PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1992,
edited by ICOT. © ICOT, 1992

Automatic Generation of a Domain Specific Inference Program for Building a Knowledge Processing System

Takayasu Kasahara*, Naoyuki Yamada*, Yasuhiro Kobayashi*,
Katsuyuki Yoshino**, Kikuo Yoshimura**

*Energy Research Laboratory, Hitachi, Ltd., 1168 Moriyama-cho, Hitachi-shi
Ibaraki-ken, Japan 316 Tel. (0294) 53-3111

**Software Development Center, Hitachi, Ltd., 549-6, Shinano-cho, Totsuka-ku
Yokohama-shi Japan 244 Tel. (045) 821-4111

Abstract

We have proposed and developed an expert system tool ASPROGEN(Automatic Search Program Generator) having a built-in the automatic generation function of a domain specific inference program. This function was based on search-based program specification and an abstract data type of search. ASPROGEN has interfaces for domain knowledge using an object-oriented approach and constraints which represent control knowledge. It is described by using domain knowledge and it can cover a detailed problem solving strategy

We applied ASPROGEN to produce three kinds of scheduling systems. These generated systems have equivalent performance in comparison with knowledge processing systems implemented by the conventional tool. Further, a two-thirds reduction of the program step numbers required as programmers' input was realized.

1. Introduction

Current expert system tools based on production rule and/or frame representation provide an environment to generate expert systems through formalizing and describing problems by production rules. They are powerful tools, and many practical expert systems have been produced by using them. Industrial field applications of expert system tools have sometimes met problems, the most important one being that tools based on the production system only prepare a rule based language, not a problem solving strategy. So, mapping the problem solving strategy to the production rules is difficult for users who are not knowledge engineers.

Domain shells[1], tools based on generic task method[2], half weak method[3], and SOAR[4] have been developed to overcome this difficulty. Domain shells are expert system tools which are restricted to the specified problem regions such as diagnosis, scheduling, and design. They have spread sheet type user interfaces and problem-specific inference programs. But, actual industrial problems include particular conditions, constraints, or problem solving knowledge, and domain shells do not have enough flexibilities to cover all of them. This leads to a conflict between tool flexibility and easy use. In general, the tool becomes more specific to some regions, so it becomes easy to use it, but loses flexibility.

The generic task method and half weak method also have this conflict. The generic task method classifies problem solving methods into several types which are called generic tasks, and prepares generic task tools to provide them. Tool users select an appropriate generic task and supply domain knowledge to develop the knowledge processing system. The half weak method regards problem solving as a search and provides pre-defined search modules. Tool users select an appropriate search module and add domain knowledge to the module. However, these methods, based on classification, do not necessarily give directions for systematic preparation of building blocks of knowledge processing systems. So, tool users must reformulate the problem definition according to the prepared building blocks.

SOAR has more flexibility for defining the problem solving strategy. It can generate a search program by defining several search control rules. But, lack of functions to relate the search program and domain knowledge restricts the applicability of SOAR to toy problems.

Then, we developed ASPROGEN(Automatic Search Program Generator). ASPROGEN is an expert system tool having a built-in automatic generation function of a domain specific inference program was built. To specify the problem to be solved, it has interfaces for describing the problem solving strategy as a search strategy, domain knowledge in an object-oriented way, and the detailed problem solving strategy as constraints among the attribute values of the domain objects.

2. Overview of ASPROGEN

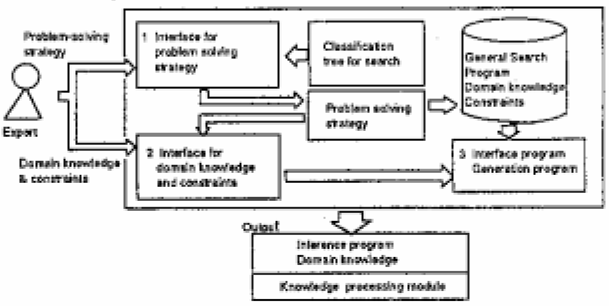
2.1 Building expert systems based on search

ASPROGEN has no embedded inference mechanism. Instead, as shown in Fig.1, its parts include the search program and search program generating mechanism which produces inference programs according to user specifications of the search program, domain knowledge, and detailed constraints.

