NIPPON STEEL TECHNICAL REPORT No. 56 JANUARY 1993

UDC677.027.5 : 681.5

A Very Fast Rule-Based Inference Engine for Expert Systems

Yutaka Miyabe*1
Toshimitsu Baba*2
Osamu Dairiki*2
Yoshiteru Matsuo*2
Shinya Mizuno*2

Abstract:

An inference engine under development for expert systems is outlined. The general-purpose inference engine stores knowledge in the form of production rules and uses C++, an object-oriented language. Pattern matching and conflict resolution algorithms of far higher performance than those of conventional inference engines were developed, making it possible for the new inference engine to run fast enough to meet execution on large-scale and high-functionality knowledge bases that are expected to increase in the future. In order to simplify knowledge description while ensuring high descriptive capacity, the object-oriented concept is positively introduced in the conventional if-then format.

1. Introduction

Nippon Steel has developed far more than 100 expert systems, and many of them are in operation now. The trend of expert system development is reviewed here from some examples of late.

Fig. 1 shows the purposes for which recent expert systems have been developed. Many expert systems have been developed as investment items for improving plant productivity or software development efficiency. This means that we have passed the boom of expert system development and entered a settling period.

Fig. 2 classifies expert systems by the type of problem to be solved. Analysis expert systems predominate, as typically seen in fault diagnostic expert systems. There has been made a drastic improvement in functions which was not seen a few years ago, such as closed-loop automation for selecting and implementing corrective measures after diagnosis. Planning exert systems are increasing to meet material flow and production planning requirements.

Fig. 3 shows the modes in which expert systems are implemented. Reflecting the increasing scale and complexity of objects and problems, few expert systems are implemented in the classical rule-based format. Instead, many expert systems are implemented as hybrid systems through the combined use of procedural programming (programming with a conventional procedural language), application of fuzzy set theory, or utilization of neural network technology. In **Fig. 3**, there is one expert system implemented using only a procedural language. This expert system was rule-based in the prototype development phase, but was finally implemented using the procedural language alone. (The system was judged to be an expert system despite its final implementation style.)

As evident from the case studies, the future trend will be toward more complicated and larger practical systems. For example, planning and design systems to cover two or more processes or plants are considered to increase in number. These tendencies brought to light such problems as low operating speed and low integration of expensive commercial tools. In-house developed shells, like an analytical reasoning shell[1], a neural network shell[2], and hypothetical reasoning shell[3], are beginning to be utilized in practical systems.

Diagnostic systems supercede planning and other systems because of the difficulty of problem solving involved. Lately, shells[1] that are fully equipped with inference knowledge about

*1 Electronics & Information Systems Divisions Group
*2 Technical Development Bureau

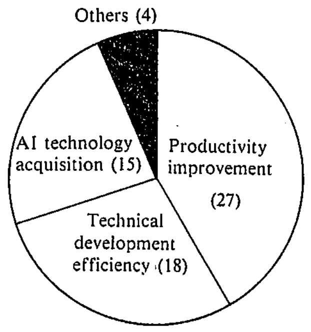


Fig. 1 Classification of expert systems by purpose of development

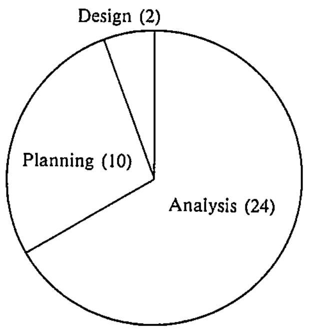


Fig. 2 Classification of expert systems by type of problem to be solved

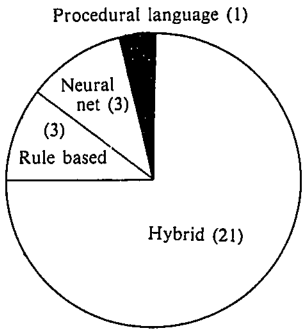


Fig. 3 Classification of expert systems by mode of implementation

diagnosis and that can be utilized by field users themselves have been put to use, improving the environment in which practical systems can be efficiently developed and maintained. In contrast, planning and design systems have no general-purpose problem solving techniques. Hypothetical reasoning proposed as an AI technique and optimization and mathematical programming proposed as operations research techniques are available as general algorithms. When these algorithms are applied to problem solving, however, they encounter many problems such as limited computer capacity and formulation difficulty. Most of the present expert systems are implemented as hybrid systems with problem-solving methods researched and developed as required.

