A Knowledge Compilation Method Through Conversion of Symbolic Rules and Facts into Functions

YASUHIRO KOBAYASHI*, TOORU MITSUTA* and YUTAKA WADA*

This paper describes a practical knowledge compilation method which is based on conversion of knowledge from rules and facts, in a symbolic data form, to those in a functional form through a semi-automatic translation process. Knowledge is doubly compiled at a declarative/procedural level of knowledge representation and at a Lisp/machine code level of computer language.

The proposed method is applied to a knolwledge-based system for automatic pipe route planning in the area of industrial plant layout design. The inference program, based on the proposed method solves route priority assignment problems more than 50 times efficiently than a reference inference program in logic programming style. The results suggest that the proposed knowledge compilation method should be a useful tool, through its coordination of the requirement of inference efficiency in system utilization and that of knowledge transparency in system construction and maintenance.

1. Introduction

Many inference mechanisms have been proposed for knowledge-based systems in accordance with the various kinds of knowledge representations available. Adequate knowledge representation is a key in problem solving, and strongly depends on the domain and problem to be treated [1]. Production systems, frame representation and predicate calculus are typical inference mechanisms. One basic, shared concept is that their inference process is based on a central interpreter which interprets knowledge in the form of symbolic data. While this characteristic of the inference mechanisms might be appropriate for use in knowledge-based systems with limited requirements of inference performance for narrow domains, it is, however, important to enhance the efficiency of inference in applications of knowledge-based systems to large-scale and/or real time problems, which are frequently encountered in engineering.

Inefficiency, for example, has been encountered in development of a knowledge-based system for automatic pipe route planning in plant layout design [2]. The system task is composed of two steps; (i) the knowledge reduction step in which routing knowledge is reduced to a planning specification of a route in the frame form obtained from if-then rules by symbolic manipulation and (ii) the path finding step in which the optimal route satisfying the planning specification between fixed start and goal points is determined by numerical computation. The knowledge reduction step is based on backward chaining by a conventional inference program KRIT-PC [3] and consumes most of computer CPU time. Then one practical incentive is to develop a method for improving the inference efficiency.

Major requirements for knowledge-based systems in engineering applications may be summarized as the efficiency of knowledge utilization and the transparency of knowledge representation. Two approaches are possible to realize knowledge-based systems which satisfy these requirements:

(a) employ a software system of high transparency and improve its efficiency by advanced hardware, or

(b) combine a software system of high transparency with a software system of high efficiency.

The first approach is being actively pursued in computer science. New programming styles have been proposed to simplify computer aided problem solving and enhance the transparency of the problem solving process. Typical examples include logic programming and object oriented programming, which potentially realize advanced knowledge-based systems through the first straightforward approach. To achieve very efficient inference in advanced knowledge-based systems, the basic research has been directed towards development of advanced computers with parallel processing architecture [4]. However, much remains to be done for this first approach to reach the stage of practical use.

The second approach offers a practical means, applicable to the inefficiency problem in present knowledge-based systems. It is often referred to as a knowledge compilation method and several types are

*Energy Research Laboratory, Hitachi Ltd. 1168 Moriyama-cho, Hitachi, Ibaraki 316, Japan.

discussed in Ref. [5]. One well-known example is translation from if-then rules to a network to describe incremental change of known facts, as proposed by Forgy [6] for production systems. It is intuitively understandable that a declarative representation, like if-then rules, is highly transparent and a procedural representation, like programs, is highly efficient in the present problem-solving systems [7]. Reddy [8] described the transformation process from logic programming based on declarative knowledge to the functional programming based on procedural knowledge. Karey and Juell [9] proposed an expert system compiler which translates a rule-based language to a set of *C* language procedures. The former language is based on knowledge representation in the declarative form and the latter language is based on that in the procedural form.

This paper describes a more practical, but ad hoc approach, to knowledge-based systems of improved inference efficiency. The proposed knowledge compilation method is based on the conversion of knowledge from rules and facts in symbolic data form to those in functional form through a semi-automatic translation process. This method attempts to satisfy two major requirements in knowledge-based system utilization; the efficiency of inference especially required for practical use of a knowledge-based system and the transparency of knowledge representation required during the construction and maintenance of a knowledge base.

2. Method

2.1 Knowledge Compilation

In this paper knowledge compilation means conversion of knowledge representation to give improved efficiency of knowledge utilization. Knowledge in the declarative form is compiled to that in the procedural form at the knowledge representation level. This is the first compilation in which rules and facts in a symbolic data form are translated into those in a functional form. Once knowledge is compiled into the procedural level, procedural knowledge in a description language can be compiled to that in a faster language at the computer language level. This is the second compilation in which the procedure, written in Lisp, is translated into that in machine code. This knowledge compilation is a powerful way to realize efficient knowledge utilization by advantageous combination of the first and second compilations. The efficiency due to this double compilation is brought about by the use of rules and facts represented in the functional form.

