Semantic Analysis of a Declarative Language Based on Knowledge Representation

Semantic Analysis of a Declarative Language Based on Knowledge Representation

Hiroyoshi BABA, Ichiro SUGIOKA

Abstract

The objective of this research is analyzing and understanding the deep machanism of a declarative language, for example Prolog, by adopting time-state, hypothetical and multi-universe inferences in such a symbolic processing. We coded the analysis system in Automated Reasoning Tool (ART*) and LISP. The main part of the system is separated into three parts, namely, the first part is 'syntax tree construction', the second part is 'attribute evaluation' and the final part is 'viewpoint network simulation'. The input to the system is source codes of Prolog. Then, after passing through inside the system, the result of processing is outputted. The result means the simulated variable at each stage, that are temporary, hypothetically and hierarchy.

This paper describes the peculiarity of declarative language, the methodologies for representing incomplete knowledge that related to the programming of our system. Then, about the implementation of the syntax and semantic analyzer, some considerations regarding the system are described.

1. Introduction

In the researches of the field of artificial intelligence (AI), it was since the last decade that some AI researchers have started to study knowledge representation including the concepts of temporal processes.

Compilers, in the other words, language processors, consists of two parts -- an analysis part and a synthesis part. In the analysis part, it is necessary to formalize the semantics of the languages for their semantic analysis and context analysis. Usually, semantic analysis are divided into static analysis and dynamic analysis. The former plays a role of verifying the correspondence between the usage of names and their declarations, inspecting the information of types, and checking whether each definition is duplicated or not. To fulfill such analysis, the attribute grammar of each programming language are basically introduced. But we still don't have the unified methodologies for the latter, dynamic analysis. Theoretically, an attribute grammar is also able to analyze the semantics of programming languages dynamically, but we tried from a different direction -- simulating the processes of a program to apply temporal, hypothetical and multi-universe reasoning. Partly because temporal inference is related to nonmonotonic reasoning while hypothetical one related to monotonic reasoning, they are the main topics of the recent researches of knowledge

*ART is the trademark of Inference Corp.

representation.

Monotonic implies that the number of facts in the database is always increasing, but nonmonotonic is not always increasing.

As concerns knowledge, there are four different kinds of ambiguities : Uncertainty incompleteness, polysemy and fuzziness. In our study, we focused on specially the incompleteness of knowledge to analyze the language processing. In incomplete knowledge, it is possible to be denied previous true facts concluded by some inferences at the time new facts are loaded in a knowledge base. So this situation has nonmonotonic characteristics. Some AI researchers have been studying the logic for incomplete knowledge, also the other establishing dynamic inference systems that can update a set of the truth in a knowledge base in real time (or, in each step of changing conditions), effectively. The representatives of the former are nonmonotonic logic and a default reasoning, and the ones of the latter are a truth maintenance system, and an assumption-based truth maintenance system. We put the essence of their concepts in the analyzer.

There are many classes in attribute grammars. Originally, an attribute grammar stemmed from an context free grammar as its semantics. Even though many researchers presented their theories and systems using an attribute grammars, it is hard to find the relationship between non-procedural languages, like Prolog or LISP, and their attribute grammars. Under such a situation, we decided to attempt analyzing Prolog language. Prolog is one of the non-procedural, namely, declarative languages. In analyzing Prolog symbolically, we considered its attribute grammars from the context free grammars as the processes of static semantic consulting.

2. Declarative Language

There are two different categories in the programming languages -- a procedural language and a declarative language. In a procedural language like FORTRAN and Pascal, programmers show exact steps of processing. In other words, they have to implement HOW to solve problems in their programs. On the contrary, we just write codes WHAT to solve programs in a declarative language like Prolog. A declarative language is sometimes called a relational language as they only describe the relationship among different predicates and clauses.

Nondeterminism is another peculiarity of a declarative language. It is the theoretical concept for the definition of abstract calculus models. Intuitively, a true nondeterministic machine is the one that can choose the next proper way at each alternative. We hardly realize such a machine truly, but we might simulate, or resemble its machanism. A logic programming model is a good tool for doing it, and the model of the programming language is declarative. Technically, a generate-and-

test method is the representative strategy for simulation. However, a don't-know nondeterminism is more difficult to figure out the right choice at each alternative than a don't-care nondeterminism as it's not always true to reach the goal through each division in a don't-know nondeterminism and the latter one is opposite : any different decisions never fail. The former nondeterminism has a limitation that we cannot find the way should be proceed to next at the each time.

Some simulation programs are suitable examples for application of the nondeterministic programming, and especially, to simulate the transition of an nondeterministic finite automaton (NDFA) is better instance for a don't-know nondeterminism. It is very easy to implement the interpreter for various kinds of automata in Prolog so that we programmed NDFA in the declarative language as a material for verification of our simulator.

We implemented the dynamic semantic analyzer for a declarative language as its simulation of a don't-know nondeterminism. We don't think it is valid to simulate the meaning of a procedural language simply because the language is quite deterministic -- the sequences of each different process has been determined in advance.

The nondeterminism of Prolog is deeply related to its recursion in the program. Sometimes it is very complicated the transitions of variables in Prolog because the both characteristics are involved tightly each other. Therefore, we constructively applied hypothetical reasoning to Prolog's nondeterministic characteristics, and simultaneously, we also did temporal and multi-universe reasoning to its recursive functions -- to inspect the language multiply -- or, in a sense, three dimensionally.

3. Knowledge Representation

In this chapter, mainly, the methodologies for representing an incomplete knowledge, and temporal reasoning are described and discussed briefly.

3.1 Frame Problem

In [1], one of the few philosophical problems in artificial intelligence was pointed out -- the frame problem. The frame problem implies such a truth that the quantities of the descriptions might be enormous and intractable in a trial to express all the changes of conditions in the world only by logic. This is an essential problem of the complexity, and the frame problem suggested that we describe or process only a partial information in a huge world. More concretely, the problem is separated into two classes : the frame problem for description and that for processing. The former is an approach to attract the best conclusion by describing the situation partially in incomplete information. The concept of complexity is related to both space and time, that is to say, pro-

cessing and description.

