STEPS TOWARD AN ACTOR-ORIENTED INTEGRATED PARSER

KUNIAKI UEHARA, RYO OCHITANI
OSAMU MIKAMI, JUNICHI TOYODA

THE INSTITUTE OF SCIENTIFIC AND INDUSTRIAL RESEARCH
OSAKA UNIVERSITY
8-1 MIHOGAOKA, IBARAKI, OSAKA 567 JAPAN

ABSTRACT

The aims of this paper is to show the formalism of an 'Integrated Parser' (IP) for text understanding, and to discuss some of the advantages and disadvantages arising from our integrated approach. IP runs as a single module: syntactic analysis, semantic analysis, and contextual analysis occur as an integral part of the parsing process. Our important claim is that understanding sentences in text should involve the interaction between general linguistic knowledge (i.e. syntactic, semantic, and linguistic knowledge) and episodic knowledge. Another our claim is that we view IP's control structure as message passing in Actor theory. That is, we correspond each non-terminal of the grammar to an actor. Computation during parsing is, thus, performed only by sending messages among actors. Our parsing mechanism offers simplicity, perspicuity, modularity, no side effects and a simple but powerful computational semantics.

1. INTRODUCTION

Prolog (Pereira 1983) is a programming language which has a unique feature that programs written in it can be read either procedurally or declaratively as Horn clauses of first-order predicate logic. Since the syntax of Prolog is a natural extension of that of Context-Free Grammar (CFG), several Prolog-based grammar formalisms have been proposed for natural language processing. In fact, a Definite Clause Grammar (DCG) (Pereira et al. 1980) and PAMPS grammar (Uehara et al. 1984) are the simplest formalisms among them. These grammars, however, have certain deficiencies to describe natural languages.

First, as a high level formalism for natural language analysis, it is awkward in a DCG and PAMPS grammar to define new kinds of abstract data structures that are independent of the physical representation of their data. The data structures of them are restricted to those that can be represented simply by terms and lists.

Second, these grammar formalisms seem to pay little attention to understanding groups of sentences, either in text or in dialogue, focusing instead on understanding single sentences. Only the system proposed by Simmons and Chester (Simmons et al. 1982) presented a primitive approach for using Horn clauses (i.e. Prolog programs) to relate successive sentences into a discourse structure. The system first maps each sentence into its syntactic structure, then translates it into its semantic structure, and finally extracts the relationship between these semantic structures. In this sequential process, semantic context is not utilized to provide the syntactic component with information that can help it along. If input sentences include some forms of deletions, ellipses, and anaphoric references, it is difficult to determine their correct meanings during parsing at the isolated sentence level.

This paper presents the formalism of an 'Integrated Parser' (IP) for text understanding which runs as a single module: syntactic analysis, semantic analysis and contextual analysis occur as an integral part of the parsing process. This integrated scheme follows the idea that not only general linguistic knowledge (i.e. lexical, syntactic, and semantic knowledge) but also episodic knowledge contained in the text must be used to parse sentences in the text. Consider, for instance, the classical example of anaphora: "The soldiers fired at the women and we saw several of them fall." To find the anaphoric referent of "them", it does not suffice to make use of only general linguistic knowledge since both soldiers and women can fall. We use the idea of both 'prediction' and 'requirement' to aid in resolving ambiguous references.

The formalism of an IP grammar is based on that of a Lexical Functional Grammar (LFG) introduced by Kaplan and Bresnan (Bresnan 1982), which was grown out of ideas from

current transformational linguistics and computational linguistics. An LFG consists of both standard context-free rules and lexical items with associated functional equations (schemata). These schemata are finally solved to produce a functional structure (f-structure), that is, a deep structure of an input sentence.

Solving the schemata can be seen as if it transfers partial f-structures from node to node upward in accordance with the growth of a parse tree. We can view this control structure as message passing in Actor theory proposed by Hewitt (Hewitt 1977). That is, we can correspond each non-terminal in the context-free rule to an actor. Computation during parsing is, thus, performed only by sending messages among actors. Another our claim is that there are three distinct metaphors available in the computational linguistics society today; procedure oriented (i.e. PROGRAMMAR (Winograd 1972) and ATN (Woods 1970)), declaration oriented (i.e. DCG and PAMPS) and actor oriented metaphors (i.e. Object-Oriented Parser (Phillips 1983) and Word Expert Parser (Small 1981)). These metaphors are usually embedded in completely separated grammar formalisms. IP was designed to incorporate all three metaphors within a single grammar formalism so as to allow users to write practical grammars with ease.