The knowledge handling process based on the proposed method is characterized by three major stages:

(i) transparent description of rules and facts in predicate calculus syntax, which is suitable to construction and maintenance of knowledge bases;

(ii) semi-automatic translation from rules and facts written in stage (i) to procedures in machine code; and

(iii) efficient utilization of doubly compiled procedures, generated at stage (ii), which is suitable to practical use of knowledge-based systems.

2.2 Comparison of Knowledge-Based System

A knowledge-based system with rules and facts in the functional form can be compared with a conventional knowledge-based system with rules and facts in the symbolic data form. (The inference method in the former is referred to as the proposed method and that in the latter as the conventional method.) The anticipated behavior of inference programs based on these two inference methods is described for a typical backward chaining process. The inference is initiated by asking the inference program a question about whether the proposition of a a hypothetical assertion is true or not. The inference program tries to prove the hypothetical assertion through backward chaining. Both methods can handle the proposition which contains an unknown variable or variables.

(1) Conventional method

The following steps are taken by the inference program using rules and facts in the symbolic data form.

(i) Focus on the proposition of the primary question to be answered. This is the hypothesis about the conclusion to be proved by backward chaining.

(ii) Repeat steps (ii−a), (ii−b) and (ii−c) to answer the primary and secondary questions, if applicable.

(ii−a) Find a fact proposition which matches the currently focused-on proposition. If such a fact is found, the present question is answered affirmatively.

(ii−b) Find a rule the conclusion proposition of which matches the currently focused-on proposition. If such a rule is found, ask a secondary question about whether its premise proposition can be true or not. Go to step (ii), focusing on the premise proposition.

(ii−c) If neither fact nor rule is found, the present question is answered negatively.

(iii) Return the answer to the primary question about whether the hypothetical assertion formed at step (i) is true or not. If the hypothesis contains unknown variables, the value for each variable is also obtained.

Thus, the inference program interprets the facts and rules in the knowledge base to process these steps.

(2) Proposed method

The following steps are taken by the inference program using rules and facts in the functional form.

(i) Focus on a keyword predicate in the argument of the function invocation for the primary question to be answered.

(ii) Repeat steps (ii−a), (ii−b) and (ii−c) to answer the primary and secondary questions, if applicable.

(ii−a) Invoke functions of facts which are associated with the focused keyword predicate. After the invoked functions return the result, if it is affirmative, the present question is answered affirmatively.

(ii−b) Invoke functions of rules the conclusion of which is associated with the focused-on keyword predicate. The invoked functions then cause function invocations to ask whether the premise of the rules can be true or not. Go to step (ii), focusing on the keyword predicate in the argument of the function invocation for the secondary question.

(ii−c) If neither fact nor rule function return an affirmative result, the present question is answered negatively.

(iii) Return the answer to the primary question about whether the hypothetical assertion formed at step (i) is true or not. If the hypothesis contains unknown variables, the value for each variable is also obtained.

The inference program utilizes the flow control mechanism embedded in the description language and does not need to control explicitly the step-wise process of backward chaining. The functions of facts and rules operate as if they know what they should do when they are invoked. This system configuration, based on atomic procedures corresponding facts and rules, assures high modularity of the knowledge base in the compiled form.

3. Inference Program

The inference method based on the double knowledge compilation was implemented as an inference program.

3.1 Program Specification

The basic specification of the program are stated in the following.

(a) Rules and facts are originally written in the syntax of KRIT-PC. [3] The system KRIT-PC is based on first order predicate calculus and is employed as a reference inference program with rules and facts in symbolic data form. The KRIT-PC program is comparable to Prolog, but it offers better control mechanisms for inference.

(b) Propositions of rules and facts are simple 3-tuples of the form [attribute object value] and may contain unknown variables for objects and/or values. The attribute of this expression is often referred to as a keyword predicate in this paper.

(c) The language for system description is Lisp.

(d) The inference type to be realized and tested is backward chaining.

