PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1984,
edited by ICOT.

LOOKS: Knowledge Representation System for Designing Expert Systems in a Logic Programming Framework

Fumio Mizoguchi, Hayato Ohwada and Yoshinori Katayama

Biosystem Research Laboratory
Dept. Industrial Administration
Science University of Tokyo
Noda, Chiba, Japan

ABSTRACT

This paper describes an object-oriented approach to designing a knowledge representation system for developing expert systems. The system is called Logic Oriented Organized Knowledge System (LOOKS for short), and is based on the belief that Prolog's expressive power consists in versatile interpretation and compatibility, so as to integrate various programming paradigms, such as object-oriented and rule-oriented systems. This integration is accomplished by the use of a concept proposed by Bowen & Kowalski which amalgamates object-level and meta-level knowledge. The meta-predicate *"world_demo"* is developed for this purpose and demonstrated in designing an expert system. The proposed system is implemented in DEC-10 Prolog and an application system called LOOKS/Glaucoma has been implemented to diagnose glaucoma. Finally, there are some discussions of the effectiveness of the present approach to knowledge representation.

1 INTRODUCTION

Recent attempts to design knowledge representation systems have adopted the concept of integrating diverse kinds of programming paradigms as tools for developing expert systems. Knowledge representation systems such as LOOPS (Bobrow & Stefik, 1981) and MRS (Genesereth M. 1983) have been developed with a trend in this direction. In LOOPS, there are four programming paradigms : data-oriented, procedure-oriented, object-oriented and rule-oriented paradigms. This system was implemented in a LISP environment and has served as a tool for developing an expert system in VLSI CAD (Brown et al., 1983).

The present paper takes a different approach to developing a knowledge representation system by using the logic programming language,Prolog. This is based on the belief that Prolog's expressive power consists in versatile interpretation and compatibility with various programming paradigms, even if Prolog is regarded as a semantically flat language among the *"LISP community"*.

As for the expressive power of Prolog, we have tested its efficiency by comparing the performance of a rule-based expert system implemented in Prolog with that of other programming languages (Mizoguchi,F.,1983). The result of this comparison shows that the code size of the Prolog implementation is almost ten times smaller than the code size of implementations in other programming languages. The overall performance of the Prolog-based system is the same level as that of expert systems implemented in other languages. Since Prolog is a self-embedding inference language, the results of comparison are quite natural. As for describing structured objects, Prolog is said to be weak and inappropriate. Our goal is to develop a semantically rich knowledge representation system using Prolog. We have developed a representation system called MIRROR, which was designed as a preparatory step to test the possibilities of reflecting several programming paradigms into the logic programming paradigm (Mizoguchi et al., 1984).

Basing on these experiences, we will develop LOOKS to describe knowledge used in expert systems. In this paper, section 2 explains the object-oriented features of LOOKS and gives examples of objects used in the diagnostic system. In section 3, we propose a meta-control predicate called *"world_demo"*, which serves to as integrate object-oriented and rule-oriented paradigms into a single expert system. Section 4 discusses the results of developing an expert system using LOOKS and the issues related to its development. Finally, we discuss on issues of knowledge representation for developing expert systems for realistic applications.

2 THE OBJECT-ORIENTED FEATURES OF LOOKS

LOOKS makes use of dual representation. Here, dual representation means that any representation system must have a dual character such as declarative vs procedural, structural vs functional, static vs dynamic. In LOOKS, there are two kinds of representation paradigms to be taken into consideration to achieve this duality. One is an object-oriented paradigm for representing structural objects using a class hierarchy, as in Smalltalk-80 (Goldberg & Robson, 1983) (Zaniolo, 1984). The other is a rule-oriented paradigm for describing procedural knowledge in terms of production rules (van Melle, 1979). The two paradigms are integrated by a meta-predicate, which is explained in the next section. Here, the notion of meta-predicate means knowledge about how to use the two types of knowledge so as to meet specific goals.