A parser is said to be an on-line parser if it parses each prefix of 'w' ('w' is an input sentence) before reading any of the input beyond the prefix. A parser which is not an on-line parser is called an off-line parser. Inherently, the original LFG is formulated for an off-line parser. The last our claim is that the LFG-formalism can be re-formulated to be suitable for an on-line parser. IP applies grammar rules and constraints simultaneously, and builds up f-structures during a single parsing process.

Our parsing mechanism offers simplicity, perspicuity, modularity, no side effects and, furthermore, a simple but powerful computational semantics. IP has been implemented in C-Prolog (Pereira 1983) and run on an ECLIPSE MV/8000-II.

2. AN IP GRAMMAR

The following introduction is fairly self-contained, but the readers are expected to be familiar with the basic concepts and notations of LFGs (Bresnan 1982).

2.1 Data Structure

In IP, the grammatical relation of an input sentence is represented by a functional structure (f-structure). The f-structure consists of a set of ordered pairs each of which consists of an attribute and a specification of the attribute's value for the input sentence. The f-structure is composed of grammatical function names, semantic forms, and feature symbols. The f-structure is of the form:

$$\begin{bmatrix} \langle attribute\_1 \rangle = \langle value\_1 \rangle \\ \vdots \qquad\qquad \vdots \\ \langle attribute\_k \rangle = \langle value\_k \rangle \end{bmatrix}$$

An attribute is the name of a grammatical function or a feature. A value is either a symbol, a semantic form, an f-structure, or a set of symbols, semantic forms, or f-structures. A symbol is a primitive type of attribute's values. There are three kinds of semantic forms. One is called an event, which is treated as a pattern for composing the logical formulas encoding the meaning of an input sentence. The event comprises a predicate name followed by a sequence of one or more arguments. A predicate name is characterized for representing its sense or meaning. An argument will be assigned to a partial f-structure which specifies the grammatical function of its thematic role. The second type of semantic forms is called an individual. Individuals are also semantic forms, but do not have any argument. The last type of semantic forms is called a class, which specifies a set for describing individuals of a particular kind. A class corresponds to a common noun in the traditional linguistics. We assume a number of predicate names --- e.g., stay-in, reach, get-on, corresponding roughly to English words, which are not so rigidly categorized as primitive acts and primitive states in Conceptual Dependency theory (Schank 1977), an arbitrary number of individuals --- e.g., John, New-York, and a number of classes --- e.g., window, book. The semantic form is recorded in its corresponding lexical item and is carried along by the f-structure.

$$\begin{bmatrix} \text{subject} = \begin{bmatrix} \text{num} = \text{singular} \\ \text{ind} = \text{john} \end{bmatrix} \\ \text{tense} = \text{past} \\ \text{event} = \text{open}\left(\begin{bmatrix} \text{num} = \text{singular} \\ \text{ind} = \text{john} \end{bmatrix}, \begin{bmatrix} \text{num} = \text{singular} \\ \text{class} = \text{window} \end{bmatrix}\right) \\ \text{object} = \begin{bmatrix} \text{num} = \text{singular} \\ \text{class} = \text{window} \end{bmatrix} \end{bmatrix}$$

Fig.1 An f-structure.

For example, 'subject' and 'object' in

Fig.1 are grammatical functions, and 'tense' is a feature. 'Singular' and 'past' are symbols. 'Open([...],[...])' is a semantic form and the f-structure '[num = singular; ind = man]' is its argument.

The data structure of IP bears some similarities to that of Augmented Phrase Structure Grammar (APSG) (Heidorn 1975), which Heidorn took as a basis for the text processing system EPISTLE. Like an f-structure, the data structure used by APSG consists of 'records' which are collections of attribute-value pairs. It seems fair to say that the record can be viewed as a special case of the f-structure. In another point of view, the data representation technique by use of f-structures is similar to slot-filler technique which is used in Frame theory proposed by M. Minsky. The fillers of slots can be associated with attribute values, whereas slot identifiers with attribute names.

