PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1988,
edited by ICOT. © ICOT, 1988

EXPERIMENTAL KNOWLEDGE PROCESSING SYSTEM

Yuichi Fujii, Hirokazu Taki and
other researchers of the Fifth Research Laboratory

ICOT Research Center, Institute for New Generation Computer Technology

1-4-28, Mita, Minato-ku, Tokyo 108, Japan

ABSTRACT

This paper describes the research activities of the fifth research laboratory at ICOT. In order to verify ICOT developments such as the PSI and PIM, we are developing next generation expert tool technologies for real application systems on them. We selected the following technologies as basic elements of the new generation expert tools: hypothetical reasoning, knowledge acquisition, constraint problem solving, object modeling, qualitative reasoning, and distributed cooperative problem solving. This paper discusses these technologies and some experimental expert systems.

1 INTRODUCTION

Experimental knowledge processing systems are an objective of research and development started in the first year of the intermediate stage of the FGCS project. The primary motives are to probe and verify hierarchical interfaces between application systems and ICOT development such as the PIM and PIMOS. To develop knowledge system building technologies, we have been researching expert systems as application systems. During the first half of the intermediate stage, we studied and surveyed next-generation tools and knowledge acquisition support and developed a. prototype tool, called PROTON. During the second half of the intermediate stage, we are striving to develop element technologies based on the surveys. We also organized academic and industrial experts into working group and subgroups to join our surveys and discussions. To verify element technologies, we developed experimental expert systems for a number of applications, for example, VLSI logic design and machinery design.

2 NEXT-GENERATION TOOLS

Conventional tools for building an expert system have a prominent feature: rapid prototyping. Although such tools are effective for some applications, they are not suitable for building large-scale application systems because a building methodology has not been established. An ideal expert system ought to provide a vocabulary matching the scope of its tasks. Conventional tools, based mainly on rule of thumb, provide only an inference engine common to knowledge representations such as rules and frames, and thus require uniformity of tasks. Under this constraint of the tools, users must state their problems. This may make the building of complicated large-scale application systems difficult. To help build knowledge systems, next-generation tools ought to be organized into problem solving frameworks matching an application domain. In other words, we think that a next-generation expert system will be realized by a set of generic tasks [Chandrasekaran 86] or a set of building blocks. From the perspective of problem solving frameworks, the research and development trend of expert systems is shifting from analytical to synthetic problems. Analytical problems infer the characteristics of a whole system from a given system structure and subsystem characteristics. Diagnostic and control problems are typical examples of this type of problem. Synthetic problems showever involve the determination of the system structure and subsystem features which would result in a set of system characteristics. Examples of synthetic problems are design and planning problems [Kobayashi 86]. Synthetic problems are basically combinatorial problems. The number of solutions in a synthetic problem may be infinite. That is, the problem may result in combinatorial explosion. In order to avoid combinational explosion, next-generation tools are expected to incorporate technologies for intelligent inference control.
Taking these trends into account, we are now researching the following five technological elements.
(1) Tool architecture for design tasks
(Constraint- based problem solving and object modeling)
(2) Hypothetical reasoning
(3) Distributed cooperative problem solving
(4) Utilization of deep knowledge and qualitative reasoning
(5) Knowledge acquisition support system

3 TOOL ARCHITECTURE FOR DESIGN TASKS

As already explained, existing expert systems are broadly classified into systems for analytical problems and systems for synthetic problems. Analytical problems are, like diagnostic problems, regarded as problems of selecting hypotheses in a limited solution space, because a set of hypothetical solutions and a set of rules for selecting hypotheses can be predetermined. Synthetic problems, however, need efficient problem solving, since solution spaces are so large that fabricating candidate solutions as hypotheses beforehand is difficult. Design problems are typical examples of synthetic problems. The development of a design expert system requires a large amount of knowledge that depends on a design object. Representation of the design knowledge and a problem solving mechanism are important for research on a design expert system. Our objectives are to clarify the architecture of design expert systems and to develop tools for building them.

3.1 Required Functions

Different groups of designers use different kinds of standard such as design methods, parts, and component units. Thus, building tools, enabling designers to build and maintain expert systems by themselves, are required. This section overviews the design knowledge representation and the problem solving mechanism in the tools needed to satisfy this requirment.