In [2], an another direct approach toward the frame problem, the unless operator, is described. This trial contains two ideas : the one is for properties and one for actions. In the former, an object retains its property until the property is explicitly changed. In the latter, an action is keeping until it is explicitly discontinued by another action.

3.2 Default Reasoning

A default type of reasoning is to formalize the inferences logically under an incomplete information. More concretely, it appoaches to expand the first order predicate logic so that the logic can accepts some exceptions and such an intuitive semantics. Default reasoning [3] and nonmonotonic logic [4] are the representatives of trials to deal with calculating the situation in incomplete knowledge from the side of logic approach. Both realize the impossibilities of proofs for some facts by an original modal symbol. The difference between them, however, is that in default reasoning, the sentences include its modal symbol appear only in the inference rules, while in nonmonotonic logic, the modal symbol itself is admitted as an formative element for logical expressions. Truth mainthenance system, it may be described later, inplies some methodologies how to realize such modal logical formalizing instead of the difficulties for apply their concept on a computer.

3.2.1 Default Reasoning and Nonmonotonic Logic

The default logic consists of a set of axiom and a set of default. It introduced a modal symbol, called consistent, for the inference rules. Default rules are supposed to indicate what conclusions to jump to.

In default reasoning, it is always possible that an inspection of its noncontradictory for judging whether a logic expression is a theorem or not fails into an infinite loop. It is because the inference rules show defaults unable to exist in the range of the countable and finite number. The serious defects of such an default type reasoning comes from the difficultly of talking into consideration the concept of time -- its mono-directional characteristics. We must consist the symbolic system and computational system that adapt the structure of our understandings under the temporal transition in a world. That is, the main problem is how to build the ontology of time and world into a logical machinery.

3.2.2 Truth Maintenance System

Truth maintenance system (TMS) realized to attract the facts as conclusion of default inference under incomplete knowledge, and to modify such conclusions dynamically and timely when they have been proved to be failures. In TMS, each fact has the attributes to show the condition : 'in' or 'out'. If a certain fact is believed at this time, its attribute is 'in', and if not, the attribute is 'out'.

This concept independents that the fact is true or false.

So the keypoint of default reasoning on a computer is how to figure out the default value caused a contradictory effectively and efficiently. Every fact if TMS consists of a node, a statement and a justification -- a node is a time tag for the fact and a justification contains the reasons of the knowledge.

3.2.3 Assumption-based Truth Maintenance System

An assumption-based truth maintenance system (ATMS) is originally based on the theory of hypothesis-based reasoning. In hypothesis-based reasoning, hypothesis generation is defined as a symbolic process to infer a hypothesis "H" which is unrecognized knowledge from a observed fact "O" that is experienced knowledge and a known axiom "H | -O" that general knowledge. The symbol " | -" means an provability.

ATMS is a truth maintenance system which determines what nodes are believed in the current situation. A particular situation in ATMS is called context. Also, the fundamental function in ATMS is to maintain consistency and non-contradictory in the whole of knowledge base. In ATMS, a fundamental unit of data is called node and a node consists of three parts -- datum, label, and justifications.

A datum is the contents of data and justifications is the same as that in TMS. A label contains several hypothetical lists called environment. Environment is a set of hypotheses and every environment is recorded in the label of each node. The fact doesn't depends on any particular reasons is called promise. At the time when we retracting on the processing of each inference, we will reach a fact or a hypothesis finally. As a result, ATMS is designed to function in tandem with a problem solver as part of an overall reason system, and the problem solver records all the inferences they make justifications and all the hypotheses make assumptions.

3.3 Temporal Reasoning

In the researches of AI, there are two standpoints toward the concept of time : the law of causality and an event. Mostly, time is regarded as the changes of a situation. The law of causality comes from the definitions of the natural transition among situations. In other words, it is the relationship of the causality from a situation to a situation. This approach aims to understand the time, that means a kind of projection from the past to the future, by predicting a dynamics in the world. On the other hand, an event is a sort of incident that causes the change of situation and it happens at a certain time. Especially, the event by the specific people is called an action. This point of view formalizes the time stream as a sequential relation of 'precedence' or 'earlier-later'.

The formal model of time is mostly established with predicate logic : to formalize the expessions

and reasoning of temporal transition in the world by using predicate logic. But unfortunately, the world of predicate logic is inadequate to describe the change of situation caused by time. To overcome such a defect, people try to expand predicate logic.

In [5], time is a set of states that are instantaneous universes and there is temporal sequence among all the states. The set of state which corresponds to a time stream from the past to the future is called chronicle. Facts are the static element of its temporal representation and events are the dynamic one. This idea approaches to the inference of causality, sequential transition with representing such facts and events.

The other temporal reasoning based on time intervals is supposed in [6].

These temporal inferences realize the concepts of causality, persistency and sequentially, and rich environment with the elements for representation, but it seems to be considered, somehow, the expansion of predicate logic attaches limitations.

4. Analysis System Implementation

We coded the analysis system in the Auotmated Reasoning Tool (ART) and LISP. ART is so far called the second generation tool for expert systems. Its syntaxes are quite similar to those of LISP and it has some feathers for realizing expert systems : schemata, logic dependencies and viewpoints. Therefore, ART is like the preprocessor of LISP language.

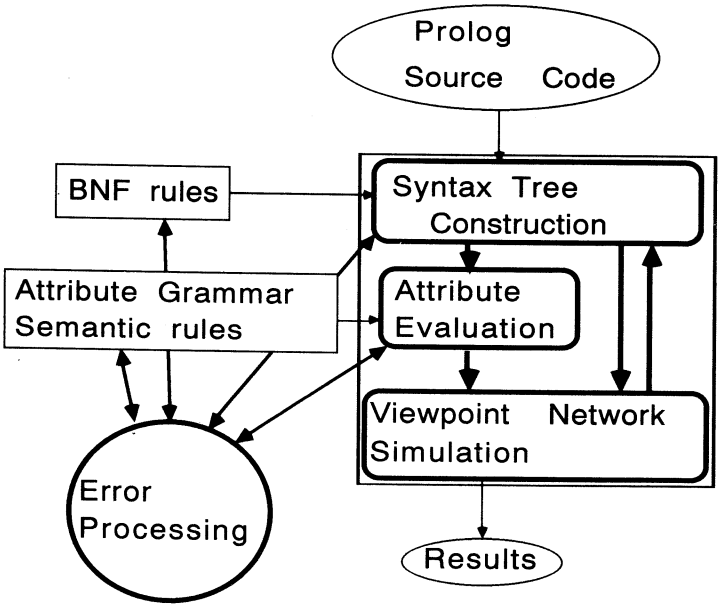

Fig.1 The analysis system

Fig.1 shows the whole structure of our analysis system we implemented.