3.2 Representation of Rules and Facts

(1) Rules

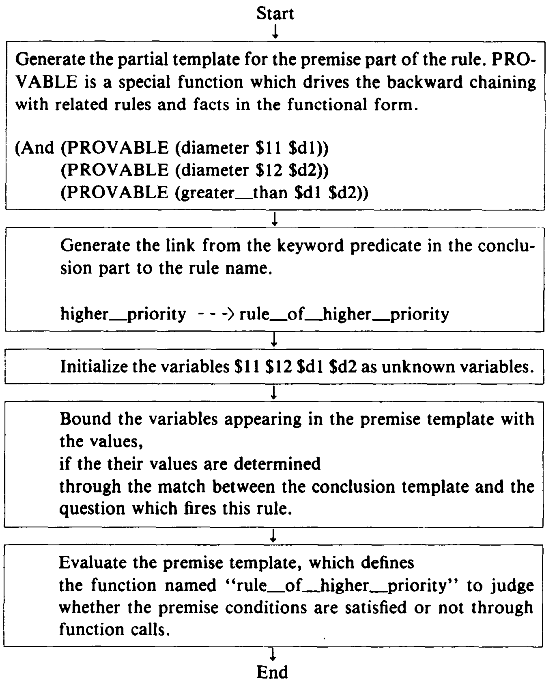
A simplified sample rule is taken to illustrate rules in the symbolic data form and the functional form. The sample rule is selected from the knowledge base of a system for automatic pipe route planning and the comparison is made in Fig. 1. A rule for pipe priority assignment is shown in Fig. 1(a). The symbolic data in Fig.

```
If the diameter of pipe L1 is D1,
   the diameter of pipe L2 is D2, and
   D1 is greater than D2,
then pipe L1 is given higher priority than
   pipe L2.
```

(a) Rule for pipe priority assignment

```
(If (And (diameter $l1 $d1)
         (diameter $l2 $d2)
         (greater_than $d1 $d2))
   (higher_priority $l1 $l2))
```

(b) Rule in symbolic data form



Start

Generate the partial template for the premise part of the rule. PROVABLE is a special function which drives the backward chaining with related rules and facts in the functional form.

```
(And (PROVABLE (diameter $l1 $d1))
     (PROVABLE (diameter $l2 $d2))
     (PROVABLE (greater_than $d1 $d2))
```

Generate the link from the keyword predicate in the conclusion part to the rule name.

higher_priority - - -> rule_of_higher_priority

Initialize the variables $l1 $l2 $d1 $d2 as unknown variables.

Bound the variables appearing in the premise template with the values,
if the their values are determined
through the match between the conclusion template and the question which fires this rule.

Evaluate the premise template, which defines
the function named "rule_of_higher_priority" to judge whether the premise conditions are satisfied or not through function calls.

End

(c) Rule in functional form

Fig. 1 Comparison between rules in symbolic data form and functional form.

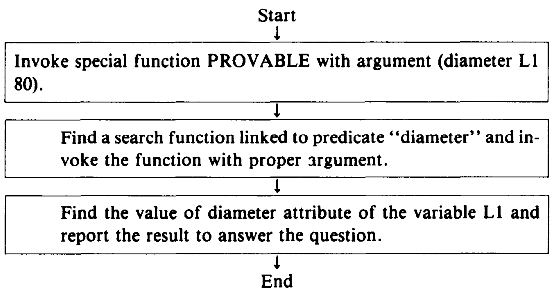
1(b) are declarative representations of the rule and contain no information about how to use themselves to realize the inference process. The function in Fig. 1(c) is the procedural representation and contains information about how to behave to realize inference process when invoked from other functions. The procedure which is embedded into the rule function is shown in Fig. 1(c) as five steps. The function of the rule judges if a proposition related to "pipe diameter" and a proposition related to "greater than" in the premise of the rule are all true on the basis of the information available when invoked from other functions. Whether the focused-on conclusion proposition can be true or not is asked by the invocation of appropriate functions to check the

The diameter pipe L1 is 80 mm.

(a) Fact about pipe diameter

(diameter L1 80)

(b) Fact in symbolic data form



Start

Invoke special function PROVABLE with argument (diameter L1 80).

Find a search function linked to predicate "diameter" and invoke the function with proper argument.

Find the value of diameter attribute of the variable L1 and report the result to answer the question.

End

(c) Fact in functional form

Fig. 2 Comparison between facts in symbolic data form and functional form.

