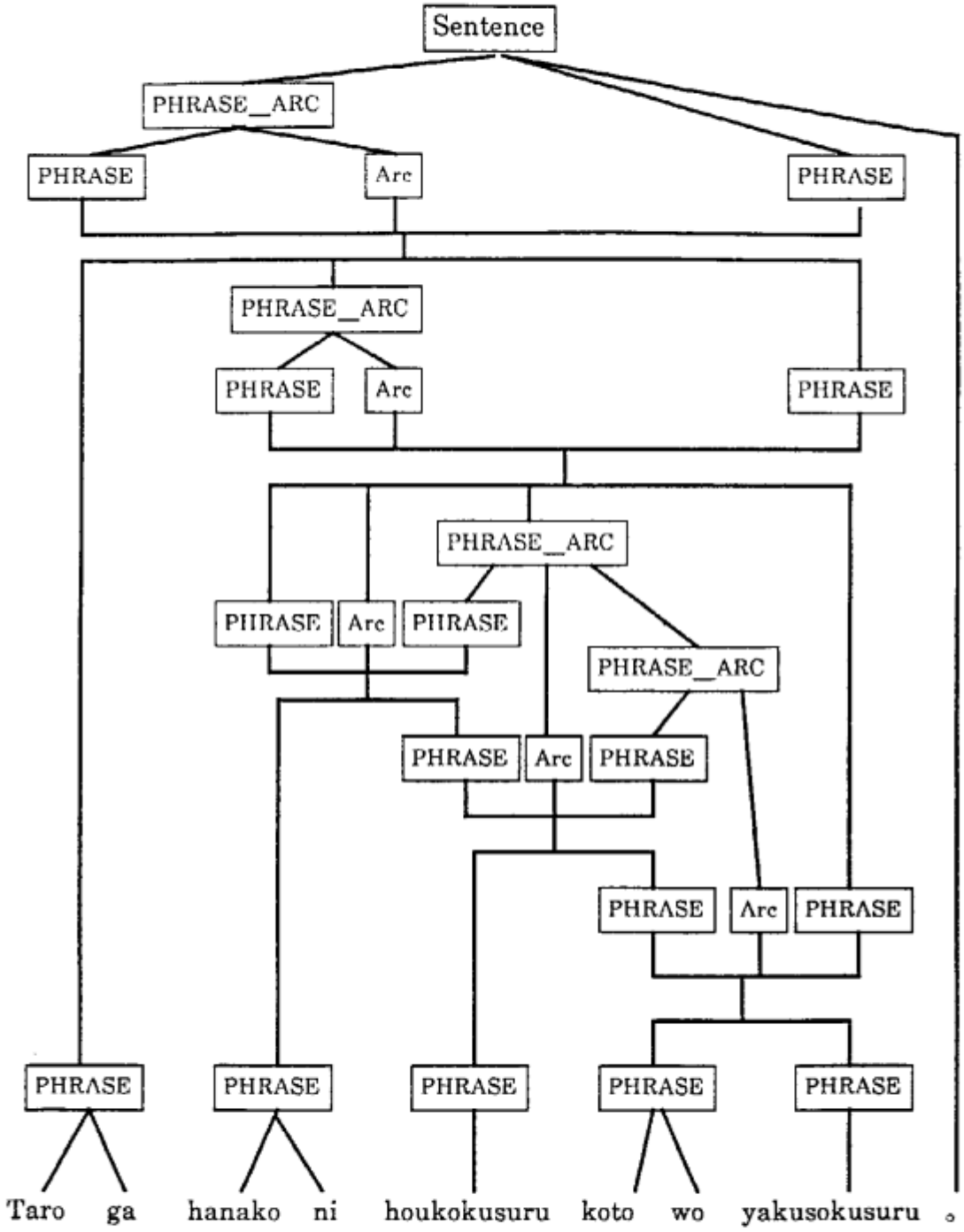

Figure 3: Parsing Chart

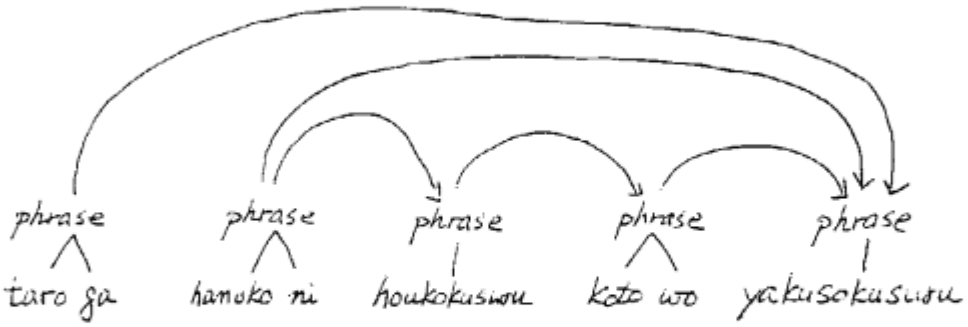

Figure 4: Resultant dependency structure

constraint could be solved by Boolean constraint solver generally as such Boolean Gröbner base solver.

$$arc(ph_3, ph_4) \wedge arc(ph_4, ph_5) \quad = \quad true \tag{22}$$

Truth value of the others are remains unknown. But if we can get the information from the context, that arc(ph_2,ph_5) is not true we can easily disambiguate the resultant dependency trees.