The objects in LOOKS are composed of meta-class, class and instance. The class object represents structured knowledge with its associated set of methods. The class decides the characteristics common to a group of instance

objects. The instance is an entity which belonging to a specific class. The metaclass object determines the behavior of a class object. In LOOKS, a class object has two kinds of variables: class variables and instance variables. Instance variables describe common property of all instances belong to a class. They also consist of information specific to an instance. A method determines the behavior of a class or an instance associated with it. Thus, an object is processed by receiving a message, which changes its internal state. An object in LOOKS represents a unit world, which is used as a prototype for a specific instance. The method is used for processing a message sent to an object. The class definition and method are shown in Figure 1.

```
defclass <Object Name> ::
   supers <Namelist> ;
   metas <Namelist> ;
   class_v <Name>-<Definition>, ...  ;
   instance_v <Name>-<Definition>, ...  ;
   method
     Selector :=: Function1,
                  Function2, ...  ;
     Selector :Arguments:
              Function1(Arguments, ...),
              Function2, ...  ;
                         ...  .
```

Figure 1. Class definition in LOOKS

The class definition in Figure 1 is classified into three hierarchies: meta-class/class, superclass/class and class/instance. This definition is similar to that of LOOPS. The superclass/class supports the is-a hierarchy, which describes a conceptual hierarchy for the domain knowledge. Property inheritance is carried out automatically between the superclass and class hierarchies. The meta-class/class is used for managing behaviors of class object.

An instance is created by receiving a message to a class object that represents the instance state. The object and method are compiled into an internal form in terms of a Prolog program through the LOOKS loader. The internal form of the object and method is shown in Figure 2. The superset/subset and meta-class/class relations are transformed into head parts of a Prolog program. *Class_v* and *instance_v* are also head parts. The method of

```
<Object>(Object,superclass,Namelist).
<Object>(Object,metaclass,Namelist).
<Object>(Object,c_var,[Name,Definition]).
<Object>(Object,i_var,[Name,Definition]).
<Object>(Self,methods,[Selector]) :-
        Function1, Function2, ...
<Object>(Self,methods,[Selector,Arguments]) :-
         Function1(Arguments, ...),
         Function2, ...  .
```

Figure 2. Internal Form of Class Definition

```
send([Object,Selector]) :-
  send([Object,Selector,[]]).
send([Object,Selector,Argument]) :-
  next_object(Object,Name1),
  gb_count(Object),
  searchmethod(Name1,Object,Selector,Argument).

searchmethod(Object,Self,Selector,[]) :-
  object_head(Object,Self,methods,[Selector]), !.
searchmethod(Object,Self,Selector,Argument) :-
  not(Argument==[]),
  object_head(Object,Self,methods,
              [Selector,Argument]), !.
searchmethod(Object,Self,Selector,Argument) :-
  super_object(Object,Super),
  searchmethod(Super,Self,Selector,Argument).
```

Figure 3. Send and Search-method predicates

definition is translated into a Prolog program. The selector and arguments in the method correspond to Prolog's head part. The function for the method are body parts in Prolog that describe procedure for processing a message. There are two types of methods for processing messages to the class object. In case of the message operator *":=:"*, a function is processed without arguments. The operator *":Argument:"* processes the functions with arguments. In principle, a message to a specific class object is processed within the local class world. Message processing is carried out through the predicate *"send"*, which is shown in Figure 3.

The predicate has three arguments; the first is the object name and the second is the selector part of the method characterizing the behavior of an instance. The third is used for message processing variables. The method search processing is carried out by the *"search_method"* predicate. This predicate searches the selector in the class. The predicate has four arguments ; the first argument is the object name for searching the selector, the second is the object that receives a message, the third is selector, and the fourth is variable. The matching process of selector is carried out by predicate the *"object_head"*. The *"super_object"* used in the predicate begins with the searching the superclass through depth-first and left-to-right manner. If there is no method processing the message to the present class, then the object tries to search for the other superclass.

In designing an expert system, we must develop a model of diagnostic formalism. LOOKS only provides a framework for representing knowledge used in the model. That is, the knowledge used in the expert system (in this case, diagnostic system) must be represented using the LOOKS framework. The overall structure of the diagnostic system is shown in Figure 4.

The model is derived from our experiences using the EXPERT formalism (Weiss & Kulikowski, 1979). The model itself is a revision of the EXPERT formalism, which is extended by introducing the notion of class objects; one is a factual object corresponding to actual data or symptoms, and the other is a hypothetical object repre-