The input to this sytem is source codes of Prolog. The main part of the system is separated into three parts, the first part is to do symbolic processing from the inputted codes into tree structure ('syntax tree construction'), the second part, 'attribute evaluation' checks the static semantics of the tree, and the final part, that is, 'viewpoint network simulation' is for the processing of dynamic semantic analysis. Then, after passing through inside the system, the result of processing is outputted. The result means the simulated variable at each stage -- temporary, hypothetically and hierarchy. Or, otherwise, some error messages may appear as result in turn, if the Prolog source codes have initially some syntax or semantic problems. The first step, 'syntax tree construction' relies its processes on the knowledge base, Backus Nour Form (BNF) rules. Also, the next process for evaluating the attributes of Prolog has another database called 'attribute grammar semantic rules'. These two previous sections access the part, 'error processing'. The error processing division can be regarded as the other database.

4.1 Syntax Analysis

The original idea in this analyzing is the unique way to convert Prolog into a LISP-like list structure.

4.1.1 Backus Nour Form

BNF is another form of context free grammars (CFG). CFG is the unified theory or methodology for designing programming language or some tools that have their original languages. Of course, CFG exists in a declarative language because the grammar is also the production rules from the start symbol toward all the terminal symbols via

```
<clause>   ::=   <fact> .    |   <rule> .

<fact>    ::=   <relational-expr>
<relational-expr>    ::=    <name>   (   <termlist>   )
<name>   ::=   <small-letter>   {   <small-letter> | _ } *
<termlist>   ::=  <term>  |   <term>  ,    <termlist>
<term>    ::=   <number>    |    <list>    |    <variable>    |
<compoundterm>
<number>   ::=   <digit> { <digit> } *
<list>  ::=    [  ]     |     [   <elementlist>  ]
<elementlist>   ::=   <term>    |     <term> | <term>    |
                                    <term>  ,   <elementlist>
<compoundterm>  ::=  <name>    [   (   <termlist>   )   ]
<variable>   ::=   <capitalletter>  [   <name>   ]

<rule>     ::=    <relational-expr>    :-    <subgoal-list>
<subgoal-list>      ::=     <subgoal>    |     <subgoal>   ,
<subgoal-list>
<subgoal>   ::=   <relational-expr>   |    <comparison>     |     !
<comparison>   ::=   <variable> <compare> <variable>    |
                              <arithmetic> <compare> <arithmetic>
<arithmetic>    ::=    <mult-exp> <adding> <mult-exp>     |
<mult-exp>
<mult-exp>  ::=   <number>   |   <variable>     |
                                   <variable> <multiplying>
<arithmetic>
<compare>  ::=   is   |   \==    |    <    |     >     |    <=   |
  >=
<adding>   ::=   +    |    -
<multiplying>   ::=    *    |     /

<digit>   ::=    0  |  1  |  2  |  .....   8  |  9
<small-letter>   ::=   a  |  b  |  .....   x  |  y  |  z
<capital-letter>   ::=   A  |  B  |  .....   X  |  Y  |  Z
```

Fig.2 The Backus Nour From for Prolog

nonterminal symbols. The start symbol is also categorized in nonterminal. Fig.2 shows the BNF of Prolog.

We should keep in mind not to confuse between several terminal symbols that peculiar to each language and the meta symbols for BNF. Especially, the symbols, a vertical bar and left-and-right brackets. In BNF, a vertical bar expresses the alternation in the body parts of the each sentences while in Prolog, that is a division in the list. The other meta symbols in BNF and their intentions are as follows : " ::= " (production)," ‹ .. › " (nonterminal symbols), " {...} * " (repetition), " [...] " (possible to omit). In the rules, we can easily find out all the symbols except meta symbols and the name surrounded by " ‹ › " are the terminal symbols that appear at each leaves in the tree.

Fig.3 and Fig.4 are the hierarchical structures of Prolog's BNF rules. In a bottom-up parsing, the analysis starts from the terminal symbols and traverses inside the tree from downside to upside along branches, then, checks whether can arrive at the root of nonterminal, "clause" or not. If it succeed, the Prolog source sentences are proved as syntactic okay. On the other, a top-down parsing begins with the root then passes down toward each leaf, terminal symbol. We designed a bottom-up parsing for syntax analysis at this time with the LISP-like lists. Next, we explain the algorithms how to make the LISP-like lists from source codes.

Fig.5 is an example of ordinary syntax tree. In every programming language, its sentence is able to be drawn to such as a tree once the context free grammars of the language has been established.