The expert system inference engine introduced here was researched and developed for rule-based expert systems to discharge the core functions of hybrid systems in view of the trends noted above.

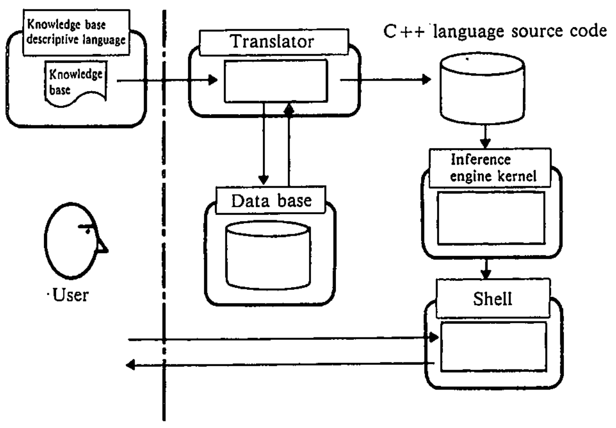


Fig. 4 Configuration of general-purpose inference engine

2. Outline of Inference Engine

2.1 Design philosophy

The inference engine developed as a prototype and tentatively named YAPS (for Yet Another Production System) belongs to the category of production systems. Production rules are compiled into a knowledge base and driven by the inference engine. Since the if-then (production rule) format is a knowledge representation method already popular on the field and has a high knowledge descriptive capability as required for an overall control and management core in future hybrid systems, it was adopted for the inference engine.

In view of the technical trends discussed above, the following requirements were set for the YAPS:

(1) High speed: A speed at least several times as high as that of existing inference engines is achieved. This requirement is set in order to secure a practical speed that is high enough to utilize a complex structure of knowledge (such as multiple contexts, hypothesis, and search).

(2) Adaptability to large-scale system: Production rules and data (working memory) elements on the order of thousands can be used, and a practically high enough inference speed can attained when the scale of the system is increased further.

(3) Excellent portability and extensibility: The inference engine is a basic building block of the system. It therefore can run on many hardware platforms, be integrated with other software, and be easily and safely customized itself.

2.2 System outline

The basic configuration of the inference engine is as shown in **Fig. 4**. The system is composed of five main parts. The knowledge base created in a knowledge base descriptive language is converted into the C++ language and utilized by the inference engine. The user interacts with the inference engine through the shell*1.

2.2.1 Rule (knowledge) base descriptive language

The language specification to transfer the knowledge of the human expert to the computer is a rule base descriptive language. It is based on the production rule representation (the so-called if-then format) that is the most typical knowledge representation method. The descriptive language specification was brought closer in similarity to the C++ language specification in order to increase compatibility with the C++ language, which is the basic language.

2.2.2 Knowledge base translator

YAPS converts user-described knowledge into the C++ code, compiles the code, and yields an executable expert system. (Knowledge is not executed by a shell as in classical expert systems.) The rule base language is converted into the C++ language by the knowledge base translator. The conversion as noted here is not conversion of knowledge into a procedure (algorithm)*2. Knowledge is declarative as it should be for an ex-

*1 The scope of the name YAPS is fluid within the development team. YAPS covers in a broad sense the entire scope of the system shown in Fig. 4 from the standpoint of the user, and refers in a narrow sense to the translator for converting the knowledge base into the inference engine kernel. In this article, YAPS is used in the broad sense.

*2 The conversion refers to preliminary definition of data structure to be used in drawing inferences as a C++ language class, to array structure of information necessary for the pattern matching algorithm, and to replacement of the action part of a rule into a C++ language function.

pert system, and inference is executed in an event-driven (nondeterministic) manner.

2.2.3 Data base

The translator parses the knowledge base and stores the results in the data base. The C++ code is generated from the information in the final stage of translation. The information is stored in the data base, and overloading detection, sorting, and other procedures are performed by the data base management system.