3.1.1 Knowledge Representation

To realize a design expert system building tool, knowledge representation requires two facilities: one is that knowledge must be represented suitably for the tool, and the other is that designers must be able to represent them easily. Design knowledge is broadly classified into knowledge about design objects themselves and knowledge about problem solving. Knowledge about design objects consists of the structures, shapes, and attributes of the design objects. A set of items of knowledge about a design object is called an object model. Knowledge about problem solving, however, is composed of methods to analyze object models, to evaluate and modify solutions, and plans to design the object and search from candidate solutions. According to the above classification, a design process can be regarded as a design requirement satisfaction process [Tomiyama 85, Ohsuga 85]; operations such as selection, modification and refinement with knowledge about problem solving are repeatedly applied to an object model. Furthermore, to enable designers to build an expert system by themselves, an environment is required where a design expert system can be built only by declaratively representing an object model and knowledge about problem solving. To realize the environment, we propose a building tool that generates a design plan from separate inputs of an object model and knowledge about problem solving, and that provides an interface between design knowledge and the problem solver. We used a constraint analyzer, similer to knowledge compiler [Araya 87] and constraint compiler [Feldman 88], to obtain these tool facilities.

3.1.2 Problem Solver

If a design plan is given explicitly, a design problem can be solved according to it. There are often cases where a design plan cannot given explicitly, but only constraints can be given. An effective way of solving these cases is to employ constraint problem solving, regarding a design process as a constraint satisfaction process. In addition to this, the whole of a design process can be captured from the single concept of a constraint satisfaction process; an object model represents constraints on the structures of the design objects, and design requirements and knowledge about problem solving also represent constraints. These constraints are given priorities and changed dynamically according to the designer's intention and preference, and to trade off between performance, due dates and cost. Therefore, a constraint solver suitable for a design problem is required.

3.2 Object Model

3.2.1 Object Model in Design Problems

Design objects in design systems are represented in the form of model descriptions. A design object model represents information and knowledge about design objects, such as their attributes, shapes, and structures. During a design process, a model that satisfies requirements is constructed; it represents a solution.

Models used in conventional design systems consist of data structures that are merely static. They need to be interpreted and manipulated in terms of design tasks or procedures. Only knowledge about design methods is important. Knowledge about design objects is embedded in model manipulation procedures or design methods. In conventional design systems, it is difficult to make effective use of the knowledge about design objects. Also, high performance design and the establishment of a general methodology by which to build design expert systems will be hindered because knowledge about design object and knowledge about problem solving are not distinguished between.

Thus, to solve design problems effectively, it is important to represent the knowledge about design objects as object models and to put those models to practical use in the design process.

3.2.2 Use of the Object Model

A frame system has been used to represent structures and attributes of objects in knowledge systems. Recently, an object oriented paradigm whose concept is similar to the frame system has been generally used and also applied to design problems. Although conventional object oriented languages are suitable for representing structures, attributes and behavior, they do not provide facilities for representing or using constraints on design objects. Therefore, introducing constraints to an object oriented paradigm provides efficient formalism for knowledge representation in terms of declarative description. In representation of a design object, however, functions are required that can describe and use not only constraints on numerical attributes (instance variables), but also constraints on the structures. We are examining two ways of using design object models. One is to generate a design plan by analyzing and compiling knowledge about the design object and about problem solving [Nagai 88a]. It is suitable for parametric design. The second way is to provide a system for supporting the design process interpreting knowledge described on object models. This system makes it possible to construct not only models that satisfy the constraints, but also support their effective construction. This second way is suitable for a problem in which the structure of the design object is not given or is not fixed. In such a case, the problem must be solved by trial and error or by interaction with users. This system is briefly explained in the next section.

3.2.3 Design Object Representation System

Currently, a knowledge representation system for design object modeling, FREEDOM [Yokoyama 88], is being developed. To support design tasks, FREEDOM provides the facilities that keep the status of the model for constraint satisfaction by interpreting constraints that are described in the object model and are dynamically added during the design process. Knowledge representation provided in the FREEDOM system, based on the object oriented paradigm, makes it possible to describe constraints about attribute values and structures. The attributes of the design object model are represented numerically or symbolically, and their values can be obtained by solving constraints derived from them. In conventional object oriented systems, the relation between a class and an instance is a static one, whereas in FREEDOM, the search for a class that satisfies design requirements is realized using a constraint satisfaction mechanism. Thus, when a structure or an attribute of an instance is modified, if constraint satisfaction cannot be executed in the class to which it belongs, the class may be changed automatically to another class to satisfy the constraints. In this way, it is possible to search for a class that satisfies design requirements not by describing the search procedure explicitly, but by using constraints about the design object. As described above, FREEDOM provides facilities for supporting design processes by using constraints described in the object model, and helps to build advanced design expert systems.

3.3 Design Plan Generation Using a Constraint Analyzer

This section first describes representation of design knowledge about problem solving. Second, design plan generation using a constraint analyzer that enables desiners themselves to build design expert systems is described.