The reason why we use search as the inference program specification is that it covers almost every inference mechanism required for expert systems, and it is simple. But, a search is not easy to describe nor is it easy to prepare controls tightly directed to a particular problem by the search strategy.

To describe detailed control strategies, ASPROGEN includes an interface for domain knowledge. Using the domain knowledge, the detailed controls or problem solving strategy can be described as constraints between attribute values of domain knowledge. The detailed control programs are complicated in the case of a scheduling system or CAD systems, and it is important to support their generation. In general, domain specific inference programs which have functional operators have complicated constraints. ASPROGEN combines these constraints to global search strategies, and generates domain specific inference programs.

ASPROGEN users develop expert systems by following this procedure:



Problem-solving strategy
Expert
Domain knowledge & constraints
1 Interface for problem solving strategy
Classification tree for search
General Search Program
Domain knowledge
Constraints
Problem solving strategy
2 Interface for domain knowledge and constraints
3 Interface program Generation program
Output
Inference program
Domain knowledge
Knowledge processing module

Fig. 1 Overview of ASPROGEN

(1)Users specify a problem solving strategy from the viewpoint of a search strategy.

(2)Users input the search strategy by selecting the classification items of the search classification tree which the tool prepares. This step is executed with the help of the tool interface.

(3)Users input domain knowledge and constraints with the help of the tool interface.

(4)ASPROGEN generates a domain specific inference program and data structures for domain knowledge.

Although (1) is an interesting problem, we limit the present discussion to (2)-(4).

2.2 Specification of problem solving strategy

To specify the problem solving strategy as a search, we define a classification tree for the search strategy and a template of the search program.

Figure 2 shows the classification tree. It comes from analyzing search trees used in various kinds of problem solving. A search tree consists of nodes and operators. We retrieve the classification items from the characteristics of the nodes and operators. The first classification item comes from the characteristics of the operators. There are two operator types. One is a functional operator which creates new nodes from parent nodes and adds them to the search tree. In the scheduling search program, a functional operator is used. The other type is a link operator. The link operator is used in the diagnosis search program which selects suitable diagnosis nodes for the observed state.

The second classification item comes from the characteristics of the nodes. They are evaluation functions to select nodes in the search procedures, pruning functions, establishment conditions, and so on. The evaluation functions define a global search strategy, for example one which prefers the deepest nodes of the search tree corresponds to the depth first search. The characteristics of the search nodes are described by specification values of the nodes in the search tree which are depth, breadth, parent relations, sibling relations, and node attributes values. Their values are retrieved from the structure of the search tree, and we can prepare these specification values or functions to calculate them. On the other hand, node attribute values cannot be retrieved from the structure of the search tree, and it is difficult to cover all attribute values of the nodes to specify the problem solving strategy.