2.2 The Syntax of an IP Grammar

An IP grammar has two kinds of rules: augmented context-free rules (hereafter we will simply say grammar rules) and lexical items. F-structures are generated by schemata associated with grammar rules and lexical items. A set of schemata, which are preceded by the non-terminal, is of the form:

[<non-terminal> (<schema_1>, ... ,<schema_k>)]

A schema is either a defining schema, which defines the value of some feature, or a constraining schema, which constrains a feature whose value is expected to be defined by a separate specialization.

A defining schema is either of the form:

<designator> = <designator> (1)

or

<designator> << <designator> (2)

(1) is an identification schema which expresses that the f-structure indicated by the left-hand side is equal to the f-structure indicated by the right-hand side. (2) is a membership schema which expresses that the f-structure indicated by the right-hand side is a set containing the f-structure indicated by the left-hand side.

A constraining schema is of the form:

<designator> == <designator> (3)

<designator> (4)

not(<designator>) (5)

(3) is an equational constraint which constrains that the f-structure of the right-hand side should be equal to the f-structure of the left-hand side. (4) is an existential constraint. The existential constraint is satisfied when the expression has some value. (5) is a negative constraint formed by adding a negation operator to a constraining schema. The negative constraint is acceptable only if the constraining schema without the negation operator is false. The evaluation of constraining schemata whose values are not determined is postponed until the schemata have gotten their values (delayed computation), which will be discussed in section 3.2.

A designator is either of the form:

(<meta-variable> <designator>)

or

(<meta-variable> <symbol_1>, ... ,<symbol_n>)

A designator consists of a meta-variable followed by another designator or one or more symbols. Meta-variables are of just two types:

<- , -> i)

<= , => ii)

A meta-variable '<-' refers to the f-structure attached to the left-hand side non-terminal of a grammar rule. A meta-variable '->' refers to the f-structure attached to the non-terminal where the meta-variable '->' itself appears. Meta-variables '<=' and '=>' are used to characterize the 'long-distance' dependencies found in relative clauses and questions. The treatment of the meta-variables will be explained in section 3.2.

Consider, for instance, the following grammar rule:

[S [[NP (<-subject=->)],[VP (<-=->)]]]

From the point of view of the S-dominated NP node, the schema '<-subject=->' says that the value of the subject attribute of S is the f-structure of NP. The schema '<-=->' says that the f-structure of S is equal to the f-structure of VP. In the original LFG, every f-structure must satisfy

the additional conditions (i.e. uniqueness, completeness, and coherency conditions) in order to get the correct meaning of a single sentence. In IP, however, semantic forms are used not only as the meanings of single sentences, but also as the semantic context of successive sentences in a text, these conditions have to be somewhat relaxed so as to accept the structures whose arguments are unspecified during the parse of a single sentence because of ellipsis and anaphora. We will postpone our discussion of the problem to section 6.

3. THE CONTROL STRUCTURE OF IP

In Actor theory, an actor is organized as a computational entity which has aspects of both procedures and data. Actors are not primarily partitioned into procedures and separate data. All of the action of an actor comes from passing messages between actors. Actors interpret the uniform message form locally. The actor oriented paradigm is well suited to applications where the description of entities is simplified by use of uniform protocols. If we are adding the idea of Actor theory to the grammar formalism, we can produce a better practical way of writing complex grammars.

In IP, the augmentations were added to CFGs, which are conditions (i.e. constraining schemata) and actions (i.e. defining schemata) associated with each rule of the grammar. These augmentations can be reduced to a single uniform mechanism, that is, message transmission among actors and the behavior of actors. Viewing parsing as passing messages among other actors, we can construct a new parsing control structure, which offers us modularity, perspicuity, no side effects, and simple but powerful semantics.

3.1 An Actor

There are two types of actors in IP; one is an actor corresponding to a non-terminal, the other is an actor corresponding to a terminal. A non-terminal actor consists of two parts: a 'script' (or action) which describes what should be done when the actor receives a message, and a set of 'acquaintances' which are the other actors that the actor knows about.

The non-terminal actor is of the form:

[<actor_name> <script>]

where <actor_name> is the left-hand side of a grammar rule. A 'script' part shows a schema which indicates how to construct an f-structure. <Script> consists of a set of right-hand sides of the grammar rules whose left-hand sides are the same non-terminal. Each element of the script is called a pattern. The script is of the form:

[[<pattern_1>],
 .
 .
 .
[<pattern_n>]]

<pattern_i> is of the form:

[<non-terminal_1> (<schemata_1>)]
 .
 .
 .
[<non-terminal_k> (<schemata_k>)]

where the first element of the sequence, '[<non-terminal_1> (<schemata_1>)]', will be called a head, whereas the sequence except the first, '[<non-terminal_2> (<schemata_2>)] continuation.