Figure 4. Objects used in the Diagnostic System

sented by a taxonomic category of the domain knowledge, such as a disease or a fault state. In order to design a more concrete image of the diagnostic system, we show one of the objects that corresponds to the diagnostic formalism. For example, class object of *"finding_set"* is defined in terms of LOOKS objects, as is shown in Figure 5. This object is used so as to collecting patient finding data, which is obtained from a user. The other class definition, shown in Figure 6, is the *"yes_no"* object, which is a subset of the *"finding_set"* class. This class generates *"yes_no"* questions to gather patient data. Thus, the EXPERT formalism is very natural for defining class objects hierarchically. As for the instance object, a symptom is defined as shown in Figure 7. The present diagnostic system for

```
defclass finding_set ::
   supers expert_unit ;
   metas meta_expert ;
   instance_v
     prompt , superset - [] ;
   method
     get_val :[Namelst,Ans]:
      ( not(pati_world(self,Val)),
       (self <- get_val0,[Namelst,Ans]);
                 pati_world(self,Ans) ) ;
     active_qa : X :
       @(prompt,N),
       writelis([wctop,N],findings1),
       print_qa1(self), classname(self,Class),
       getcvalue(Class,type,X1),
       writelis([X1],findings1),
       (self <- read_qa,X) ;
     read_qa : Ans :
       repeat, in_tty(' * ',self,X),
       (self <- typecheck,[X,Ans]) .
```

Figure 5. Finding_set Class

```
defclass yes_no ::
   supers finding_set ;
   metas meta_expert ;
   class_v type-'Yes-No:' ;
   method
     active_qa :Ans:
       @(prompt,N),
       writelis([wctop,N],findings1),
       classname(self,Class),
       getcvalue(Class,type,Ty),
       writelis([6,Ty],findings1),
       (self <- read_qa,Ans) ;
     typecheck :[Ans,[Ans]]:
       type_yesno(Ans),
       pati_world(self,[Ans]).
```

Figure 6. Yes_no Class

```
definstance symptom ::
  class_n multiple ;
  instance_v prompt - 'Symptoms:' ,
        superset - [topics(symptom)] ,
        candidate -
        [(rainbow, 'rainbow vision/halos'),
         (complaint, 'complaints of decreased
                                visual acuity'),
         (pain, 'ocular/head pain'),
         (discomfort,'discomfort'),
         (eye_strain,'eye strain'),
         (vomiting, 'vomiting'),
         (headache, 'headaches in dim light'),
         (transient, 'transient blurred vision'),
         (redness_tearing, 'redness/tearing'),
         (photophobia, 'photophobia'),
         (loss, 'complaints of loss of field') ].
```

Figure 7. Symptom Instance

glaucoma contains about 187 objects (15 class objects, 172 instance objects).

3 USE OF META-LEVEL KNOWLEDGE

3.1 Concept of meta-level control

In the previous section, we described the objects used in LOOKS and the diagnostic model. We also showed some examples of object definitions for the model using the LOOKS formalism. In this section, we explain meta-predicates dealing with the integration of different worlds. This is accomplished by amalgamating object-level knowledge and meta-level knowledge, as proposed by Bowen & Kowalski (1981). The concept was implemented by Kitakami et al (1984). This concept uses a procedure called *"demo"* predicate, which checks the provability of a Prolog program. We adopted this predicate for integrating the object-level and meta-level. The object-level encodes knowledge about the facts of the domain (in our case, dual objects and patient state), which the meta-level encodes as control strategies (in this case, methods of integrating

different paradigms). This amalgamation has the following two advantages:

- The separation of object worlds and control information preserves the clarity of program structure by making it modular (Davis, 1980), (Bundy, 1981).
- Meta-level integration unifies different programming paradigms. By meta-level control, an abstract control mechanism is realized for the common control of the various programming paradigms.

Thus, by introducing meta-level knowledge, we achieve a general control structure, which combines different paradigms into a unified world model. This is carried out by the meta-predicate called *"world_demo"*, which processes a set of object-level knowledge corresponding to domain knowledge. It takes the form:

world_demo(World,Goal,Dependency,Justification)

where *'World'* denotes a world representing a set of object-level knowledge, and *'Goal'* is Prolog clauses to being solved through a set of *'World'* knowledge. The third argument *'Dependency'*(Shintani & Mizoguchi, 1983) returns proof tree in solving *'Goal'*. It plays a role of explanation facilities such as HOW in EMYCIN. The fourth argument *'Justification'* returns the degree of belief which denotes justification of *'Goal'*. Here, justification is indicated by an integer between -100 and 100, which corresponds to an EMYCIN-like certainty factor. The declarative reading of *"world_demo"* interprets that a certain goal is derived from given world (a set of object-level knowledge) and returns the dependency relation and justification. The control flow of *"world_demo"* is shown in Figure 8.