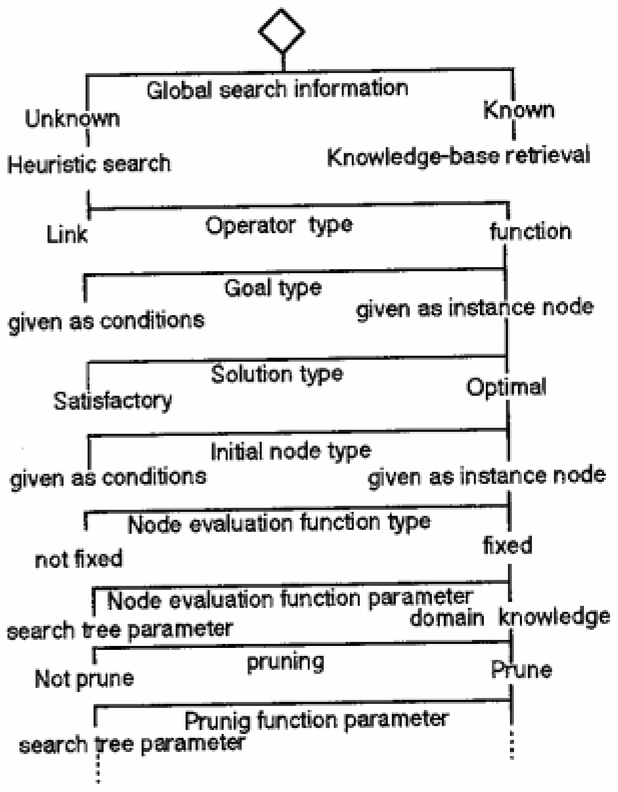


Fig. 2 Search classification tree

To mitigate this difficulty, we rank the attribute values from the viewpoint of their relation to the search tree operators. The attributes which search for the operator directly are called first-order attributes. For example, in the scheduling system starting time and ending time of each job are first order, and the resource constraints are not, if search operators are functions which adjust job scheduling. To describe programs in detail, not only first-order attributes but also multi-order attributes or variables are required. The first-order attributes and the multi-order attributes are domain knowledge. We do not embed detailed domain knowledge in ASPROGEN, instead an interface is prepared to describe the domain knowledge and constraints of attributes of domain knowledge and global search strategy. By combining the global search strategy, described as a search strategy, and domain knowledge, ASPROGEN covers not only toy problems, but also applications for industrial uses.

2.3 *Representation of domain knowledge and constraints*

ASPROGEN has an interface for describing the domain knowledge. Domain knowledge is described by objects and attributes, attribute value ranges, and attribute constraints. There are two types of objects. One is a class objects which defines attributes, and relations between other objects. The other type is an instance object which has instantiated attribute values.

Figure 3 shows a representation scheme of domain knowledge for ASPROGEN. Nodes of the search tree are also objects. Node objects are related to other objects. The relations among objects are of three types.

(1) Class-instance relations: Instance objects have the same attributes as class objects, and the values of the attributes are inherited from the class objects.

(2) Attribute-value relations: The value region of the attributes can be described by he class objects. Thus, the attribute value region is a set of instance objects of the class objects.

(3) Attribute-object relations: The attributes of the objects can be described by the class objects. Thus, the attributes of the nodes are instance objects of the class objects, and attribute values are those of the instance objects.

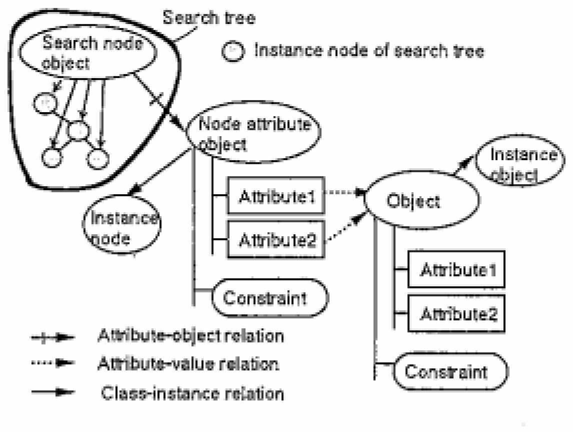


Fig. 3 Scheme of knowledge reprsentation in ASPROGEN

On the basis of these definitions of domain knowledge, ASPROGEN users describe constraints. ASPROGEN prepares a simplified language which can describe constraints by using object names and attributes.

2.4 *Generation of problem-directed inference program*

The inference program generated by ASPROGEN consists of two parts, the search program which corresponds to the global problem solving strategy, and constraint satisfaction programs which correspond to the domain knowledge. Figure 4 shows an outline of the inference program. The control program is embedded into ASPROGEN, and the global search program and constraint satisfaction programs are generated according to user input. If the inference program is completed, it behaves as follows. Using the global search strategy, the inference program activates an operator and generates or selects new node. Then, the constraint satisfaction programs activate and adjust the attribute values of the objects for every constraint. According to the result of the constraint satisfactions, the operator is activated again. This process continues until termination conditions are satisfied. The generating process of the inference program consists of three steps.

(1) Generate the search program which represents a global search strategy

ASPROGEN has a general search program which is independent of domain and includes six search sub-functions as shown in Fig. 5. When completed, it becomes the global search program of Fig. 4. Constraint satisfaction programs are activated in the sub-function of 'Apply operator'. The difference between each search strategy is reflected in the difference of the six element functions.

ASPROGEN prepares two reference tables and abstract data types for search[5],[6]. Parent function parent(c,k) which returns the parent of the node k of search tree c, and Left_most_child(c,k) which returns a child node which was first generated or selected are examples of abstract data types for search. Here, an abstract data type of search makes up the functions for the search program. The first reference table is a table intended for generation of search element functions.