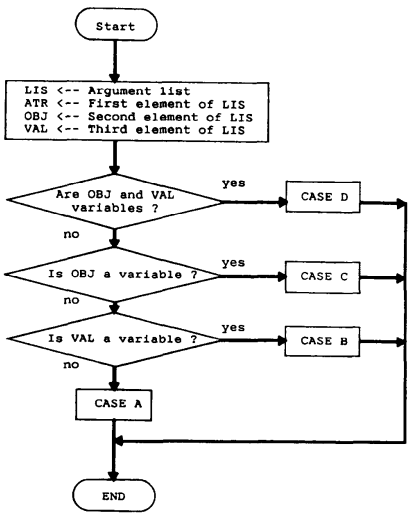

Start
LIS <-- Argument list
ATR <-- First element of LIS
OBJ <-- Second element of LIS
VAL <-- Third element of LIS
Are OBJ and VAL variables ?
yes
CASE D
no
Is OBJ a variable ?
yes
CASE C
no
Is VAL a variable ?
yes
CASE B
no
CASE A
END

Fig. 3(a) General procedural flow for fact search.

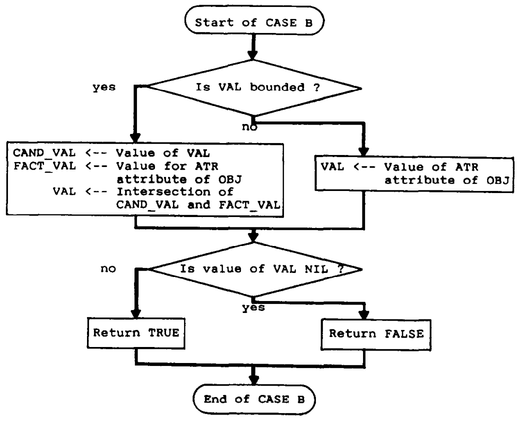

Start of CASE B
Is VAL bounded ?
yes
no
CAND_VAL <-- Value of VAL
FACT_VAL <-- Value for ATR attribute of OBJ
VAL <-- Intersection of CAND_VAL and FACT_VAL
VAL <-- Value of ATR attribute of OBJ
Is value of VAL NIL ?
no
yes
Return TRUE
Return FALSE
End of CASE B

Fig. 3(b) Partial general flow for CASE B.

propositions appearing in the premise of the rule and by evaluating the returned value from the function.

Two stages are adopted to realize the inference process by function invocation: the first stage generates a function to check propositions and the second executes this function. Template is used as the prototypical symbolic datum for function definition, which can be interpreted as the "source list" of a function definition program. The function source list to check propositions on the premise of the rule is cast on the template. The template is "EVAL"ed in Lisp to define the objective rule function. The first stage is covered by steps from 1 to 4 and the second stage is step 5. Two special functions PROVABLE and UPDATE appear in the template used in stage 1. PROVABLE drives the backward chaining with related rules and facts in the functional form. UPDATE manages the status of unknown variables which are unified in the course of backward chaining. A rule in the functional form is translated from the corresponding rule in the symbolic data form by one to one mapping.

(2) Facts

A sample fact is taken to illustrate facts in the symbolic data form and the functional form. This fact is related to the sample rule given in section (1). The comparison of the forms is made in Fig. 2. A fact about pipe diameter is shown in Fig. 2(a). The symbolic data in Fig. 2(b) are declarative representations of themselves and contain no information about how to use the fact to realize the inference process. The function in Fig. 2(c) is the procedural representation and contains information about how to realize the inference process when invoked from other functions to check fact propositions.

The procedure which is embedded into the fact function is shown in Fig. 2(c) as three steps. The fact function is simpler than the rule function, since it handles only one proposition, not the logical combination of propositions. When the pipe name and diameter value are known, the function of the sample fact behaves as a simple check function. The function returns "true", if and only if the value of the pipe diameter is correct in the knowledge base. It returns the value of the pipe diameter if it is found in the knowledge base, when the pipe name is known, but the diameter is not. The fact function can be obtained from that in the symbolic data form through one-to-one translation, as in the rule function. Unlike the rule function, the fact function can be provided by using several kinds of general purpose primitive search functions, because checking fact proposition can essentially be realized by several kinds of basic search tasks. The fact function behaves as if it knows what kind of search task is required to check the fact and which general purpose search function should be invoked from it.

(3) Fact search function

The function LOOKATR is one primitive search function and a search function appears in step 2 of Fig. 2(c). The general flow of this search function is shown in Fig. 3(a) and (b). LOOKATR treats a wide range of search tasks and covers various cases where objects and/or values are known, unknown or partially known by CASE $A-D$ in Fig. 3(a). The value is referred to as known, if it is not a variable or if a final value is given to the variable. The value is referred to as partially known, if no value is given when the inference is initiated and candidates of its final value are obtained in the course of backward chaining. The simple fact in Fig. 2 does not require such general functions. The procedural flow in Fig. 3(b) gives CASE B, as a case of them, in which VAL is a variable but OBJ is not in the proposition [ATR OBJ VAL].