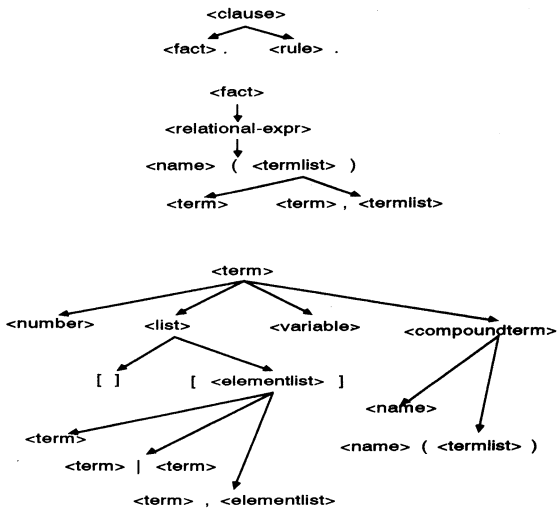

Fig.3 The tree structure for BNF of Prolog's fact

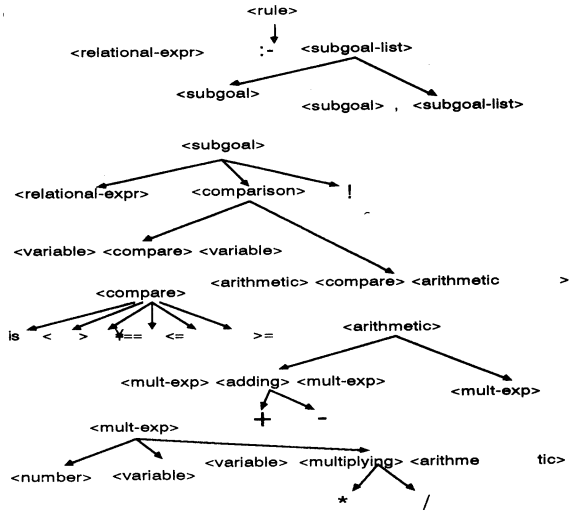

Fig.4 The tree structure for BNF of Prolog's rule

Obviously, all the leaves of the tree keep terminal symbols while the nodes are occupied with non-terminal ones.

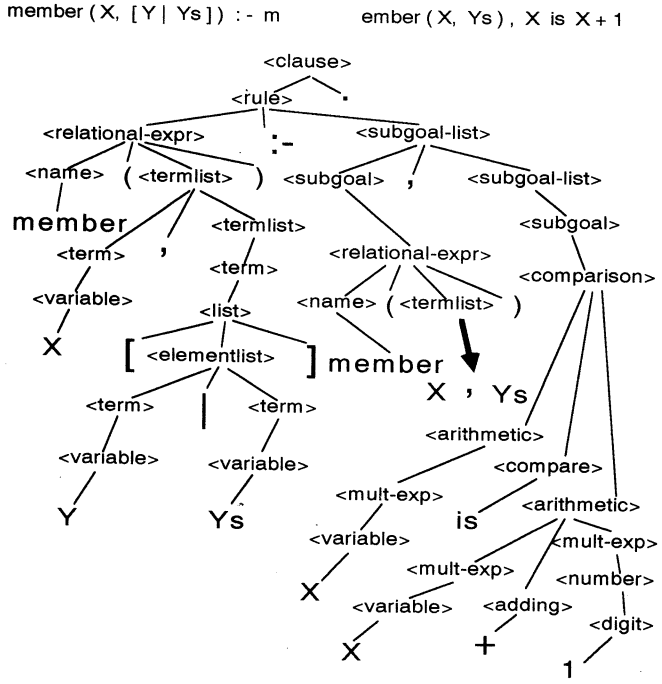

Fig.5 The traditional BNF system tree for Prolog's rule

4.2 Semantic Analysis

Semantic sanalysis can be divided into two components, analyzing static semantics and dynamic semantics. We originally designed the structure of intermediate codes from the BNF tree of source programs. The LISP-like list structure is useful to detect the nature of attributes of the each terminal symbol and helps to convert the codes into the mechanism of dynamic semantic analysis.

4.2.1 Static Semantic Analysis

The main part of analyzing static semantics of Prolog relies the fundamental concepts on attribute grammars. Namely, the system achieves its purpose, checks whether the declaration of terms and their usages have been matching correctly, by attributes.

4.2.1.1 LISP-like Lists

The essence of LISP-like lists is simple : reflecting the depth from the root in BNF tree of each terminal symbol to the depth of the nest in the list. In Fig.6, each number from 1 to 5 intends the depth of terminal symbols. Now we ignore the configuration of nonterminal symbols. As shown in Fig.6, for example, " . " is in the depth "1", the name "member", " (" and ") " are located in the

depth '2', and the symbols " [" and "] " are in the ' 4 '. Then, Fig.7 may be able to construct.

First, we changed all the symbols themselves, except the names which have alphabetical notation initially as "member", into their English notations such as from " | " to " bar ". It is not convenient to keep their symbolic notations still inside a nested list. Also, we must avoid confusing between the parentheses for signs of the beginning and the end of list and "left parenthesis" or "right parenthesis".

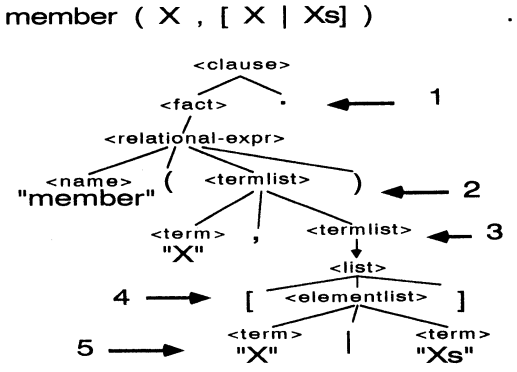

Fig.6 The first approach for the L.L.L. from the BNF tree (The expression L.L.L. is the abbreviation of LISP-like lists)

Then, we arranged the terminal symbols horizontally at each nest from the left to right in order, reflected the position in BNF tree. the new tree drawn at the bottom in Fig.7 shows the nesting situation of each terminal symbol by hierarchical structure. This is not the tree that simply come from the BNF rules. This is the original. All the nonterminal symbols are hidden in the rectangles in the tree. Apparently, every leaf of the tree has been occupied by terminal symbol and this tree shows the depth of nesting of the symbols exactly.

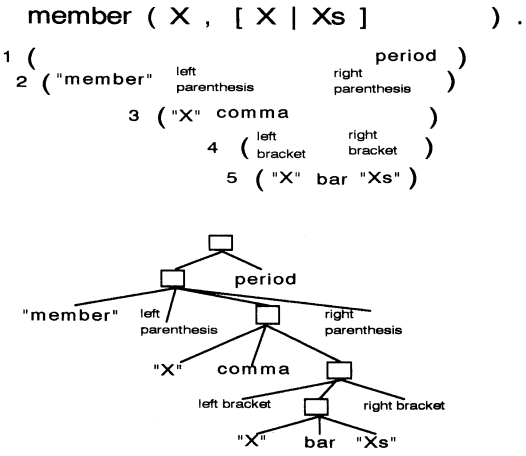

Fig.7 The initial L. L. L. structure and its supplemental tree

Fig.8 shows more elegant structure compared with the previous pictures in both the original list and tree. As it is troublesome and there aren't any serious reasons to keep "left-and right-parenthesis" separately, we combined them like "parentheses" or "brackets". And we have rearranged the position of these terminal symbols so that nonalphabetical symbols appear at the left-most side in each nested list. The reason is simple : it is easier to control the list processing in LISP by the functions such as "car" or "caddr". In addition, the tree below LISP-like lists has changed the location of its leaves correspondingly.