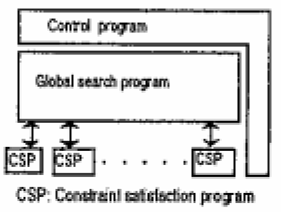
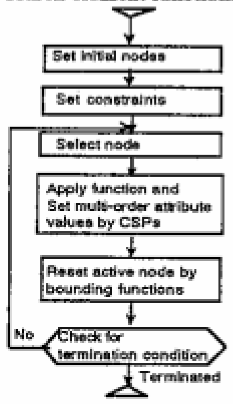


Fig. 4 Outline of the inference program



Set initial nodes
Set constraints
Select node
Apply function and Set multi-order attribute values by CSPs
Reset active node by bounding functions
No
Check for termination condition
Terminated

Fig.5 General search program

Search element functions are program parts of sub-functions and Fig.6 shows some. They consist of the abstract data type of search and domain-dependent search functions. Examples functions are node evaluation function for domain dependent corresponding search element functions to the userspecified problem solving strategy.

Figure 7 shows the generating process of the problem-directed inference program. Referring to the problem solving strategies using the reference table, the system decides the search element function. This is done by domain dependent search control functions such as evaluation function for node.

Then, the same as in the definition process, sub-functions are defined by abstract data types of the search tree.

Functions concerning search tree configuration

(1) parent(c,k)	:returns parent of c in search tree k.
(2) Leftmose_child(c,k)	:returns eldest son of c in search tree k.
(3) Right_sibling(c,k)	:returns next younger brother in search tree k.
(4) Label(c,k)	:returns label of c in search tree k.
(5) Root(k)	:returns root node of k.
(6) Clear(k)	:makes search tree k null set.
(7) Deep(c,k)	:returns depth of c in search tree k.
(8) Height(c,k)	:returns height of c in search tree k.
(9) Leaf(c,k)	:if c has no children returns yes; otherwise, return no.

Functions concerning search tree operations(search element functions)

(10) Evaluate(c,k)	:evaluate c, and returns evaluation value.
(11) Change_e(n_t,S,k)	:change evaluation function of node type n_t to S.
(12) Search state(c,k)	:if c is an open node returns Current; if c is a removed node returns Finished; otherwise returns Yet.
(13) Jumping(c,k)	:returns the node, when c is established.
(14) Back_tracking(c,k)	:returns the node, when c is not established.
(15) Initial(c,k)	:if node c is initialized state returns yes, otherwise returns no.
(16) Kill(c,k)	:removes node c from the active node.
(17) Active_c(k)	:returns active node of search treeof k.
(18) Goal(c,k)	:if c is a goal node returns yes.
(19) Cond_node(n_t,c,k)	:if node c satisfies establish condition of node type of n_t, returns yes, otherwise returns no.
(20) Cond_node_type(n_t,c_t,c,k)	:if node c satisfies establish conditions of node type of n_t, returns yes, othewise returns no.
(21) Establish(c,k)	:if c is established returns yes, otherwise returns no.

Fig. 6 Search element functions

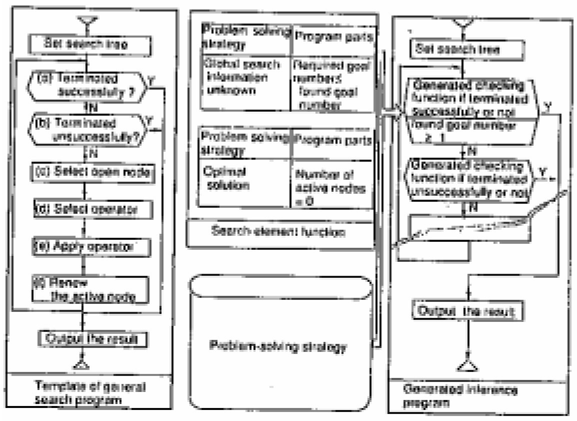

Set search tree
(a) Terminated successfully ?
Y
N
(b) Terminated unsuccessfully?
Y
N
(c) Select open node
(d) Select operator
(e) Apply operator
(f) Renew the active node
Output the result
Template of general search program
Problem solving strategy
Program parts
Global search information unknown
Required goal numbers found goal number
Problem solving strategy
Program parts
Optimal solution
Number of active nodes = 0
Search element function
Problem-solving strategy
Set search tree
Generated checking function if terminated successfully or not found goal number ≥ 1
Y
N
Generated checking function if terminated unsuccessfully or not
Y
N
Output the result
Generated inference program