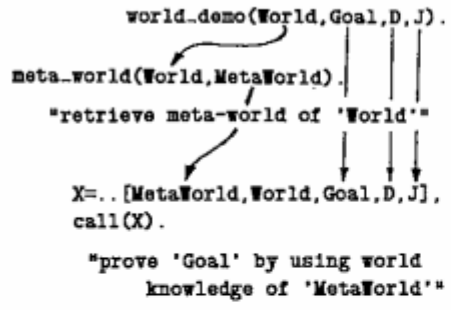

Figure 8. Control Flow of "world_demo" predicate

Each object world is associated with meta-world which controls a set of knowledge of object world. Meta-world is specified as follows:

meta_world(<object_world>,<meta_world>)

A form of <meta_world> knowledge is described as below.

<meta_world>(World,Goal,Dependency,Justification)
:- Body

Here, four arguments: *World*, *Goal*, *Dependency* and *Justification* are same as used in *"world_demo"* predicate.

3.2 Realization of meta-level control

In order to apply *"world_demo"* in a diagnostic system, we prepared three world models: object-world (*O-world*), patient-world (*P-world*) and decision-world (*D-world*). *D-world* contains heuristic knowledge represented in rule-oriented language. This language is described in terms of *"IF,THEN"* type production sets. This is expressed by Prolog programs with a certainty factor. In this case, the rule itself is inherited from the EXPERT rule formalism. *O-world* corresponds to the diagnostic formalism shown in the previous section. This world is described by LOOKS objects and controlled by message passing. The *O-world* meta-predicate shifts control of *"world_demo"* to LOOKS object-oriented message passing. It converts *'Goal'*, which is the second argument of *"world_demo"* to message sending form in LOOKS. The relation between *"world_demo"* and object oriented programming in LOOKS is shown in Table 1.

Table 1. Relation to object oriented programming

"world_demo"	object oriented programming
object(variable(Value))	getvalue(object,variable,Value)
object(variable(value))	putvalue(object,variable,value)
object(selector,arguments)	send(object,selector,arguments)

```
/* Meta-predicate dealing with
   knowledge of patient world */
meta_patient_world(World,P,D,J) :-
        nonvar(P),P=assert(Q),!,
        X=..[World,Q,D,J],
        assert(X).
meta_patient_world(World,P,[],100) :-
        nonvar(P),system(P),!,P.
meta_patient_world(World,(P;Q),D,J) :-
        nonvar(P),nonvar(Q),!,
        meta_patient_world(World,P,D1,J1),
        meta_patient_world(World,Q,D2,J2),
        ( max(J1,J2,J),
          D=D1,J=J1 ;
          D=D2,J=J2 ).
meta_patient_world(World,(P,Q),D,J) :-
        nonvar(P),nonvar(Q),!,
        meta_patient_world(World,P,D1,J1),
        meta_patient_world(World,Q,D2,J2),
        append(D1,D2,D),
        times(J1,J2,J).
meta_patient_world(World,P,D,J) :-
        metacall(World,P,D,J),
        J==100.
meta_patient_world(World,P,D,J) :-
        not(J==100),
        bagof((D1,J1),metacall(World,P,D1,J1),S),
        collect(S,D,J).
```

Figure 9. The meta-predicate for patient world

P-world deals with the patient knowledge base, which is described in terms of physiological states. The world is defined by asserting clauses. The meta-predicate for *P-world* is shown in Figure 9.

3.3 Control Structure in Diagnostic System

In developing expert system, we must implement a specific inference mechanism for the domain. The *"world_demo"* deals with the *D-world*, which is described using production rules. The following inference is shown in Figure 10.