Fig.8 and Fig.9 are almost final stages for designing the LISP-like lists from BNF tree. The Pro-

log sentence in Fig.9 is a rule while the one in Fig.8 a fact. In the latter, the nonalphabetical symbol " :- " appears as the English word " if ". Plus, we finally prepared the list as if it were LISP programs. The element in the list consists of six different nonalphabetical symbols and the English names depends on the programmer : " period ", " if ", " parentheses ", " comma ", " brackets ", " bar " , and so on. The fact described here is all caused by the Prolog's grammar, and is figured

```
member ( X , [ X | Xs ] )
( period
    ( parentheses "member"
        ( comma "X"
            ( brackets
                ( bar "X" "Xs" ) ) ) )
```

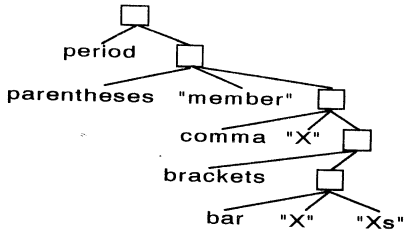

Fig.8 The goal of L. L. L. and its supplemental tree

```
member ( X , [ Y | Ys ] ) :- mem ber ( X , Ys ) .
(period
    ( if ( parentheses "member"
            (comma "X"
                ( brackets
                    ( bar "Y" "Ys" ))))
        (parentheses "member"
            (comma "X" "Ys" ) ) ) )
```

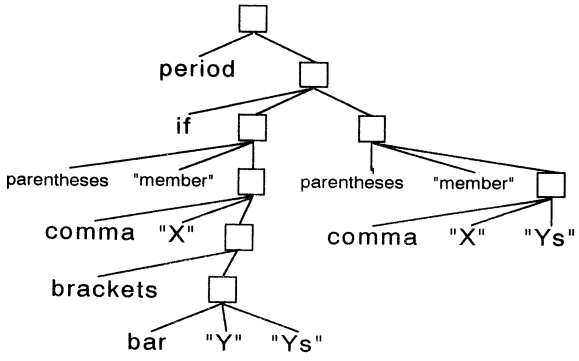

Fig.9 An another example of the L. L. L. using Prolog's rule

out by the symbolic analyzing with LISP-like lists and their supplemental tree structure.

4.2.1.2 Attribute Grammars

The attribute grammar is proposed to describe the semantics of context free grammars (CFG). In Prolog, we found out the several rules of semantic checker that are out of the range of CFG, otherwise BNF rules of Prolog. Remark the sequential ordering of nonalphabetical terminal symbols shown in the supplemental tree, and we might notice the priorities of the appearance from a root of the tree. First, the symbol "period " must be appeared. Second, if the specific sentence of Prolog is a rule, the symbol " if " should be followed. If the sentence is just a fact, " if " never show up. We show the order vertically from up to down with an arrow in Fig.10.

Even though the rule shown in Fig.10, only the symbol " comma " is exception : that is no rules for the " comma " regarding the order of its appearance. For instance, sometimes the " comma " is followed by the " parentheses ", and the other sometimes the vice versa. The reason of such a strange behavior in the " comma " is rather serious and important : it is an explicit fact that the different kinds of " comma " have been existing. The one plays a role of dividing between the variables contained by the specific one predicate, and the other does between predicates themselves that in the body part of the rule sentence in Prolog. Therefore, we may not make the " comma " join to such an attribute rule directly without consideration unless we introduce the special treatment only for the " comma ", like as defining the two different commas, "the meta comma" and "the ordinary comma".

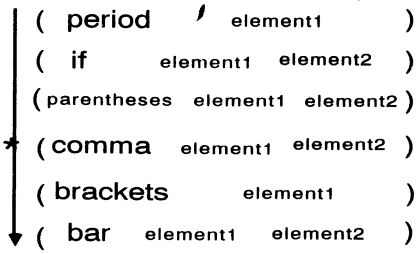

Fig.10 The sequential priority of the appearance of symbols

In addition to that, we checks the attributes from another point of view with the LISP-like lists. At first, each nonalphabetical symbol has the datermined number of arguments. As an example, the "period" attracts one argument while "parentheses" and "comma" must have two. Secondly, the declaration of arguments in the different list is also a target to be examined as follows :

‹ rule 1 ›

(parentheses argument1 argument2

---› The argument1 must be the name of predicate, not be variables.

‹ rule 2 ›

(operator argument1 argument2)

---› Both the argument1 and argument2 must not be the name.

They should be arithmetical operands.

In the rule2, the meaning of "operator" is the mathematical symbols such as "+", "−" and so forth. The most of the time in Prolog, it is obvious that those arguments should be a integer number. Fig.11 shows an example of the attribute checking.

The original Prolog code of the supplemental tree in Fig.11 as follows :

```
fib (N,F) :-N1 is N - 1, N1 >= 0,
    N2 is N - 2, N2 >= 0,
    fib (N1,F1), fib (N2,F2), F is F1 + F2.
```

The bold or plain text rectangles that surround some sets of leaves, or some of the subtrees, shows the necessity of the rule2 ("minus N 1" or " plus F1 F2" etc.) and the unpredicatability of the appearance the "comma".

All the arguments of arithmetical operators such as "minus" or "plus" must be integers or variables. Also, as the rule1, the first argument of the "parentheses" must be the name of the specific predicate something like "fib".