Fig. 7 Generating function of program-directed inference program

Figure 8 shows an example of the element function generating process, which exemplifies the node evaluation function. According to the user-input problem solving strategy that the depth of the tree has a high evaluation value, the tool selects the depth function from abstract data types and completes the node evaluation function.

Reference table

classification item	selected value	statement
solution type	satisfactory	-
	optimum	depth (c, k)

```
Evaluate(c, k)
   {
     return(
      user_define_func +
      depth(c, k)
            )
                          }
```

Fig. 8 Generation Example for search element functions

Figure 9 shows an example of a sub function generating process, which exemplifies function (a) in Fig.5 which is named SUCCES_END(c,k) here. Since the optimal solution is requested in the problem-solving strategy, the tool generates the checking successful termination function which terminates the inference program only if an optimal solution is found.

Reference table

classification item	selected value	statement
solution type	satisfactory	-
	optimum	Active_c(c, k) == () &&
number of goal	GOAL_NUM	GOAL_NUM

```
SUCCES_END(c, k)
   {
     if (Active_c(c, k) == () &&
         GOAL_c(c, k) >= GOAL_NUM)
         return(TRUE);
   else return(FALSE);}
```

Fig. 9 Generation Example for search sub-functions

(2) Generate the constraint satisfaction programs, according to the user specifications in simplified language

The object names and attribute names of the objects which the tool users input are registered in the ASPROGEN as key words for simplified language constraints description. We call the language, SCRL (Simple Constraint Representation terminal symbols. The SCRL compiler accepts only the following style sentence.

[value clause] [comparing key words] [value clause]

A value clause consists of object name, attribute name and object relation key words.

Table 1 lists key words and their meaning in SCRL. There are set operation key words, comparison key words, and object relation key words such as 'of'. Figure 10 shows an example of a constraint described by SCRL in which the number of persons required for each time span is less than the available personnel number.

```
Constraint name: personnel

  sum((time_span of job),  person )
      >
  time_span of   avairable_person
```

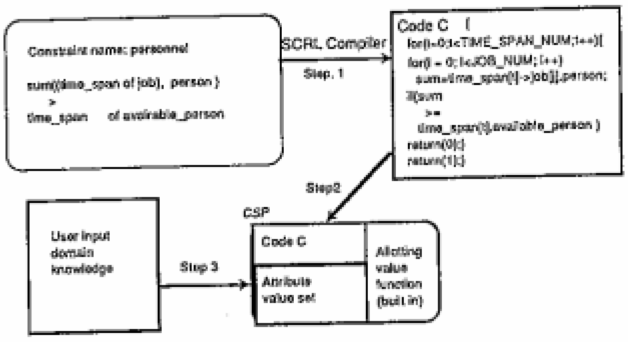
Fig. 10 Example of constraint

Table 1 Example of key words of the SCRP

Set operation keywords
- A include B: $A \supseteq B$
- A have e: $e \in B$
- SUM(A,B): sum up attribute value of B of all instance object of A

Comparing keywords
- $x > y$
- $x = y$

object relation key words
- A of y: value of attribute y of object A.


Constraint name: personnel
sum((time_span of job), person)
>
time_span of avairable_person
SCRL Compiler
Step. 1
Step2
CSP
Code C
Attribute value set
Allotting value function (built in)
User input domain knowledge
Step 3

```
Code C   {
 for(i=0;i<TIME_SPAN_NUM;i++){
 for(j = 0; j<JOB_NUM; j++)
   sum=time_span[i]->job[j].person;
 if(sum
    >=
   time_span[i].available_person )
 return(0);}
 return(1);}
```

Fig. 11 Generating process of constraint satisfaction program

ASPROGEN generates constraint satisfaction programs from the kernel of the constraint satisfaction programs. This kernel of the constraint satisfaction programs comes from the relation of attributes and their value range. The allotting mechanism of the attribute values is built to ASPROGEN. The mechanism selects values from the value range. If constraints are not satisfied, other values are selected. ASPROGEN generates each constraint satisfaction program by setting the object attribute values and their range.