The fact function invokes the proper type of search function which is specific to the fact, checks if the fact is true or not, and returns the result, when invoked from other functions. Fact propositions are represented by structured Lisp objects. Since the proposition [attribute object value] is structured as a frame, the search for the value is effecient once the object is pointed out in the data retrieval process. The search function LOOKATR uses the procedure in the following to find the proposition with unknown objects.

(i) Find the type of variable OBJ on the basis of a relationship between objects and attributes.

(ii) Find the instances of objects from the set of instance sets which is given for the type of objects.

(iii) Set the union of instances as a candidate value, which is partially known. A partialy known value is screened to the final value throughout the inference process.

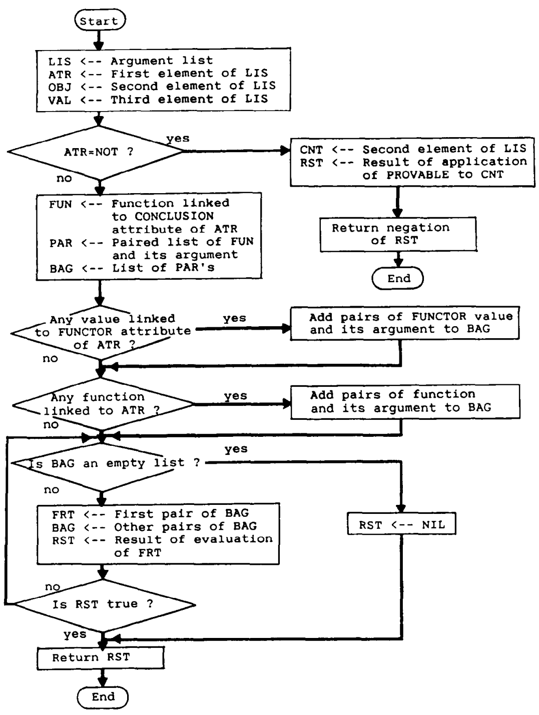
3.3 Control of Inference

In the knowledge-based system with rules and facts in the functional form, the flow of inference is inherently controlled by the function call mechanism embedded in the description language. A special function is provided to tune the flow control of the inference process realized by the function calls. The function PROVABLE manages detailed control of backward chaining. The general flow of PROVABLE is shown in Fig. 4. In the course of backward chaining to judge whether a given proposition X is true or not, intermediate propositions Y are secondarily hypothesized and tested. Two basic processes are used to judge whether the proposition Y is true or false from the rules and facts in the knowledge base.

(i) Find whether proposition Y exists as a fact.

(ii) Find the rule which has proposition Y in the conclusion and proposition Z in the premise and replace Y by new intermediate proposition Z.

These processes are embodied in the present inference



Start
LIS <-- Argument list
ATR <-- First element of LIS
OBJ <-- Second element of LIS
VAL <-- Third element of LIS
ATR=NOT ?
yes
no
CNT <-- Second element of LIS
RST <-- Result of application of PROVABLE to CNT
Return negation of RST
End
FUN <-- Function linked to CONCLUSION attribute of ATR
PAR <-- Paired list of FUN and its argument
BAG <-- List of PAR's
Any value linked to FUNCTOR attribute of ATR ?
yes
no
Add pairs of FUNCTOR value and its argument to BAG
Any function linked to ATR ?
yes
no
Add pairs of function and its argument to BAG
Is BAG an empty list ?
yes
no
FRT <-- First pair of BAG
BAG <-- Other pairs of BAG
RST <-- Result of evaluation of FRT
RST <-- NIL
no
Is RST true ?
yes
Return RST
End

Fig. 4 General flow of function PROVABLE.

program by the following three schemes.

(a) If a special search function is defined for the intermediate proposition, then call the function directly. The name of the special search function is given in FUNCTOR attribute of the keyword predicate of the proposition.

(b) If a general purpose search function is specified for the intermediate proposition, then call the function. The name of the search function is also given in FUNCTOR attribute of the keyword predicate of the proposition.

(c) If a rule function the conclusion proposition of which matches the intermediate proposition is found, then call the function. Chaining with the rule function is continued to call the function, the name of which is attached to the CONCLUSION attribute of the keyword predicate of the intermediate proposition.

In the flow of Fig. 4, the variable BAG is the template of the flow control function which sequentialy executes three schemes (a), (b) and (c), and is generated by symbolic manipulation. The flow control function is defined on the basis of this template in PROVABLE. The inference is initiated by invoking the function and the inference result is returned as the value of the function invoked. Thus, the function PROVABLE can locally control the flow of the inference process which is based on a function call mechanism offered from the description language Lisp.