4.2.2 Dynamic Semantic Analysis

The purpose of a trial to analyze the Prolog language's semantics dynamically is to examine the nondeterministic characteristics behind its syntax as a background. As technological problems, we had to rely our original idea of analysis on a practical tool the ART. The fundamental objects in implementing with ART is to construct some kinds of expert systems. More specifically, the ART has a peculiar feather called viewpoints that is able to realize the temporal and hypothetical reasoning simultaneously.

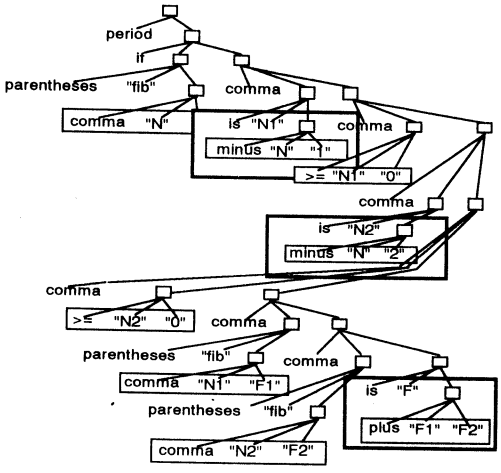

Fig.11 The supplemental tree for an attribute grammer

Fortunately, some theoretical approaches toward knowledge representation have implied the positive flexibility in the tool in the field of mono-directive inferences such as time stream and in that of assumption-based inferences such as hypothetical-based reasoning.

The function of viewpoints in the ART is to help updating a set of the structured facts. These flexible facts is also stored in the fact database in the system. Additionally, it is possible to modify such facts dynamically and drastically with transiting inferences. Concretely, viewpoints multi-

plies the fact database constructs a fact network where facts can easily be connected and taken apart. Originally, these are two types of viewpoints called time-based and hypothetical-based, but we have put some essences to accomplish the multi-universe reasoning, too.

4.2.2.1 Temporal and Multi-universe Viewpoint

At first, we describe the usage of viewpoint without the hypothetical one, only the concept of joining both temporal and multi-universal transition. Fig.12 explains visually the changing the situation under the two different criteria in an example of the Prolog rule sentence. The above picture in Fig.12 shows that the head or the left-hand side of the rule is in the standard "Time-1 & Universe-1", then the body or the right-hand side is in the "Time-2 & Universe-1" at the top-level or at the most outside's standpoint. We can notice that the body part has been nested by the condition "Time-1 & Universe-2".

This definition is not so hard to understand : the transition from the left-hand to the right-hand might be recognized as time passing, and in the body part, it might be necessary to restandardize at the universal point of view in order to reset a time. That's why the inside situation in the left-hand is in "Time-1 that means the time resetting & Universe-2".

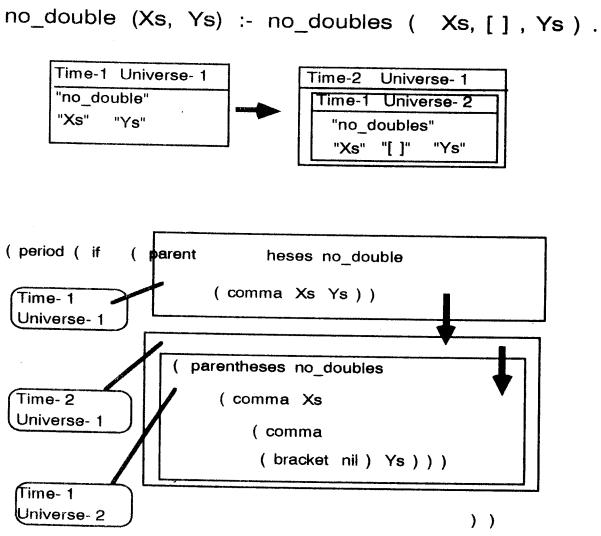

Fig.12 How to apply the temporal and multi-universe viewpoints

On the other hand, the below chart in Fig.11 shows the same concept by the LISP-like lists of the Prolog rule. In this case, the supplemental tree for the lists is omitted. The upper nested list starts by "parentheses no_double" is analogy of the head of the rule while the lower sublist corresponds the body so that it is in the both "Time-2 & Universe-1" and "Time-1 & Universe-2".

Fig.13 describes the different situation. This is also the rule sentence in Prolog, but the body contains two independent predicates. Those predicates are joined by a comma so that we introduce the temporal transition from before to after comma. This chart is quite understandable that the first predicate in the left-hand side, "member" in the situation defined "Time-1 & Universe-2", that show the time criterion at this point in the nested level. Then, the second predicate exists in the world "Time-2 & Universe-2" where the one time interval from the first have been passed.

```
no_doubles ( [ X | Xs ] , Ans ,        Ys ) :- member ( X , Ans )        ,
                                              n          o_doubles ( X , Ans , Ys ) .
```

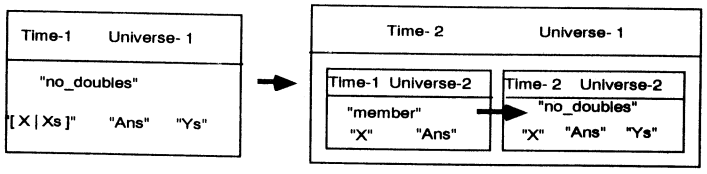

Fig.13 An second example of the two viewpoints

4.2.2.2 Hypothesis-based Viewpoint

The hypothesis-based viewpoint is called only when the heads of plural sentences of the rule happen to coincide. In Fig.14, the heads of both rule sentences are the same : the name of predicate, the number of arguments those predicates contain, the contents of the three argument. This is a real don't-know nondeterministic situation. At this point, we have no keys or priorities which body should be selected at first. Therefore, we labeled the both alternatives from the upper "Hypothesis-1" and "Hypothesis-2" respectively. Other determination for setting conditions at each predicate is all followed by the rule described at the previous section 4.2.2.1.

```
no_doubles ( [ X | Xs ] , Ans , Ys        ) :- member ( X , Ans ) ,
                                               no_dou        bles ( X , Ans , Ys ) .

no_doubles ( [ X | Xs ] , Ans , Ys        ) :- nonmember ( X, Ans ) ,
                                               no_dou        bles ( X , [ X | Ans ] , Ys ) .
```