Figure 11 shows an example of the constraint satisfaction program generating process. The constraint is like in Fig. 10, a personnel constraint. First, the sentence is pursed into the C language by the SCRL compiler (code C in Fig. 11). And attribute value range and code C are set to the allotting function which ASPROGEN prepares, and the constraint satisfaction program is completed.

(3) Synthesize all constraint satisfaction programs and the search program

Finally, ASPROGEN synthesizes all the constraint satisfaction programs and the search programs, and generates the domain specific inference program. The key point of the synthesis is to ensure consistency of the attribute values of the objects which the tool users define. To make the argument clear, we define the identity of the search node and scope of the attribute values.

Identity of the search node

The identity of the node is defined by equality of the value set of the first-order attributes (cf. Section 2.2). Search tree operators operate them directly. So, it is possible that the inference programs generate different results, though the problem solving strategies are the same.

Scope of the attribute values

We define scope of the attribute values in the search tree node. The attribute value of the objects should have a consistency in the tree node, and the change of the attribute values in the process of the constraint satisfaction must propagate to other constraints.

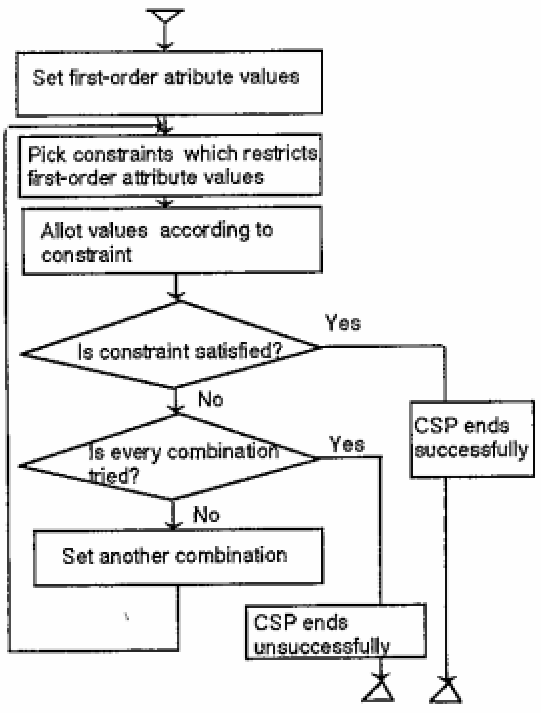

Set first-order atribute values
Pick constraints which restricts first-order attribute values
Allot values according to constraint
Is constraint satisfied?
Yes
No
CSP ends successfully
Is every combination tried?
Yes
No
Set another combination
CSP ends unsuccessfully

Fig. 12 Simplified procedure for constraint satisfaction

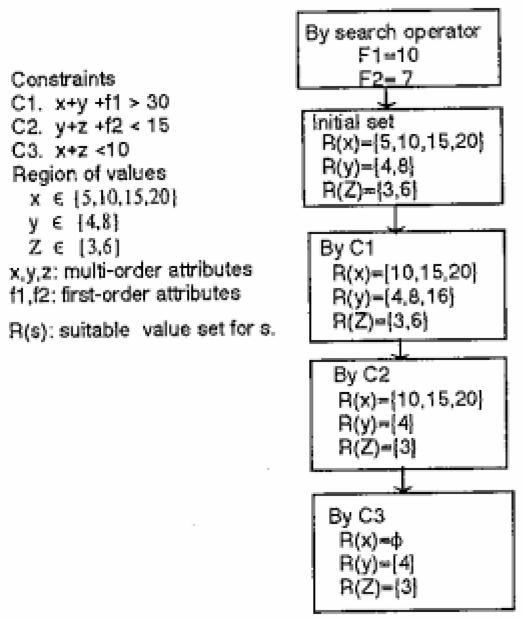


Fig. 13 Example of filterinq process

Figure 12 shows a simplified mechanism to assure consistencies of the attribute values. At first, using the search tree operator, first-order attribute values are instantiated. In the next step, attributes which are constrained by the first-order attributes are instantiated by the allotting mechanism of the attributes. This process continues to survey all constraints. If the set of attribute values is found, then the first-order attribute set is suitable, and if not so, the node is unsuitable. But, this simple algorithm has a fatal defect, i.e. ineffectiveness of the allotting process. If global consistency among the constraints does not exist, the algorithm searches for every combination of the attributes until no solution is found.