3.4 Semi-Automatic Conversion of Knowledge Representation

The knowledge compilation method proposed here is based on the translation from rules and facts in symbolic data form into those in functional form. Practical use of knowledge-based systems requires that this conversion process should be done efficiently and systematically. It is, therefore, desirable to automate it as much as possible. This conversion of knowledge representation is, however, only semi-automatic in a sense that a limited number of basic functions and data must be prepared manually prior to the main translation process.

(1) Translation of rules

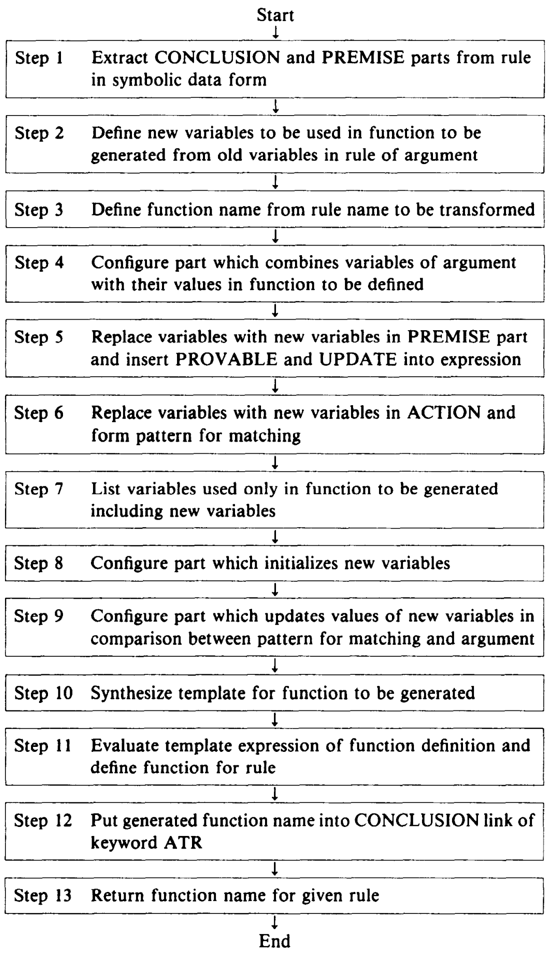
The general flow of the translation of a rule is shown in Fig. 5. First of all, the template for the prototype of a function to be defined is generated by symbolic manipulation. Then the function is defined by evaluating the template, since the template is the function definition statement in Lisp. This activates the function of the objective rule. The major elements of the template are illustrated in Fig. 6. They are parts of the source code which describes the corresponding rule function and their essential part is extracted from the rule in the symbolic data form. Steps 1 to 10 (Fig. 5) are employed to combine parts of the source code to produce a template for rule function definition. The key code to call the rule function at step 1 is stored at step 12. The name of the rule function is added to the CONCLUSION attribute of the keyword predicate in the concluding proposition of the rule.

(2) Translation of facts

Two steps are used for translation from rules and facts in the symbolic data form to those in the function form.

(a) The proposition of the fact tuple is translated to the more accessible structured Lisp object. As an example, the proposition [ATR OBJ VAL] is used to set the value of the ATR attribute of the OBJ frame as VAL.

(b) The search method is specified by adding the name of primitive search function to the FUNCTOR attribute of the keyword predicate of the fact proposition. For example, The diameter attribute of the pipe frame has the name of the general purpose search function LOOKATR in its FUNCTOR attribute.



Start
↓

Step 1	Extract CONCLUSION and PREMISE parts from rule in symbolic data form
Step 2	Define new variables to be used in function to be generated from old variables in rule of argument
Step 3	Define function name from rule name to be transformed
Step 4	Configure part which combines variables of argument with their values in function to be defined
Step 5	Replace variables with new variables in PREMISE part and insert PROVABLE and UPDATE into expression
Step 6	Replace variables with new variables in ACTION and form pattern for matching
Step 7	List variables used only in function to be generated including new variables
Step 8	Configure part which initializes new variables
Step 9	Configure part which updates values of new variables in comparison between pattern for matching and argument
Step 10	Synthesize template for function to be generated
Step 11	Evaluate template expression of function definition and define function for rule
Step 12	Put generated function name into CONCLUSION link of keyword ATR
Step 13	Return function name for given rule

↓
End

Fig. 5 General flow for rule generation in functional form.

4. Application of Method

4.1 Sample Knowledge-based System

The proposed knowledge compilation method is applied to a knowledge-based system for automatic pipe route planning during plant layout design. Making priority assignments to individual pipe routes, which are encountered in the pipe route planning, are made aided by designer's knowledge.