An 'acquaintance' part shows a set of 'reachable' actors. A non-terminal actor, A, is said to be reachable from a non-terminal actor, R, if there is a derivation tree of R which has A on its left branch. When a set of grammar rules and lexical items is stored in the system, reachability relationship between non-terminals is pre-computed before parsing. After the pre-computation, all the non-terminals reachable from a non-terminal are recorded in the acquaintance part of the non-terminal actor, though it is not shown in a visual way. If there are a lot of applicable patterns, some of which would turn out to be useless, reachability relationship is useful to restrict the number of applicable ones to be considered next. The idea of an 'acquaintance' is very similar to that of 'top-down filtering' used in PAMPS.

A terminal actor does not have any acquaintance, though it has a script which specifies its syntactic features and semantic forms in terms of patterns. The terminal actor is of the following form:

[<actor_name> [<pattern_1>
 .
 .
 .
 <pattern_m>]]

where <actor_name> is a terminal.

3.2 Activated Patterns and Inactivated Patterns

In Actor theory, communication between actors are assumed to be done in parallel. However, since achieving parallelism within our implementation would consume more space during parsing and require more difficult implementation technique, IP parses sentences in a traditional top-down serial way with automatic backtracking. If a non-terminal actor has some applicable patterns, one of them is chosen, evaluates the schemata of its head, and sends its continuation to the non-terminal actor of its head, the selected pattern is called an activated pattern. All the other patterns are turned out to be inactivated and wait to be executed until the chosen one fails to be sent to another actor. These patterns are called inactivated patterns. There are two cases that the parsing fails to proceed; one is the case that the actor cannot construct a proper f-structure of the input sentence, the other is the case that there are no more patterns waiting to be activated. In these cases, the most recently activated actor abandons the computation made by transmitting a message, chooses an alternative among inactivated patterns, and evaluates the pattern.

3.3 A Message

Once an actor, A, transmits a message to another actor, B, the computation proceeds by following the script of B using information from A. To do this, a message must have fairly rigid form. This provides the basis for meaningful communication between actors. The message consists of five parts:

1) F-structure: A partial f-structure constructed during passing messages among actors.

2) Trail list: Trail list is used as a push-down list. It is used to store the names of actors which need to be re-activated on backtracking.

3) Continuation: A set of non-terminal-schemata pairs. When a message is transmitted to an actor, the actor picks up the first constituent from the continuation, and evaluates its schemata.

4) Hold list: It is used to analyze the phenomenon which in generative transformational grammars would be called left extraposition. A set of f-structures of extraposed constituents is recorded in the hold list. This enables the actor to simulate the HOLD-VIR mechanism in an ATN.

5) Constraining schema list: Constraining schemata whose meta-variables could not be instantiated so far are recorded in the constraining list. When an actor receives a message, it always checks to see if the elements of the constraining schema list are computable or not. This list makes the actor have the power to perform the delayed evaluation.

4. TEXT UNDERSTANDING IN IP

Recently there has been much effort to pay attention to the whole text instead of individual sentences. A number of linguists have investigated to capture the relations that link sentences in conversations and in texts. These have variously been called 'coherent relations' (Hobbs 1979), 'causal chains' (Schank 1977), and so on. We will, hereafter, go into the discussion about text understanding, especially the extraction of coherent relation and disambiguation of anaphoric references.

An event, which was mentioned above, can express states and actions in text. We assume that actions are defined as intentional human acts that change from one state to the next in some way that would not have come about otherwise. Schank and his colleagues have worked out the causal syntax about actions and states. Some of the causal rules are:

1) Actions can result in state changes.

2) States can enable actions.

3) Actions can enable other actions.

4) States can causes state changes.

These rules were slightly simplified from the original one, since we will not deal with mental events.

According to these rules, we begin with the premise that when one reads a sentence and understands its meaning, one can easily make the prediction about what kind of events are coming next, which is called a predicted event. That is, we assume that the next sentence can be predicted from the current sentence by use of the above rules. Conversely, in order to extract the event from a current input sentence, some requirements, which are called a pre-required events, must be fulfilled by the sentences that have already been read. Both 'prediction' and 'requirement' are encoded implicitly encoded within the structure of a lexical item, rather than encoded explicitly within a separate inference module, such as

common sense inference rules in Wilks's machine translation system (Wilks 1975). Fig.2 shows a schematic view of a verb-type lexical item.