2.2.4 Inference engine kernel

The inference engine kernel is a drive mechanism for the basic functional elements of the inference engine, such as pattern matching, conflict resolution, and rule firing. By linking it to the knowledge base converted into the C++ code, a runtime module is obtained for the inference engine.

2.2.5 Shell function

The shell function provides interactive interfaces not only for the basic operation of inference but also for operations such as tracing inference and displaying working memory in the development phase*3.

2.3 Implementation and performance

At present, a prototype that fulfils the above-mentioned functions is complete. System development and implementation at the present stage are performed on an NS-SUN workstation, which calls for the following environment:

(1) Sun workstation (machine: Sun4/40; SPI, Sun OS 4.0.3*4)
(2) Sun C++ compiler (version 2.0)
(3) Sun Japanese Open Windows*5 (version 2.0; necessary only when using the graphic shell)
(4) db_VISTA*6 data base management system (version 3.10)

Fig. 5 shows the results of a run speed benchmark test. The "monkeys-and-bananas" problem[4], a classical problem for rule-based reasoning, was used in the benchmark test*7. The original problem involves one monkey and one bunch of bananas. The knowledge base was modified for the benchmark test so that two or more monkeys would compete for two or more bunches of bananas. The numbers involved were changed to study the correspondence between the data scale and the inference speed. OPS83 (version 2.2.1), recognized as one of the fastest commercially available systems, was selected as control for comparison.

Fig. 5 shows that YAPS is several times higher in absolute run speed than OPS83, is small in speed drop with increasing problem scale (scale factor), and expands its advantage with increasing scale. These results indicate that the prototype satisfies the requirements of system development.

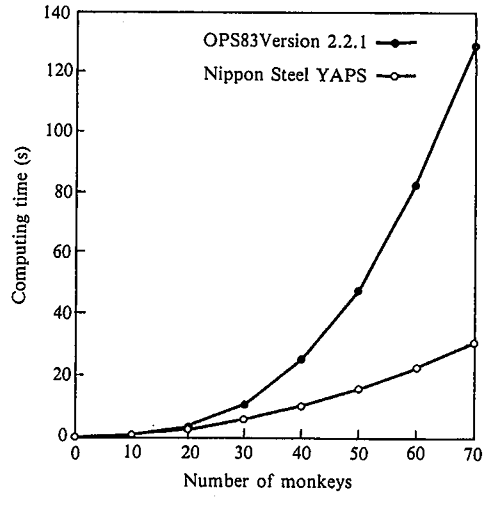


Fig. 5 Benchmark test results

3. Outline of YAPS Functions

The authors developed an original algorithm as central function for executing the inference cycle (match, select, and fire as described below).

Generally, the inference engine provides the user with the functions of proceeding with inference by matching the condition part of the knowledge stored in the knowledge base (production rules) with the data stored in the working memory and by firing rules in a chain action.

The inference process consists of the following steps:

(1) Match: Comparison of the condition part of a rule with the working memory (pattern matching)
(2) Select: Selection of rules and instantiations (conflict resolution)
(3) Fire: Execution of the action part of a rule

The inference process is usually performed until there are no more rules that can be fired.

Pattern matching and conflict resolution are processing tasks of very high load. Initial-stage production systems are reported to have consumed as much as 95% of the total computing time in pattern matching and conflict resolution[5]. Therefore, various studies have been made on high-speed algorithms for pattern matching[6,7]. As a result, today's high-performance inference engines are based almost without exception on pattern matching by the Rete[6] algorithm. The Rete algorithm is firmly established as a general-purpose pattern matching formula. It has already been improved from various points of view, leaving scarcely any more room for further improvement to its fundamental structure.

As a breakthrough to this situation, a new algorithm, called the D/O net[8], was developed for the pattern matching section, and another new algorithm, called the Y/M Select[8], was developed for the conflict resolution section*8. The prototype was developed after repeating careful research on software design and coding in the implementation phase. This effort resulted in achieving a basic performance that allows the high-speed running of a large-scale system as indicated by the benchmark test results. For lack of space, the details of algorithms are left for a separate

*3 Generally, a rule-based system makes it difficult to understand the processing flow as compared with procedural processing, and easy-to-understand information display is strongly required. YAPS allows the construction of various shells (user interfaces) from a text-based console to a high-functionality graphic console like the X-Window by utilizing libraries to provide such a function as required.