To avoid this ineffectiveness ASPROGEN deals with attributes as a set. In the first stage, using the search tree operator, first-order attribute values are instantiated. Then the available value set of multi-order attributes are filtered by the constraints. Figure 13 shows a simple example of the filtering process. At first the region of the attributes value set is a candidate for solution. Filtering by the constraints, inconsistent values are retrieved from the candidate set. The process continues until no retrieval value exists/or no suitable value exists for some attributes.

4. Example and Result

Using ASPROGEN, we built three kinds of scheduling systems. They were a maintenance scheduling system (Problem A), construction scheduling system (Problem B), and jobshop scheduling system (Problem C).

Problem A is a scheduling system for maintenance scheduling of a nuclear power plant[7]. The generated program produces a schedule under constraints of maintenance personnel limitations and interferences between tasks. Problem B is a plant construction scheduling program. The generated program produces a schedule under previous relations between tasks and personnel limitations. Problem C is a jobshop scheduling system. The generated program produces a schedule under constraints of resource limitations and appointed date of delivery.

Table 2 Test problems

Problem	Characteristics of solution	Evaluation function	Constraints	Variety of resources
Maintenance scheduling	optimal	working time	· task interference	2
Construction scheduling	satisfactory		· task execution order	1
Jobshop sche	satisfactory			3

The problems are shown in Table 2. Table 3 summaries the problem solving strategy for each scheduling problem. These problems differ regarding solution type and resource numbers.

Figure 14 shows the domain model of each problem. They are the basis of ASPROGEN input. The framework of these problems is the same. This means that global search strategies are the same. First-order attribute values are the starting and ending times of each jobs. Preference of the node is total scheduling time. There are interference constraints that some jobs cannot be executed simultaneously. Domain knowledge differs. For example, problem A and problem B have personnel limitations, and problem C has machine constraints.

Table 3 Specification of the test problem-- task specific knowledge

Items	Definition of problem solving method		
	maintenance scheduling	construction scheduling	jobshop scheduling
Number of goal	1	all	1
Initial number	1	1	1
Global search information	none	none	none
Type of operator	function of adjusting schedule		
Type of initial state	state of representing work schedule		
Type of goal state	conditions that satisfies all constraints		
Solution type	optimal	satisfactory	satisfactory
Establish conditions about tree configuration	none	none	none
Evaluation function	fixed	fixed	fixed

Figure 15 shows program step numbers which programmers input. Comparing the inference programs implemented by using a conventional tool[8], equivalent performance is realized with two-thirds reduction in number of program steps required as programmer input. Of course, the reduction rate depends on the applications, for example, a diagnosis system has more domain knowledge, and the reduction rate may be smaller than for a scheduling system. But, overall some reduction of programmers load will result by the tool.