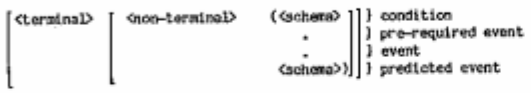

Fig.2 A schematic view of a verb-type lexical item.

The condition part consists of a set of constraining schemata which are conditions for the application of a lexical item. Lexical items are highly specific and a single word may have several different entries corresponding to its various meanings. Conditions are used to select the correct meaning depending on the context where the word appears. The pre-required event part includes some pre-required events which must have been satisfied previously. The event part specifies the semantic form corresponding to each meaning of the word. The predicted event part contains the predicted events which will be fulfilled after the analysis of the current sentence. In Fig.3, we show some examples about these events.

(1) John went to pet shop.

```
pre-required_event=come(john,(<-place),pet_shop)
event=go(john,pet_shop)
predicted_event=stay_in(john,pet_shop)
```

(2) There he bought a dog.

```
pre-required_event=stay_in(he,pet_shop)
event=buy(he,dog)
predicted_event=pay(he,(<-obj),money)
                get(he,dog)
```

Fig.3 Events extracted from sentences.

Sentence (1) has the predicted event that John stays in the pet-shop, and the pre-required event that John came to the pet-shop from somewhere. Sentence (2) has the pre-required event that John stays in the pet-shop and the predicted events that John pays some money to get a dog, and that he gets it.

Fig.4 shows the lexical item of the word 'eat'.

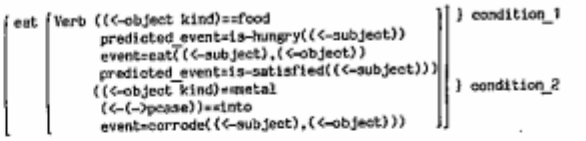

Fig.4 The lexical item of the word 'eat'.

In this example, the lexical item of 'eat' is highly specialized. It is applicable to the use of 'eat' in sentences like 'The man eats an apple' where the condition part of the word 'eat' should contain the condition that the object must be a kind of food. The pre-required event indicates that the subject is-hungry. The predicted event says that the subject is-pleased. The event means that the subject eats the object. Though, to analyze the sentence 'Acids eat into metals,' the condition part must contains the condition that the object must be a metal. The event means that the subject corrodes object.

Assume that S1 refers to the sentence currently being processed, S0 to be a previous one, and S2 to be a next one. If the event of S0 can pattern-match with the pre-required event of S1 or the predicted event of S1 can pattern-match with the event of S2, then we will say that S0 (S1) has the coherent relation to S1 (S2). In the example of Fig.3, the predicted event of (1) can pattern-match with the pre-required event of (2), thus both sentences are coherent.

5. ANAPHORA AND ITS REFERENT

Events are also useful to deal with simple cases of anaphora and ellipsis. Even if S0 and S1 are coherent by a continuity of state-action tracks understood in terms of events, some arguments in events may be partially undefined or omitted. These undefined parts occur because of anaphora and ellipsis. To understand a text, we must instantiate these undefined parts. The values of these arguments may be determined when the event pattern-matches against the predicted event.

Consider the text:

1) John took a train to New York.
2) He got off it at Grand Central.

From sentence 1), we can infer the predicted event:

```
get_off([num=sg          ],[def=infinite ],[ind=New_York ])
        [class=john      ] [class=train  ] [concept=place]
        [sex=male        ] [num=sg       ]
        [concept=human   ] [concept=cargo]
```

The event of sentence 2) is:

```
get_off([num=sg          ], [num=sg] ,[ind=Grand_Central])
        [sex=male        ]            [concept=place     ]
        [concept=human   ]
```

From these events we can find that the word 'he' should be identified as referring to 'John'. The second argument of the event must be:

```
[def=infinite ]
[class=train  ]
[num=sg       ]
[concept=cargo]
```

In order to pattern-match the predicted event to the event, IP must know the fact that Grand Central is located in New York. This mechanism can be performed simply by associating a hierarchy with actors (property inheritance). This hierarchy supports a very convenient form of default reasoning and increases the brevity and modularity of an IP grammar. In addition, it should provide a more general and powerful approach to resolving anaphora and ellipsis. We must leave to a separate paper the details of this 'fuzzy' unification algorithm.