*4 Sun, SPI, and Sun OS are registered trademarks of Sun Microsystems Incorporated.

*5 The Japanese Open Windows is a registered trademark of NIHON SUN MICROSYSTEMS K.K.

*6 db_VISTA is a registered trademark of Raima Corp.

*7 The monkey tries to reach for a bunch of bananas hanging from the ceiling using a ladder or chair.

*8 The D/O net and Y/M Select are tentative names used within the laboratory.

report, and an outline description of YAPS with its features are presented hereunder.

3.1 System development by object-oriented description

YAPS is basically a rule base descriptive language like the one used in many inference engines. Once data types and rules are declared, inference proceeds with data tokens that are dynamically created and removed during the inference. The data types, which are also called templates, and the rules are approximately declared as follows:

```
deftemplate monkey
  {
    class:
      string description;
    instance:
      symbol holds;
    .......
  };

defrule mb1: mb_do_holds
  {
    (monkey on == ceiling;);
  =>
    make.....
  };
```

Deftemplate declares the data type, named monkey, and defines slots such as "description" and "holds." The rule declared by defrule is named mb1 and can be fired when there is a monkey reaching the ceiling. If there is such an environment that the graphic shell on the window system can be used, coding is not required, and the development environment shown in **Fig. 6** and the run environment shown in **Figs. 7** and **8** can be utilized*9.

YAPS is characteristic in that it makes positive use of object-oriented styles for functions and description types while adopting a conventional knowledge representation method. Its features are excerpted below from the language specification$^{9)}$.

3.1.1 Inheritance

Hierarchical inheritance can be utilized in the declaration of templates and rules.

Templates may be inherited in the form of an entire type or a specific slot. The template monkey declared in the foreground window of **Fig. 6(a)** has three slots description, on, and holds defined. The slot location defined by the parent :item is a slot of the template monkey. All the templates declared for the parent are inherited to the child as types. This is also true of slot inheritance. As shown in **Fig. 6(a)**, the user-defined slot type OBJECTS can be declared by utilizing inheritance like the template.

```
defsymbol OBJECTS: Parent_Symbol_Type
  {…}
```

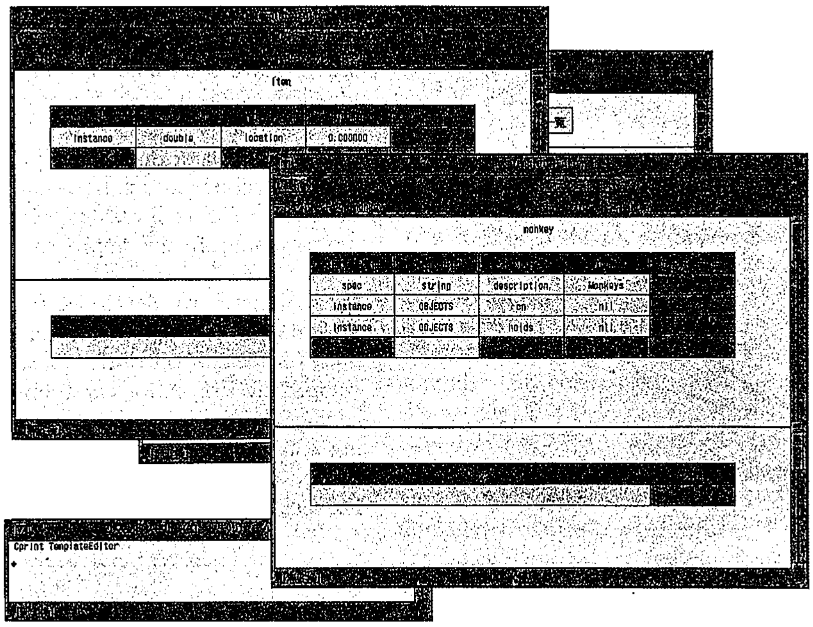
Rules have their own hierarchical inheritance. The left-hand sind (LHS) of a parent rule, including bound variables, is inherited to the LHS of a child fule. The conditional statement

on == ceiling

is defined for the rule mb1 edited in **Fig. 6(b)**. Together with the conditional statement (shown in the lower part of the window) inherited from the parent rule mb_do_holds, it constitutes the actual LHS of the rule mb1.