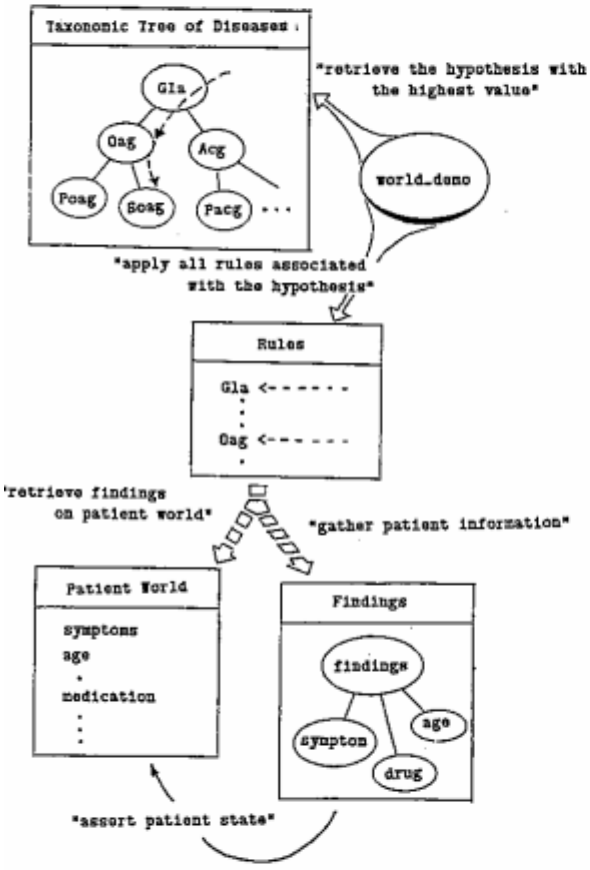

Figure 10. Inference Mechanism in Diagnostic System

The basic features of inference system are as follows:

- Backward chaining
- Best-first search using weighted disease hypothesis

The inference algorithm is summarized by the following steps.

(1) Retrieve uppermost disease hypothesis.

(2) To verify the hypothesis, apply all rules associated with it.

(3) If there are some lower hypotheses, generate them and calculate their weight. Otherwise, stop.

(4) Retrieve the hypothesis with the highest value and go to (2), recursively.

The weight is determined by frequency of occurrence based on past statistical data. Initially, this weight is based on expert knowledge; it is then revised by calculating the certainty factor.

3.4 Implementing status

LOOKS is currently implemented in DEC-10 Prolog (Peirera et al., 1978) at the Biosystem Research Laboratory, Science Univ. of Tokyo. LOOKS was developed together with MIRROR, an experimental knowledge representation system using a logic programming framework. Some LOOKS modules are inherited from MIRROR components. By utilizing these modules, we improved the speed of the LOOKS implementation. LOOKS consists of the following lines of Prolog code: loader(150), kernel(250), message handler(50), utility(150) and window package(100).

As for the expert system domain, we have chosen a diagnostic system for glaucoma. Knowledge about glaucoma is obtained from our past experience in developing diagnostic systems, and from our collaboration with Dr. Kitazawa of the Department of Optholmogy, Tokyo University Medical Center. We have developed at least three versions of a glaucoma diagnostic system. The first used the EXPERT formalism developed at Rutgers University. The second was developed using our tool called MIRROR/Glaucoma, which was implemented in C-Prolog on a VAX 11/780 at Tokyo University. The third is the current implementation of LOOKS/Glaucoma using DEC-10 Prolog at the Biosystem Research Laboratory. The LOOKS/Glaucoma displays the results of the diagnosis through a multi-window system. In order to explore useful expert system tools, LOOKS was ported to a small-scale micro computer (NEC PC-9801F). This system is called mini-LOOKS. Prolog KABA, which is compatible with DEC-10 Prolog, is used for this implementation. Mini-LOOKS will probably serve as the pre-processing system for LOOKS/Glaucoma, which will perform more detailed diagnoses. In mini-LOOKS, we consider the graphic and color display facilities for representing visual field and other instrumentation data.

3.5 Output Example

This output shows the diagnostic process of the LOOKS/ Glaucoma system. This output is from the DEC-20 system at ICOT Research Center. The first input in this example is part of patient topics for glaucoma prompting the checklist type question. The response to the question by a user is followed by the asterisk(*). In this case, question items 2 and 5 are selected. In LOOKS, this selection is regarded as the message to the symptom object. The system responses to the items generating the state of the object. In order to obtain the explanation of diagnostic process, *"world_demo"* is used by sending the message to patient world. The output shows the

```
Symptoms:
   1 .  rainbow vision/halos
   2 .  complaints of decreased visual acuity
   3 .  ocular/head pain
   4 .  discomfort
   5 .  eye strain
   6 .  vomiting
   7 .  headaches in dim light
   8 .  transient blurred vision
   9 .  redness/tearing
  10 .  photophobia
  11 .  complaints of loss of field
Checklist:
    * [2,5].
Angles:
   1 .  Grade 0 - complete or partial closure
   2 .  Grade 1 - narrow extreme
   3 .  Grade 2 - narrow moderate
   4 .  Grade 3 - open extreme
   5 .  Grade 4 - open moderate
   6 .  Slit
Choose one:
    * 4.
Enter the intraocular pressure (applanation
      Numerical:
    * 22.
     ...  (continuing Q & A)

yes
| ?- conclusion.

   --- Diagnosis ---
poagm 40 Primary open angle glaucoma mild risk
pacg 28 Primary angle closure glaucoma
congl -30 Congenital glaucoma
     ...  (partial example of conclusion)

yes
| ?- world_demo(patient_world,pacg,D,J).

D = [[((pacg:-age(56),40<56),20)],
     [((pacg:-sex(male)),10)],
     [((pacg:-symptom(discomfort);
              symptom(eye_strain)),50)],
     [((pacg:-angle(gra3),intpr(22),20<22),-100)
J = 28
      ...  (reasoning process)
```