Consider the following example.

The soldiers fired at the women and
we saw some of them fall.

After the analysis of the first sentence, we can get the following events:

```
pre-required_event=have([def=definite ], [concept=weapon])
                        [noun=soldier ]
                        [num=plural   ]
                        [concept=human]

event=fire_at([def=definite ],[def=definite ])
              [class=soldier] [class=woman  ]
              [num=plural   ] [num=plural   ]
              [concept=human] [sex=female   ]
                              [concept=human]

predicted_event=injury([def=definite ])
                       [class=woman  ]
                       [num=plural   ]
                       [sex=female   ]
                       [concept=human]
              die([def=definite ])
                  [class=woman  ]
                  [num=plural   ]
                  [sex=female   ]
                  [concept=human]
              fall([def=definite ])
                   [class=woman  ]
                   [num=plural   ]
                   [sex=female   ]
                   [concept=human]
```

The event of the second sentence is:

```
event=see([num=plural ],[num=plural],(<-vcomp_event))
          [concept=human]

vcomp_event=fall([num=plural])
```

One of the predicted events of the first sentence can pattern-match with the vcomp-event of the second. Thus, we can easily infer that 'them' refers to 'women'. Note that the proper treatment of quantified noun phrases, such as 'some of them', is difficult to handle semantically. We will, therefore, not go into the analysis of the phrase.

Dealing with ellipsis is slightly harder than dealing with anaphora, since the ellipsis handling needs a special grammar which can treat extra-grammatical utterances as in (Kwasny 1981). Therefore we will not go into anymore detail.

6. A MORE COMPLICATED EXAMPLE

First of all, We show a simple text in Fig.5. The story was taken from the Japan Times of December 1, 1981.

(1) Mrs. Miura was shot in Los Angeles on Nov. 18, 1981.
(2) Mrs. Miura was wounded in the head.
(3) She entered the university hospital.
(4) She received intensive medical treatment there.
(5) The woman had not recovered consciousness since the incident.
(6) She eventually died of a lesion of the brain without regaining consciousness.

Fig.5 A simple text.

Fig.6 indicates the IP grammar for the simple text.

```
[sentence [(s          ( <- = -> )] .          ]]
          [(endsymbol  ( <- = -> )]            ]]
[s        [(np         ( <-subj = -> )] .      ]]
          [(vp         ( <- = -> )] .          ]]
          [(ppr        ( <-adjunct = -> )]     ]]
[np       [(lm         ( <-left_mod = -> )] .  ]]
          [(noun       ( <- = -> )]            ]]
( np      [[(noun      ( <- = -> )]]]]
[vp       [(v          ( <- = -> )] .          ]]
          [(vp1        ( <-vcomp = -> )]       ]]
( vp      [[(v         ( <- = -> )]]]]
( vp1     [[(vp        ( <- = -> )]]]]
( lm      [[(adj       ( <- = -> )]]]]
( ppr     [[(pp        ( <-(->pcase) = -> )]]]]
[ppr      [(pp         ( <-(->pcase) = -> )] .  ]]
          [(ppr        ( <- = -> )]             ]]
[pp       [(prep       ( <- = -> )] .           ]]
          [(np         ( <- = -> )]             ]]
```

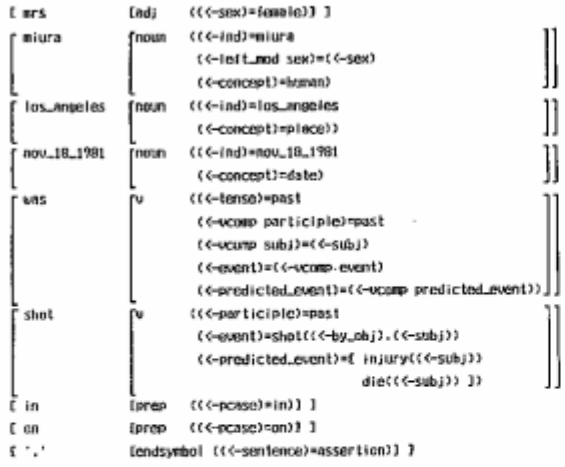

Fig.6 Examples of the IP grammar.

Fig.7 shows an f-structure of sentence (1) in the text.