3.1.2 Data structure

When a data type is declared as a template, it is still the definition of the type. Structured data is created only by using the make function, for example. The data is generally deposited on



instance	double	location	0.000000

spec	string	description	Monkeys
instance	OBJECTS	on	nil
instance	OBJECTS	holds	nil

```
Cprint TemplateEditor
◆
```

Fig. 6(a) Template declaration by graphic shell

*9 The template declaration is performed as shown in Fig. 6(a), and the rule declaration is performed as shown in Fig. 6(b). The way an inference is carried out using the knowledge is shown in Figs. 7 and 8.

a logical work space called the working memory.

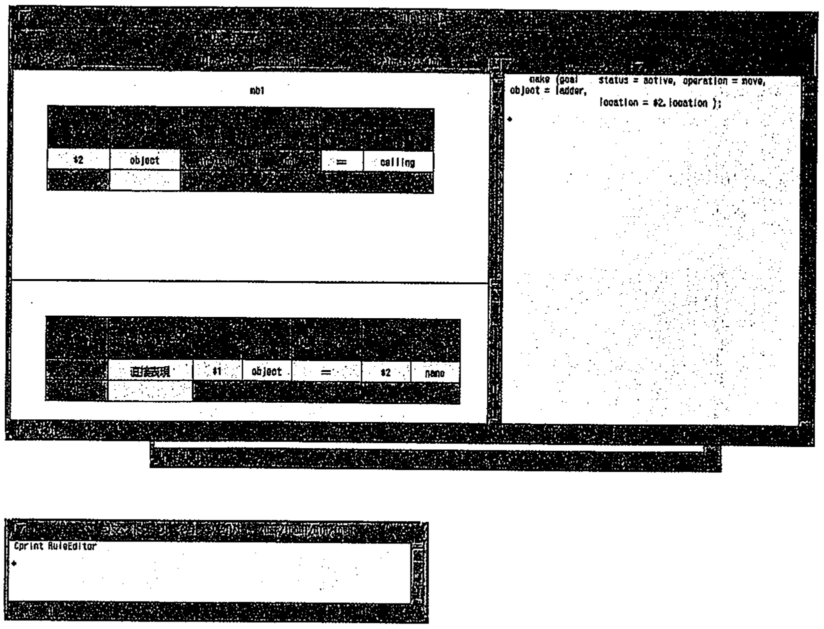
Data are usually transitory information, and are generated and deleted as the inference process proceeds. In their use in actual systems, however, they are often utilized also as persistent data. The data to be recognized as objects can be identified and directly referred to when they are declared persistent.

Variables (slots) are classified into class variables and instance variables. The class variable has such a value per template type that is shared by all instances, whereas the instance variable has a value intrinsic to each data (instance). Demons can be set for these slots. The demon constantly monitors a slot and implements an assigned task when the slot value is updated. It is a function effective in ensuring data consistency, simplifying the description of rules, and improving the readability of rules. As shown in **Fig. 6(a)**, these functions can also be utilized as from the graphic shell.

3.1.3 Modularization

The concept of module was introduced in response to the large-scale and high-functionality knowledge base for which YAPS is intended.

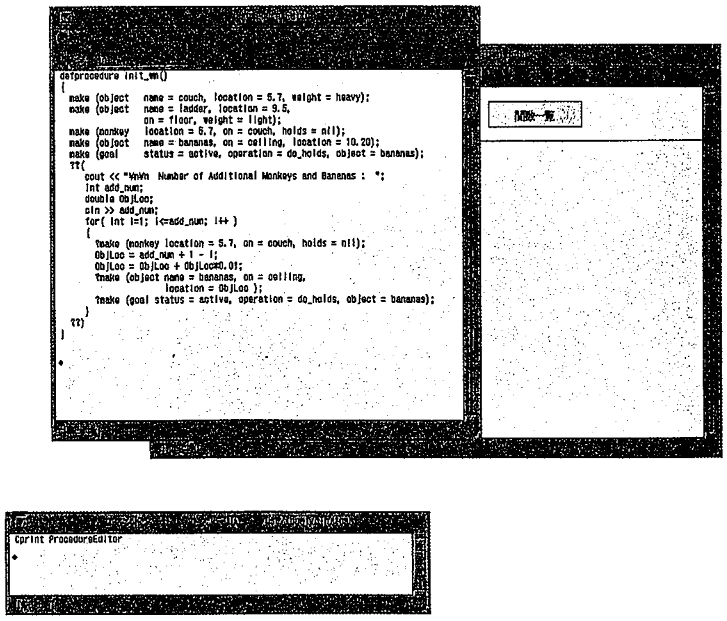
Rules can be grouped according to context and importance. This function is significant in organizing the structure of the rule



```
    make (goal    status = active, operation = move,
object = ladder,
                  location = $2.location );
◆
```

```
Cprint RuleEditor
◆
```