The knowledge-based system described in Fig. 7 is chosen as an example system to demonstrate the devel-

Element 1:	Function name for translation of given rule
Element 2:	Function name for given rule
Element 3:	Function name for function type definition
Element 4:	List of arguments
Element 5:	Comments for function to be generated
Element 6:	Function name for program statement
Element 7:	List of local variables
Element 8:	Local binding variables in arguments and their values
Element 9:	Initialization of variables used in template for rule description
Element 10:	Definition of premise part of template
Element 11:	Definition of conclusion part of template
Element 12:	Update of value of template variables through matching between template and arguments
Element 13:	Evaluation of premise template with updated template variables
Element 14:	Return evaluation result

Fig. 6 Configuration of template for rule function definition statement.

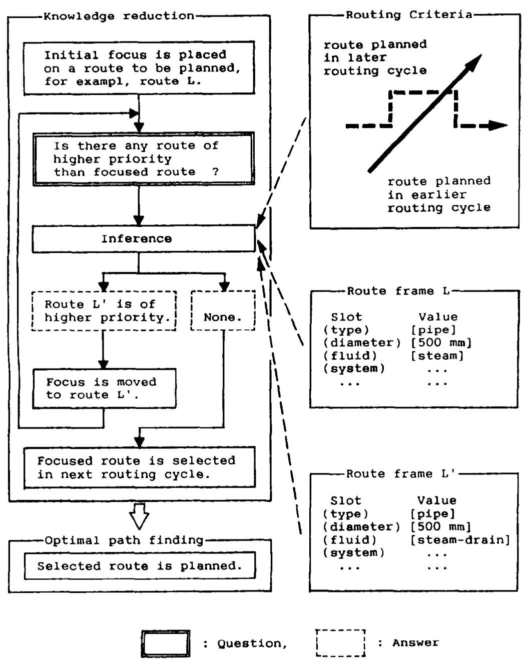

Knowledge reduction
Initial focus is placed on a route to be planned, for exampl, route L.
Is there any route of higher priority than focused route ?
Inference
Route L' is of higher priority.
None.
Focus is moved to route L'.
Focused route is selected in next routing cycle.
Optimal path finding
Selected route is planned.
Routing Criteria
route planned in later routing cycle
route planned in earlier routing cycle
Route frame L
Slot Value
(type) [pipe]
(diameter) [500 mm]
(fluid) [steam]
(system) ...
Route frame L'
Slot Value
(type) [pipe]
(diameter) [500 mm]
(fluid) [steam-drain]
(system) ...
: Question,
: Answer

Fig. 7 Description of example knowledge-based system.

oped inference program. The left half of this figure shows the flow of the pipe route planning which is composed of two steps; knowledge reduction and path finding. Since pipe routes are planned sequentially on the priority basis, the priority assignment problem is solved through a knowledge-based approach in successive routing cycles. At the top of this flow, an arbitrary route *L* is chosen as a basis for priority comparison. Then a question is asked as to whether there is any route of higher priority than the currently focused-on route. This question is answered by the inference program through backward chaining. If the answer shows that another route *L'* is higher in routing priority than the focused-on route *L*, the focus is moved from route *L* to route *L'* and another question is asked. If no higher priority route exists, the focused-on route is selected for planning in the next routing cycle. The planning specifications of selected route are also determined in this knowledge reduction step, and the selected route is planned in the path finding step.

The right half of Fig. 7 illustrates knowledge utilized by the inference program for routing priority determination. Routing criteria are if-then rules to describe routing priority for various kinds of routes. Route frames are structured Lisp objects to represent the characteristics of individual routes. A few typical slots and their values are shown in the figure. Major slot values are supplied from a plant design database through a pre-processing procedure.

rule #1 If the type of route $x1 is pipe,
the type of route $x2 is pipe,
the diameter of $x1 is $d1,
the diameter of $x2 is $d2, and
$d1 is greater than $d2,
then route $x1 is the higher rank than route $x2
for diameter condition.

rule #2 If route $x1 belongs to system $s1,
system $s1 is for facility $f1,
route $x2 belongs to system $s2,
system $s2 is for facility $f2, and $f1 is equal to $f2,
$f1 is equal to $f2,
then route $x1 is of equal rank to route $x2
for construction approval condition.

rule #3 If the type of route $x1 is pipe,
the type of route $x2 is pipe,
route $x1 has start point $a, and
route $x2 has branching point $a,
then route $x2 is a parent pipe of route $x1.

rule #4 If the fluid in route $x1 is steam, and
the fluid in route $x2 is steam-drain,
then route $x2 is of higher rank than route $x1
for fluid condition.

rule #5 If the start point of route $x1 is $a, and
x, y, z coordinates of $a are given,
then the start point of route $x1 is fixed.

rule #6 If route $x1 is not planned,
route $x2 is not planned, and
route $x1 is of higher rank than route $x2
for branching point condition,
then route $x1 is of higher priority than route $x2.

rule #7 If route $x1 is not planned,
route $x2 is not planned,
route $x1 is of equal rank to route $x2,
for branching point condition, and
route $x1 is of higher rank than route $x2
for diameter condition,
then route $x1 is of higher priority than route $x2.