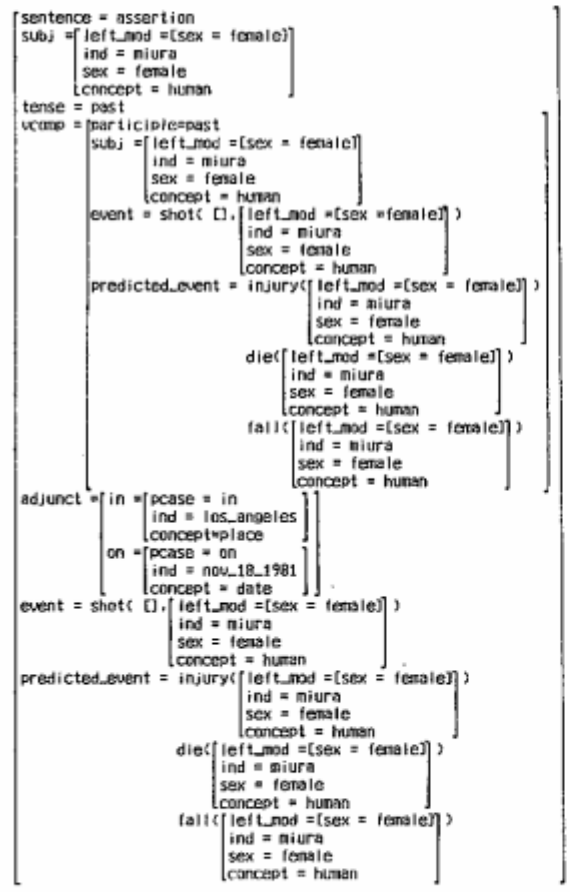

Fig.7 An f-structure of sentence (1).

Fig.8 shows the representation of the coherent relation among six sentences. Sentence (1) has the event which means that Miura was shot by someone, and three predicted events that Miura was injured, that Miura was dead, and that Miura fell. Sentence (2) has the predicted event which means that Miura was injured. The predicted event of sentence (1) can pattern-match with the event of sentence (2), thus we can find that these two sentences are coherent. Both pre-required events of sentence (3) can pattern-match with the event of sentence (2) and the predicted event of sentence (3), respectively. We can, thus, find that the word 'she' refers to 'Miura'.

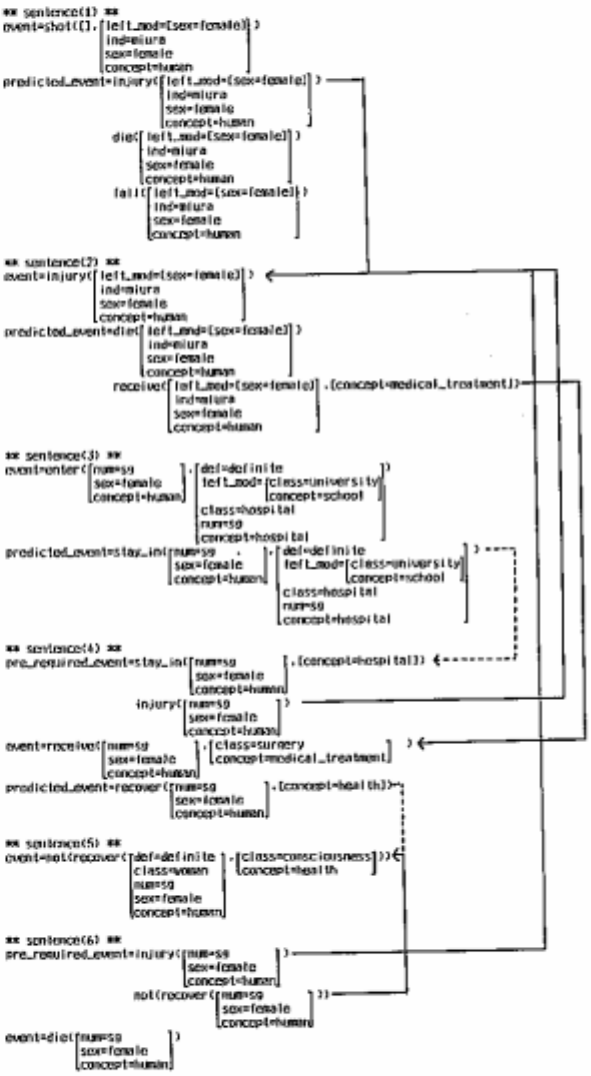

Fig.8 Representation of the coherent relation in the text.