Fig. 6(b) Rule declaration by graphic shell



```
defprocedure init_wm()
{
  make (object   name = couch, location = 5.7, weight = heavy);
  make (object   name = ladder, location = 9.5,
                 on = floor, weight = light);
  make (monkey   location = 5.7, on = couch, holds = nil);
  make (object   name = bananas, on = ceiling, location = 10.20);
  make (goal     status = active, operation = do_holds, object = bananas);
  ??(
     cout << "¥n¥n  Number of Additional Monkeys and Bananas :  ";
     int add_num;
     double ObjLoc;
     cin >> add_num;
     for( int i=1; i<=add_num; i++ )
     {
       ?make (monkey location = 5.7, on = couch, holds = nil);
       ObjLoc = add_num + 1 - i;
       ObjLoc = ObjLoc + ObjLoc*0.01;
       ?make (object name = bananas, on = ceiling,
                     location = ObjLoc );
       ?make (goal status = active, operation = do_holds, object = bananas);
     }
  ??)
}
◆
```

```
Cprint ProcedureEditor
◆
```

Fig. 6(c) C++ description by graphic shell base and improving the readability of the rule base and, at the same time, in improving the inference speed by utilizing only rule groups necessary for the time being. Hierarchical inheritance, adopted for the rules and templates, can be introduced also for this purpose when necessary.

A module of greater significance is relationship with other languages, including the C++ language. Conversation between a rule-based system and another functional (program) module written in other language is performed on a structured data basis. The global data type is available as a means of access from other languages. This is template data globally declared in addition to the above-mentioned persistent declaration. As in the case of the rule-based system, its contents can be updated when referred to by name from other modules. C++ direct coding provides access from the rule-based system. **Fig. 6(c)** shows the right-hand side (RHS) of a rule. The portion enclosed with double question marks (??) is directly written in the C++ language.

3.2 Example of implementation

The way the modified monkeys-and-bananas problem is solved in the benchmark test is as shown in **Figs. 7** and **8**. **Fig. 7** shows a working memory browser. A menu is shown at the upper left, and Working Memory Operations (the title at the upper part of the window) is opened as entry pane. A data element of the monkey type is being newly created in Make Working Memory, and the contents of an existing data element of the object type are being changed in Modify Working Memory. **Fig. 8** shows a rule browser. Pattern matching situation browsing and rule control (such as fire priority modification) are being performed by operation on a rule group basis (Rule Group Display and Rules List) and by direct operation on specific rules (Rule Display).

3.3 Important optional functions

The distribution of YAPS has been started only for in-house trial use. Its basic type is a production system having the above-mentioned features. YAPS has event-driven forward reasoning by the pattern matching function and conflict resolution like that of MEA[4].

Hypothetical reasoning, backward reasoning, and Japanese support can be established as options. Since YAPS is developed