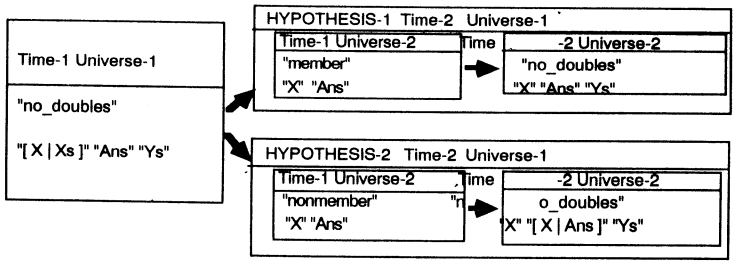

Fig.14 A fundamental concept of the hypothetical viewpoint

As a result of these definition, the temporal and multi-universal position are determined all the predicates, the head and the body, the fact and the rule, but whether the hypothetical reasoning is

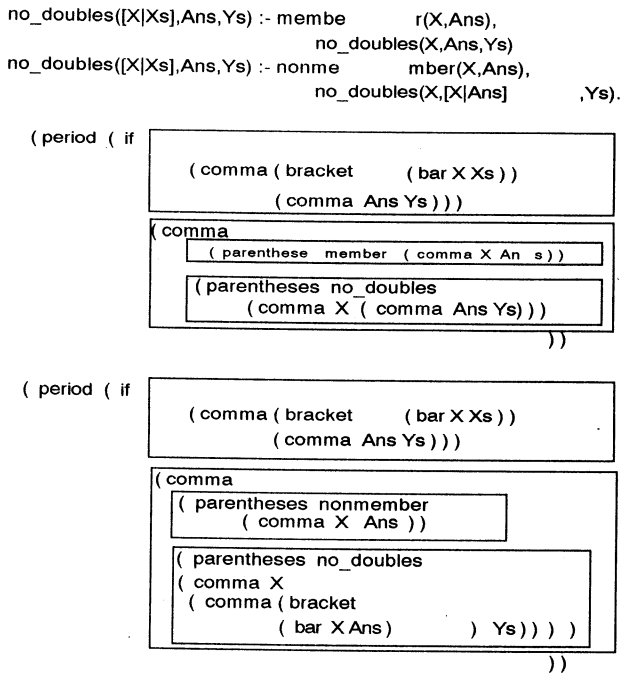

Fig.15 An example clauses for applying three viewpoints

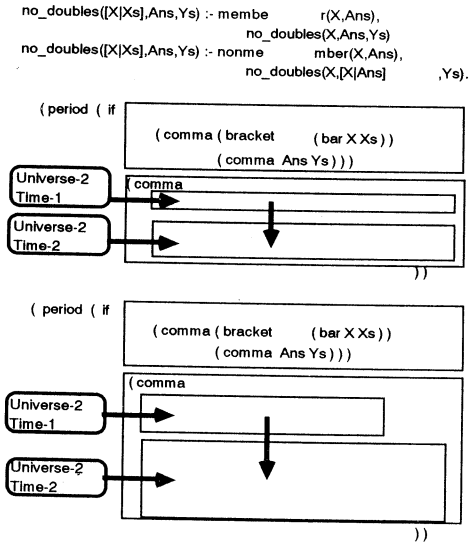

Fig.16 How to apply the two viewpoints to the L.L.L.

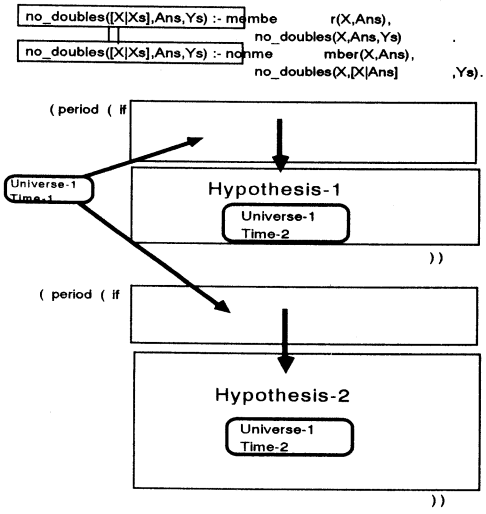

Fig.17 How to apply three viewpoints to the L.L.L.

required or not is quite depends on the Prolog source code. It is because some Prolog programs do not need a don't-know nondeterministic processing, just enough to process procedurally.

This time, we describe the relationship among these different viewpoints from standpoints of the LISP-like list. Fig.15 is an instance of the two Prolog rules and their conversion into the lists. All the rectangles appeared on the lists. And we show another two independent lists in Fig.16 and Fig.17. Fig.16 presents the relationship between the temporal and multi-universal viewpoints. In each sentence of Prolog, the first argumenu of the "comma" list is regarded in the situation "Universe-2 & Time-1", and the second one is in "Universe-2 & Time-2". Because of the "comma" list itself has been regarded as the second argument of the list "if". The depth of the "comma" is incremented from the top-level, that is why the level of "universe" is 2. Also, the time is counted in each transition of the situation, so the level of "time" is incremented by passing from the first argument of the list "comma" to the next.

Fig.17 describes the relationship when the hypothesis-based viewpoint has been also considered. Both of the first argument of the list "if" in the two Prolog sentence are coincided exactly (Fig.15), so that the second arguments of the "if" list are determined as "Hypothesis-1" and "Hypothesis-2" from up to down, respectively. In addition, the first arguments of the list "if" are both in "Universe-1 (that means they are in the top-level) & Time-1" and the second argument of the "if" list are considered as "Universe-1 & Time-2" in the each sentence. These definitions stand along at each sentence. Remember, the hypothesis viewpoints are not always defined in Prolog programs.