7. IP GRAMMARS AND SOME RELATED GRAMMARS

Now we shall discuss about the relationship between IP grammars, APSGs and DCGs.

First, these grammars are based on the same grammar formalism, i.e. CFG. In the course of parsing of IP grammars and DCGs, non-terminals are replaced by the sequence of non-terminals that are matched by unification, though the unification process of IP is quite different from unification in the concept of definite clause logic. An APSG uses explicit constructors and selectors instead of unification.

Second, in DCGs, the skeleton of a structure (such as a parse tree) has already been determined before parsing proceeds, since each parse tree is specified by a compound term, whose name characterizes the top node of the parse tree and arguments indicates the children of the node. Variables show places where undetermined structures of the parse tree are assigned. IP, on the other hand, has the f-structure whose skeleton can be shown only when the parsing is finished. Its undetermined sub-structures are completely omitted from the f-structure. Furthermore, the order of the children of DCG's structure must be specified in a visual way, since the unification mechanism proceeds in order of the sequences of arguments. Whereas, in a IP grammar and APSG, it is not necessary to pay attention to the order of the children.

Third, the structure of a DCG is more or less described in terms of functional notation (i.e. compound terms) or list notation. The structure of IP is in effect a hierarchy of a set of ordered pairs each of which consists of an attribute and the value of the attribute. As was mentioned before, APSGs have the similar data structure named 'records'. It seems to be fair to say that a record is a special case of an f-structure.

Finally, IP grammars and DCGs parse sentences in a left-to-right, top-down, serial way. APSGs use a strictly left-to-right, bottom-up, parallel-processing algorithm in which all rules that can be applied to an input sentence at a particular time are applied. Certain deficiencies with DCGs have been inherited by the IP grammar. For example, in the process of backtracking it is difficult to describe default behavior. Debugging in DCGs is very difficult for lack of a distinction between failures and errors.

ACKNOWLEDGEMENTS

We would like to thank Professor Osamu Kakusho, who has been a source of constant support of our research.

REFERENCES

1) Bresnan, J. (ed.): The Mental Representation of Grammatical Relations, MIT Press (1982).
2) Heidorn, G.E.: Augmented Phrase Structure Grammars, in B.L. Nash-Webber and R.C. Schank (eds.), Proc. of Theoretical Issues in Natural Language Processing, ACL (1975).
3) Hewitt, C.: Viewing Control Structures as Patterns of Passing Messages, Artif. Intell., Vol.8, No.3, pp.323-364 (1977).
4) Hobbs, J.R.: Coherence and Coreference, Cognitive Science, Vol.3, No.1, pp.67-90 (1979).
5) Kwasny, S.C.: Relaxation Techniques for Parsing Grammatically Ill-Formed Input in Natural Language Understanding Systems, AJCL, Vol.7, No.2, pp.99-108 (1981).
6) Pereira, F.C.N. et al.: Definite Clause Grammar for Language Analysis, Artif. Intell., Vol.13, No.3, pp.231-278 (1980).
7) Pereira, F.C.N.: C-Prolog User's Manual, Version 1.2a, EdCAAD (1983).
8) Phillips, B.: An Object-Oriented Parser for Text Understanding, Proc. of the 8th IJCAI, pp.690-692 (1983).
9) Schank, R.C.: Scripts, Plans, Goals, and Understanding, Lawrence Eribaum (1977).
10) Simmons, R.F. et al.: Relating Sentences and Semantic Networks with Procedural Logic, Comm. of the ACM, Vol.25, No.8, pp.527-547 (1982).
11) Small, S.: Viewing Word Expert Parsing as Linguistic Theory, Proc. of the 7th IJCAI, pp.70-76 (1981).
12) Uehara, K. et al.: A Bottom-up Parser Based on Predicate Logic, Proc. of the 1984 International Symposium on Logic Programming, pp.220-227 (1984).
13) Wilks, Y.: An Intelligent Analyzer and Understander of English, Comm. of the ACM, Vol.18, No.5, pp.264-274 (1975).
14) Winograd, T.: Understanding Natural Language, Academic Press (1972).
15) Woods, W.A.: Transition Network Grammar for Natural Language Analysis, Comm. of the ACM, Vol.13, No.10, pp.591-606 (1970).