Figure 11. The Output in LOOKS/Glaucoma

verification whether the current input data is provable or not within the patient database.

3.6 Related Topics

Knowledge representation is one of the key issues for research in artificial intelligence. There have been a number of attempts to develop knowledge representation languages beginning with KRL (Bobrow & Winograd, 1977), and culminating in the recent development of LOOPS. These languages and systems were built on the idea of *"Frames"* (Minsky, 1975) and were implemented in LISP (Mizoguchi, 1982). These systems represents an important aspect of the *"LISP culture"*.

Recently, several knowledge representation systems have been implemented using a logic-based approach. Systems like MRS and KRYPTON (Brachman et al., 1983) use the framework of predicate logic. PROLOG/KR (Nakashima, 1984) is similar to LOOKS, but adopts a frame-oriented approach using the *"with"* predicate for multiple world modeling. We accomplish this mechanism using *"world_demo"*, which integrates the different worlds. In our present research, the *"knowledge programming paradigm"* is a central concern. It is adopted in Mandala (Furukawa et al., 1984), which uses Concurrent Prolog, a language with great expressive power for parallel computation. As a result, Mandala's proposed model of *"simulate"* is one of the most elegant for dealing with inheritance using state variables. Our approach is a more realistic one, we feel, for using meta-level control. Such control distributes the processing to different worlds, and goals are then solved within each world.

The main differences between LOOKS and Mandala are in the implementation language and the applied aspects of knowledge representation. LOOKS takes advantage of Prolog flexibility for implementing object-oriented knowledge representation. Further, LOOKS has been tested using large objects for the diagnostic system. But our efforts should be toward the parallel inference strategy for using Concurrent Prolog (Shapiro & Takeuchi, 1983).

4 CONCLUSION

In this paper we have described an object-oriented approach to knowledge representation, called *Logic Oriented Organised Knowledge System* (LOOKS). Our efforts were based on the belief that Prolog's expressive power consists of versatile interpretation so as to integrate various programming paradigms. This integration was accomplished by the use of *"world_demo"*, which amalgamated object-level and meta-level knowledge. The *"world_demo"* predicate was demonstrated in the design of the glaucoma diagnostic system. The proposed system is implemented in DEC-10 Prolog and application system called LOOKS/Glaucoma is working on the glaucoma diagnostic area.

The significances of present paper are summarized as follows:

- The object-oriented approach in LOOKS provides a clear structure for separating the object definition of the domain knowledge and the control of inference.
- Knowledge representation in LOOKS provides a rich semantic structure for describing structured objects with class hierarchies which are similar to a *"frame"* or a *"semantic net"* representation.
- The rule-based representation in LOOKS provides a support for unifying the existing production rules into a LOOKS framework.

- The meta-level predicates, *"world_demo"* in LOOKS provides a basis for integrating object-oriented and rule-oriented programming paradigms.
- The essential parts of LOOKS provides module libraries for implementing small scale knowledge representation system, mini-LOOKS which runs on micro-computer (NEC-PC9801F) using Prolog KABA (Prolog for micro-computer).

Finally, we have emphasized the meta-level control as a mean of integrating object-oriented and rule-oriented programming paradigms in LOOKS. We have, however, not yet adequately studied the memory management of the objects through the same concept. In order to accomplish the realistic application, we must consider the large memory space for objects. In that sense, we feel the strong necessity of Prolog machine with large memory space such as PSI machine which ICOT has developed (Yokota et al., 1983). Probably, LOOKS will be translated into ESP (Chikayama, 1984) which is a kernel language of PSI machine.