5 Conclusion

Declarative rules for dependency structures, and basic methodo for constraint analysis was presented. As the next research point, we should study about pragmatic facts which relates sentence disambiguation. Anyway, we believe the method presented here to be a first step for formal approach for dependency analysis and its disambiguation.

References

[1] V. Dahl. "More on Gapping Grammar", In *Proceeding of the International Conference on FGCS '84*, page 669–677, Tokyo, 1984

[2] V. Dahl and H.Abramson. "On Gapping Grammar", In *Proceeding of 2nd ICLP*, page 77-88, Sweden, 1984,

[3] Gunji, T., "Japanese Phrase Structure Grammar", D.Reidel Publishing Company, Dordrecht, 1987

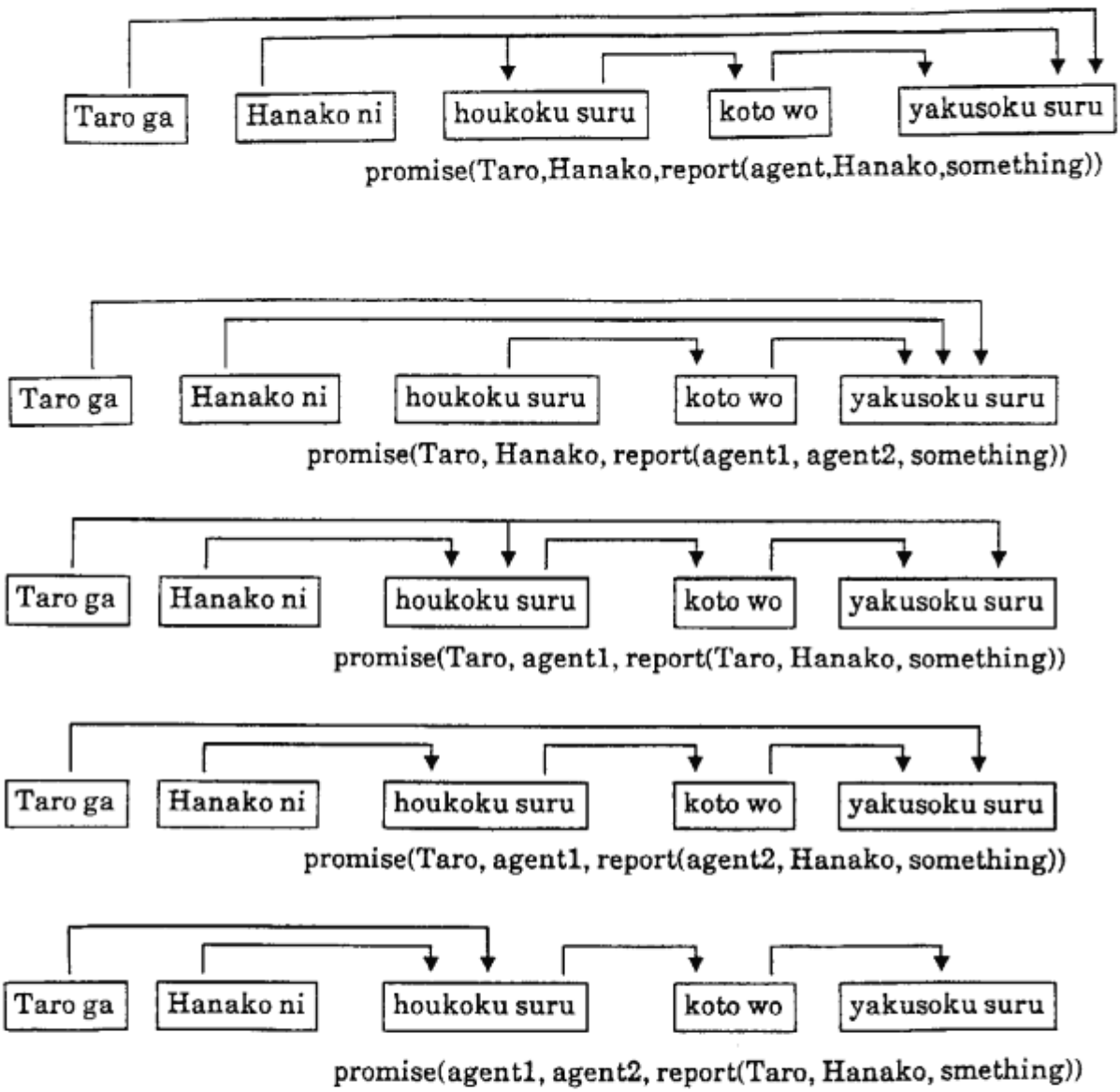

Figure 5: Resultant dependency structure

[4] Matsumoto, Y. and Sugimura, R. "A Parsing System Based on Logic Programming", In Proceeding of IJCAI 87, 1987

[5] Nakamura, J., "Solutions for Problems of MT Parser – Methods used in Mu-Machine Translation Project", Coling '86, pp.133–135, 1986.

[6] Nitta, Y. "Idiosyncratic Gap, A Tough Problem to Struture-bound Machine Translation", Coling '86, pp. 107–111, 1986

[7] Sugimura, R. "Constraint Analysis on Japanese Modification", Natural Language Understanding and Logic Programming, II, V.Dahl, and P.Saint-Dizier (ed), North-Holland, pp.93–105, 1987

[8] Sugimura R., Hasida K., Akasaka K., Hatano K., Kubo Y., Okunishi T. and Takizuka T., "A software environment for research into discourse understanding systems", FGCS-88, 1988