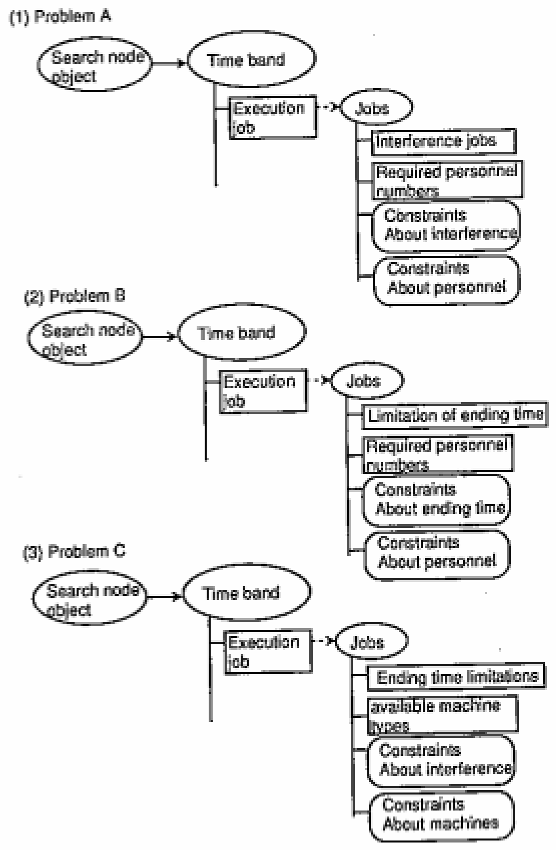

(1) Problem A
Search node object
Time band
Execution job
Jobs
Interference jobs
Required personnel numbers
Constraints About interference
Constraints About personnel
(2) Problem B
Search node object
Time band
Execution job
Jobs
Limitation of ending time
Required personnel numbers
Constraints About ending time
Constraints About personnel
(3) Problem C
Search node object
Time band
Execution job
Jobs
Ending time limitations
available machine types
Constraints About interference
Constraints About machines

Fig.14 Domain model of the problems

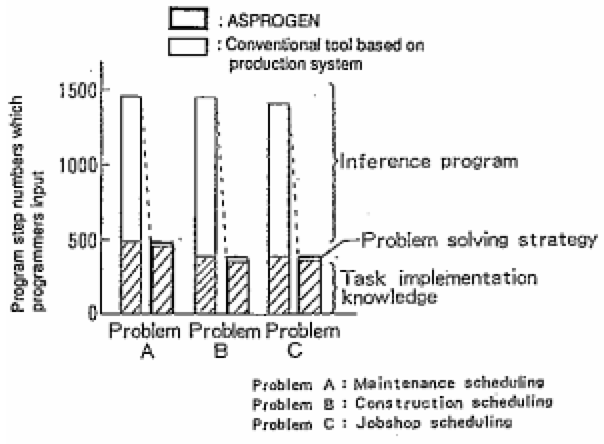


Fig.15 Program step numbers which programmers input

5. *Conclusions*

We have proposed and developed an expert system tool ASPROGEN(Automatic Search Program Generator) in which the automatic generation function of a domain specific inference program was built in. This function was based on search-based program specification and an abstract data type of search. ASPROGEN has interfaces for domain knowledge using an object-oriented approach and constraints which represent control knowledge. It is described by using domain knowledge and it can cover a detailed problem solving strategy

We applied ASPROGEN to produce three kinds of scheduling systems. These generated systems have equivalent performance in comparison with knowledge processing systems implemented by the conventional tool, and two-thirds reduction of the program step numbers required as programmer input was realized by ASPROGEN.

We have applied ASPROGEN only to scheduling systems, we are now going to check its applicability to CAD systems and diagnosis systems.

References

[1] K. Okuda et al.: Model Based Process Monitoring and Diagnosis, Proc. of IEEE Pacific Rim International Conference on Artificial Intelligence'90,pp.134-139, Nagoya, Japan (1990).

[2] B. Chandrasekaran: Towards Functional Architecture for Intelligence Base on Generic Information Processing Tasks, Invited Talk of IJCAI-87(1987).

[3] J. McDermott: Using Problem-Solving Methods to Impose Structure on Knowledge, Proc. of IEEE International Workshop on Artificial Intelligence for Industrial Applications, pp.7-11,Hitachi, Japan(1988).

[4] J.Laird, et al.: Universal Subgoaling and Chunking, Klur Academic Publishers(1987).

[5] E. W. Dijakstra et al. : Structured Programming, Academic Press, London(1979).

[6] A.V. Aho, et al.: Data Structure and Algorithms, Addison-Wesley Publishing Company, Inc., Reading Mass.(1983).

[7] T.Kasahara, et al.: Maintenance Work Scheduling Aid for Nuclear Power Plants, Proc. of IEEE International Workshop on Artificial Intelligence for Industrial Applications, pp.161-166 Hitachi, Japan(1988).

[8] S. Tano, et al.: Eureka-II A Programming Tool for Knowledge-Based Real Time Control Systems, International Workshop on Artificial Intelligence for Industrial Applications, pp.370-378, Hitachi, Japan(1988).