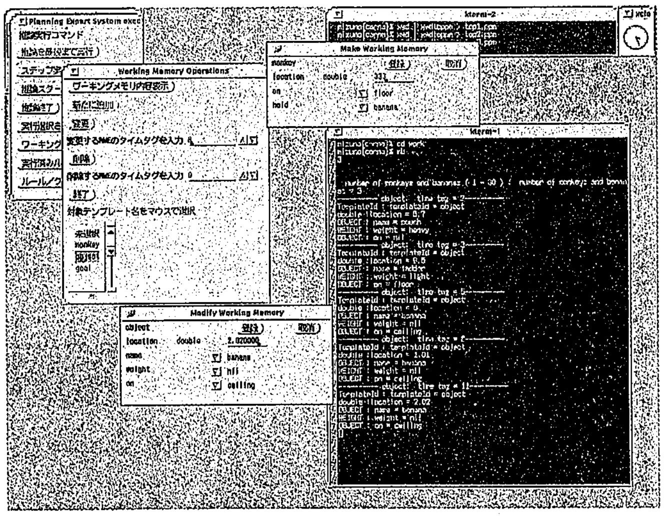


Fig. 7 Runtime template browser

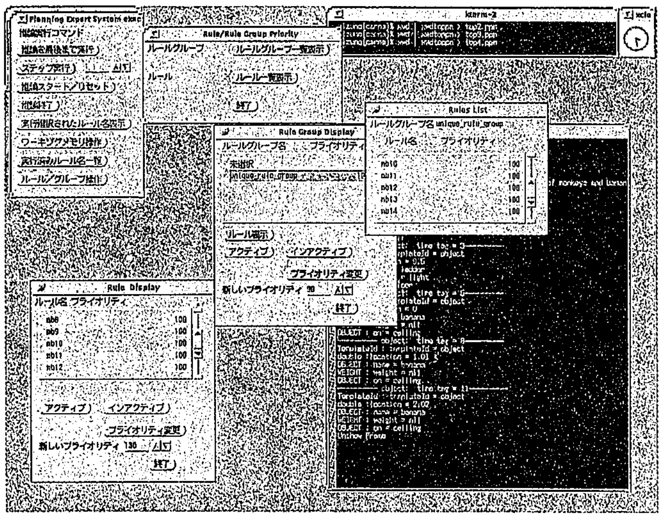


Fig. 8 Runtime rule browser

in C++, an object-oriented language, these optional additions can be very easily made through a link with the corresponding C++ class library and the inheritance of basic classes.

The shell can be easily modified into a simple version without graphics. The simplified version of YAPS consists of the C++ code after translation of the knowledge base and the original inference engine. It can be implemented on hardware platforms that can utilize the C++ or C language and can provide high software portability.

4. Conclusions

The general-purpose inference engine, tentatively named YAPS and developed for expert systems, has been outlined above. Although only its prototype is complete now, it has been confirmed that YAPS is capable of performing several times faster than commercial inference engines even when utilized in large-scale systems.

In its continued evolution, emphasis will be placed on the improvement of the originally developed algorithm to serve as its core, and on its functional enhancement as a practical inference engine. In developing a higher-level version of YAPS, we will strengthen the object-oriented characteristics of YAPS, realize an integrated utilization environment with other systems and programs, and develop YAPS as a functional module of a distributed cooperative system.

References

1) Minami, E., Miyabe, Y., Dairiki, O.: ESTO: A Practical Environment for Industrial Diagnostic Systems. Proc. Industrial & Engineering Applications of Artificial Intelligence. 1990, p. 684-691
2) Takada, H.: Development of Parallel Processing Neural Net Simulator. 2nd Transputer/Occam Int. Ntl. Conf. 1989, p. 123-132
3) Ohgai, H., Mishima, Y., Harada, T.: Realization of High-Speed Hypothetical Inference Mechanism. Turing Machine. 2 (5), 20-30 (1989)
4) Brownston, L. et al.: Programming Expert Systems in OPS5. Reading, Addison-Wesley, 1986, p. 62-71
5) McDermott, J., Newell, A., Moore, J.: The Efficiency of Certain Production System Implementations. In Waterman, D.A., Hayes-Roth, F.: Pattern-Discreted Inference Systems, New York, Academic Press, 1978, p. 155-176
6) Forgy, C.L.: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. Artificial Intelligence, 19, 17-37 (1982)
7) Tano, S., Masui, S., Sakaguchi, S., Funabashi, M.: A Fast Pattern Match Algorithm for a Knowledge Based System Building Tool-EUREKA (in Japanese). IEIC Trans. 28 (12), 1255-1268 (1987)
8) Nippon Steel Corporation: Private communication, September 1991
9) Nippon Steel Corporation: Private communication, September 1991