Finally, we reexamine the methodology of time, hypothesis, and universe viewpoints by using the tree structure (Fig.18 and Fig.19). These are not the BNF syntax tree but the supplemental tree for the LISP-like lists. Fig.18 shows how to apply both concepts of time and universe to the tree. The flow of time can be drawn as the horizontal arrow while the nesting of universe as the vertical arrow. For example, the node "parentheses nodoubles" where the first argument of "if" is in "T-1" (Time-1), and the node "comma" that is located the right side of the "parentheses" node, we can also say this is the second argument of the list "if" by observing the tree horizontally at the third level from the root, is in the position "T-2" (Time-2). These time flowing are both in the depth "Universe-1". Also, we can find out another time flowing on the "Universe-2" where all nodes are the children of "Universe-1". Be careful that there are not any realtions among the time points in the different universes. Once we traverse the tree toward its leaves, the time arrangement in the previous depth might be all void. We must initialize the time interval again at that updated depth. Fig.18 is presenting another aspect as regards these viewpoints as a two-dimensional

space.

On the other hand, Fig.19 is implemented the concept hypothetical-based viewpoint. This tree intends the following discovery : if the structure of the subtree that is recognized as the first argument of the "if" list is exactly the same as that in the other tree, the whole structure of the next subtree as the second argument of the "if" list should be regarded as the one of hypotheses. The

```
no_doubles( [ X | Xs ] , Ans, Ys)        :- member ( X, Ans ) ,
                                          no_double        s (X , Ans , Ys ).
```

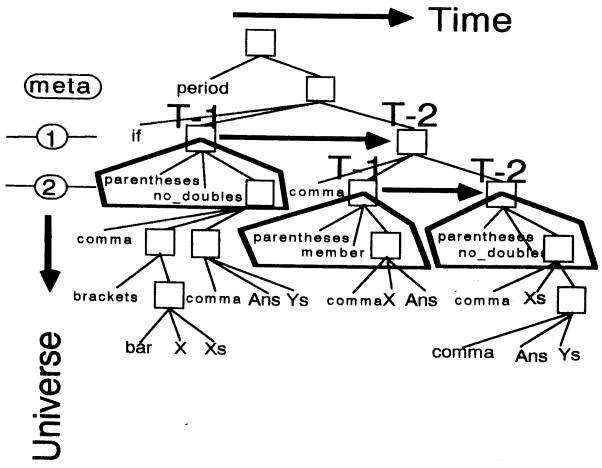

Fig.18 Two-dimentional relation between time and universe

```
no_doubles( [ X | Xs ] , Ans, Ys)        :- member ( X, Ans ) ,
                                          no_double        s (X , Ans , Ys ).
```

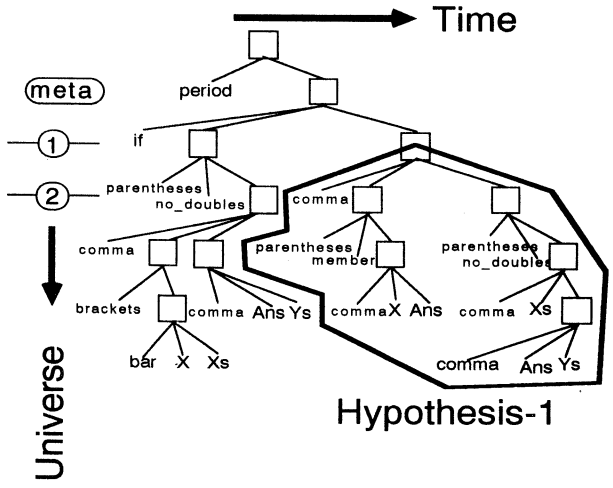

Fig.19 Three-dimentional relation among time, universe and hypothisis

subtree surrounded by a bold polygon is in the situation "Hypothesis-1". And the "Hypothesis-2" condition must exist in the part of (or the subtree of) some different trees. Each Prolog sentence either a fact or a rule has such a LISP-like list's tree, and if we try to pile the plural trees one another, we might discover the fact such that some subtrees located on the first argument of the "if" list have coincided perfectly and these on the second argument are consist of the different structures. The latter must be defined as hypotheses. Therefore, as the final conclusion of the estimation concerned with multi viewpoints, we have found out that the tree independent viewpoints these are time, universe and hypothesis are developed to form a three-dimentional world of the tree. That is, time viewpoint is spread out horizontally, the universe spread out vertically, and the hypothesis viewpoint grown toward the above by piling on the hypotheses.

5. Conclusion

As the final remark of this paper, we conclude the research as follows :

We proposed an algorithm, such as how to construct the LISP-like list from a target language Prolog, that asserts the dynamics of semantic analysis as the central part of compliers for designing a declarative language processor.

When we design a new programming language, once we determine the precious grammar, its peculiar context free grammar, we can adopt the concepts of our algorithm. Of course, the reserved words of each independent language are peculiar. But imaging the analogies from the idea in this paper is not so hard. We believe it might be one of the most optimal method that to analyze the language, especially, a nondeterministic language, symbolically such as using dynamic and simple data structures in LISP.

We did rather stress on the relationship between the field of "semantic analyzer" and that of "knowledge representation".

The validities of applying reasoning deal with the concept of time,hypothesis and multi-universe to the processing and the result of the symbolic analysis of a nondeterministic calculus model via Prolog are gotten in this research. But, these are omitted on account of limited space unavoidably. Moreover, it is much to be regretted that there are still unsolved matters for the some phrase in a compiler such as error consulting.

References

[1] McCarthy, J., et al. : Some Philosophical Problems from the Standpoint of Artificial Intelligence, Machine Intelligence, Vol.4, pp.463-502, 1969

[2] Sandewell, E. : An Approach to the Frame Problem and its Implementation, Machine Intelligence, Vol.7, pp.195-204, 1972.

[3] Reiter, R. : A Logic for Default Reasoning, Artificial Intelligence, Vol.13, pp.81-132, 1980.

[4] McDermott, D. V., Doyle, J., : Nonmonotonic Logic I, Artificial Intelligence, Vol.13, pp.41-72, 1980.

[5] McDermott, D. V. : A Temporal Logic for Reasoning About Processes and Plans, Cognitive Science, Vol.6, pp.101-155, 1982.

[6] Allen, J. F. : Towards a General Theory of Action and Time, Artificial Intelligence, Vol.23.

[7] ART Refernce Manual, Inference Corp.
ART Programming Tutorial, Inference Corp.