Fig. 8 Examples of rules for routing order.

4.2 Results of Case Study

The case study problem involves selecting a pipe route to be planned with highest priority among 43 pipe routes on the basis of 56 rules and about 500 facts in the knowledge base. Typical examples of rules are shown in Fig. 8 in slightly abstract expressions. They are used to describe relative priorities between routes of different types and terms for the characteristics of routes. Symbol names which begin with $ indicate unifiable variables. Their rules and facts can be automatically translated from the symbolic data form to the functional form.

The reference inference program KRIT-PC is adopted to maintain the knowledge base in which rules and facts are written in symbolic data form. This conventional inference program takes an average 42 CPU seconds on an 8 mips computer to solve priority assignment problems. The inference program with proposed double compilation takes 0.8 CPU second to do the same inference task. The developed inference program can solve the problem more than 50 times efficiently than the reference inference program using the logic programming style. These results shows that the knowledge compilation method is useful in the reduction of computational efforts and, therefore in the mitigation of the imbalance in efforts between symbolic and numerical computations in the sample knowledge-based system.

5. Concluding Remarks

A practical knowledge compilation method was developed on the basis of the conversion of knowledge from rules and facts in symbolic data form to those in functional form through a semiautomatic translation process. The characteristics of the proposed method are summarized as follows.

(a) The inference process was accelerated through compilation at the declarative/procedural level of knowledge representation and at the Lisp/machine code level of computer language.

(b) The semi-automatic translation of rules and facts in symbolic data form to those in functional form.

The proposed knowledge compilation method was applied to a knowledge-based system for automatic pipe route planning in plant layout design. The route priority assignment problem was solved by backward chaining with 56 rules and about 500 facts by both the conventional inference program and the developed inference program. The rules and facts could be automatically translated from the symbolic data form to the functional form. The lattar inference program could solve the problem more than 50 times efficiently than the reference inference program in the logic programming style.

These results suggested that the proposed knowledge compilation method should be a useful tool, achieved by coordinating requirements of inference efficiency in system utilization and knowledge transparency in system construction and maintenance.

Acknowledgements

The authors would like to express their thanks to Dr. H. Motoda of Advanced Research Laboratory, Hitachi, Ltd. for many helpful discussions of this work.

References

1. BARR, A. and FEIGENBAUM, E. A. (eds.). The Handbook of Artificial Intelligence, *William Kaufmann, Los Altos, Calif.* (1981).
2. MITSUTA, T. *et al.* A Knowledge-Based Approach to Routing Problems in Industrial Plant Design, *Proc. of the 6th Int. Workshop on Expert Systems & Their Applications, Avignon, France* (1986), 237-256.
3. YAMADA, N. *et al.* A Diagnosis Method of Dynamic System using the Knowledge on System Description, *Proc. of 4th IJCAI, Karlsruhe, West Germany*, (1983), 225-229.
4. KAWANOBE, K. Current Status and Future Plans of the Fifth Generation Computer Systems Project, *Proc. of Int. Conf. on Fifth Generation Computer Systems, Tokyo, Japan* (1984), 3-17.
5. HAYS-ROTH, R. *et al.* (eds.) Building Expert Systems, *Addison-Wesley, Reading, Mass.* (1983).
6. FORGY, C. L. Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, *Artificial Intelligence*, **19**, 1, (1982), 17-37.
7. WINOGRAD, T. Frame Representations and the Declarative/Procedural Controversy, D. G. Bobrow and A. Collins (eds.): Representation and Understanding: Studies in Cognitive Science, *Academic Press, New York* (1975).
8. REDDY, U. S. Transformation of Logic Programs into Functional Programs, *IEEE Proc. of Int Symposium on Logic Programming*, (1984), 187-196.
9. KARY, D. D. and JUELL, P. L. TRC: An Expert System Compiler, *ACM SIGPLAN Notice*, **5**, 5 (1986), 64-68.

(Received June 26, 1987)