ACKNOWLEDGMENTS

The author would like to thank the members of ICOT Working Group No.4 for fruitful discussions. We would also especially like to thank Dr. Kazuhiro Fuchi, Director of ICOT Research Center, and Kouichi Furukawa, Chief of ICOT's Second Research Laboratory, who developed Mandala. The glaucoma knowledge was obtained from the collaboration with Drs. Yoshiaki Kitazawa and Shiroteru Shirato of Tokyo University Medical Center. We also thank Takashi Chikayama of ICOT for allowing us to use his macro commands TEX editor. Finally, thanks is due to the referees for useful comments and suggestions.

REFERENCES

Bobrow, D. G. and Winograd, T., An Overview of KRL: A Knowledge Representation Language. *Cognitive Science 1*, 1977

Bobrow, D. G. and Stefik, M., *The LOOPS Manual*, Memo KB-VLSI-81-13, Xerox Palo Alto Research Center, Calif., Aug. 1981, rev. Aug. 1982

Bowen, K. and Kowalski, R. A., *Amalgamating Language and Meta Language in Logic Programming*, School of Computer and Information Sciences, University of Syracuse, 1981

Brachman, R. J. and Levesque, H. J., *KRYPTON: A Functional Approach to Knowledge Representation, Fairchild Laboratory for Artificial Intelligence Research*, Fairchild TR No.639, 1983

Brown, H., et al., Palladio: An Exploratory Environment for Circuit Design. *IEEE tran. Computer*, 41-56, 1983

Bundy, A. and Welham, B., Using Meta-level Inference for Selective Application of Multiple Rewrite Rule Sets in Algebraic Manipulation, *Artificial Intelligence 16*, 189-212, 1981

Chikayama, T., *ESP Reference Manual*, ICOT Tech. Report, 1984

Davis, R., Content Reference: Reasoning about Rules, *Artificial Intelligence 15*, 223-239, 1980

Furukawa, K., Takeuchi, A. and Kunifuji,S., *Mandala: A Concurrent Prolog Based Knowledge Programming Language/System*, ICOT Tech. Report, 1984

Genesereth, M., *MRS: A Metalevel Representation System*, Working paper HPP-83-28, Stanford University, Heuristic Programming Project,June 1983

Kitakami, H., Kunifuji, S., Miyachi, T. and Furukawa, K., *A Methodology for Implementation of Knowledge Acquisition System, Proc. of 1984 International Symposium on Logic Programming*, 131-142,1984

Minsky, M., *A Framework for Representing Knowledge, The Psychology of Computer Vision (ed. Winston, P.)*, McGraw-Hill, 1975

Mizoguchi, F., *A Software Environment for Developing Knowledge Base Systems, JARECT, Computer Science & Technologies (ed. Kitagawa, T.)*, Ohm*North-Holland, 1982

Mizoguchi, F., *PROLOG Based Expert System, New Generation Computing*, 1, 1983

Mizoguchi, F., Honda, E. and Katayama, Y., *Kagami & Sugata: Design and Application of Object Oriented Based Knowledge Representation Language in Prolog, Proc. of the Logic Programming Conference '84*, 1-12, 1984

Nakashima, H., *Knowledge Representation in PROLOG/KR Proc. of 1984 International Symposium on Logic Programming*, 126-130, 1984

Shapiro, E. and Takeuchi, A., *Object Oriented Programming in Concurrent Prolog*, ICOT Tech. Report, TR-004, 1983

Shintani, T. and Mizoguchi, F., *An Approach to Design of Default Reasoning Systems in Knowledge Base*, Information Processing Society Journal, Vol. 24, No. 5, 605-613, 1983

Peirera, F. C. M. and Warren, D. H. D., *User's Guide to Decsystem-10 Prolog*, Occasional Paper No. 15, Department of Artificial Intelligence, University of Edinburgh, 1978

Weiss, S. and Kulikowski, C., *EXPERT: A System for Developing Consultation Models, Proc. of IJCAI-79*, 1979

van Melle, W., *A Domain-independent Production Rule System for Consultation Programs, Proc. of IJCAI-79*, 1979

Yokota, M., et al., *The Design and Implementation of a Personal Sequential Inference Machine: PSI, New Generation Computing*, 1, 1983

Zaniolo,C., *Object-oriented Programming in Prolog, Proc. of 1984 International Symposium on Logic Programming*, 265-270, 1984