3.3.1 Knowledge about Problem Solving

Knowledge about problem solving consists of methods to analyze object models, to evaluate and modify solutions, and plans to design the object and search from candidate solutions. The characteristics of design knowledge about problem solving are various representation types: there is knowledge, such as design formulas, where solving procedures are represented explicitly, and knowledge, such as that expressed by inequalities, where solving procedures are not represented explicitly. In

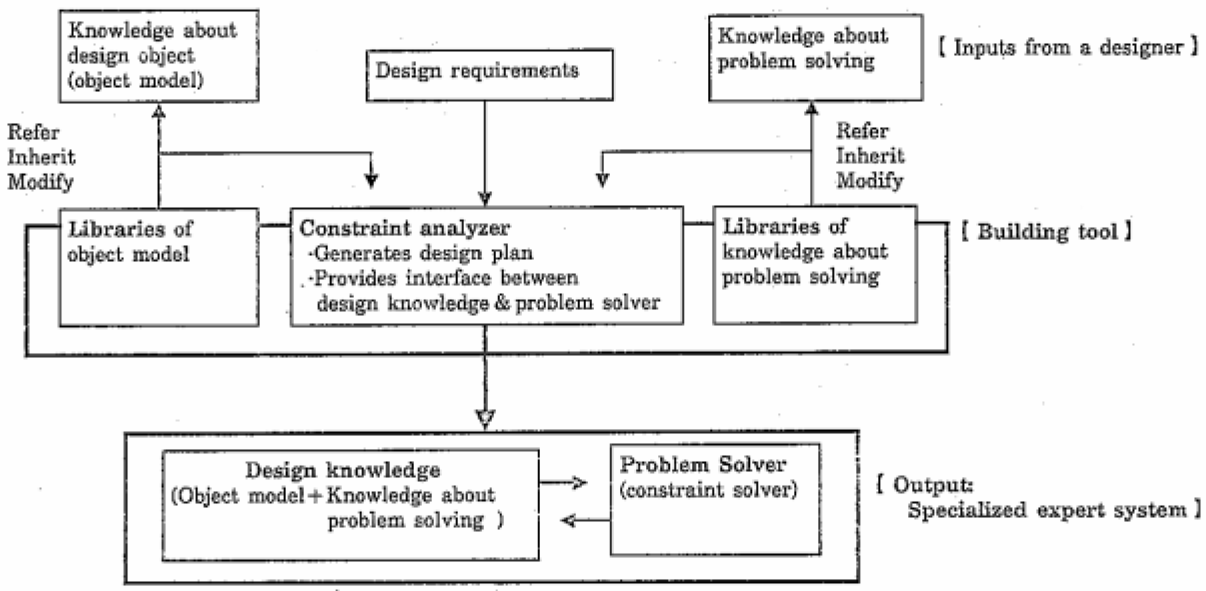

Knowledge about design object (object model)
Design requirements
Knowledge about problem solving
[Inputs from a designer]
Refer Inherit Modify
Refer Inherit Modify
Libraries of object model
Constraint analyzer
·Generates design plan
·Provides interface between design knowledge & problem solver
Libraries of knowledge about problem solving
[Building tool]
Design knowledge (Object model + Knowledge about problem solving)
Problem Solver (constraint solver)
[Output: Specialized expert system]

Fig. 3.1 Architecture for an expert system building tool

addition, knowledge that is independent of a design object and heuristics that is closely dependent on a certain design object are mixed. For example, design formulas and searching from catalogues in design knowledge about problem solving, and basic parts and function units in object models are independent of a design object. Therefore, with the aim of enabling designers to represent this knowledge easily, we employed an approach where those independent kinds of knowledge are prepared as system libraries; we prepared sets of design formulas and catalogues in knowledge about problem solving, and sets of basic parts and function units in object models. These types of knowledge must be expressed in a form that designers can easily refer to and modify. Consequently, designers' heuristics can be expressed explicitly, by referring to or by inheriting and modifying libraries.

3.3.2 Design Plan Generation Using a Constraint Analyzer

A constraint analyzer can handle various types of knowledge and can specialize knowledge by combining knowledge independent of a certain design object and designers' heuristics which depend on a certain design object. Since the constraint analyzer can generate a design plan by analyzing dependencies among constraints, design knowledge can be also represent declaratively. Inputs to the tool are design requirements, object models, and knowledge about problem solving. They are given by specifying system libraries, or by modifying libraries with referring or inheriting libraries. Reference to results of previous design and designers' heuristics about searching from alternatives are also represented as knowledge about problem solving. From these inputs, the tool analyzes dependencies among constraints and parameters, generates a design plan, and provides an interface between the design knowledge and the constraint solver. The output from the tool is a specialized expert system including designers' heuristics. Therefore, a flexible environment in which build an expert system can be built by designers themselves is obtained by dividing design knowledge into object models and knowledge about problem solving, and by employing design plan generation using a constraint analyzer.

3.4 Constraints in Design Problems

The structural information derived from the object model is constraints expressed explicitly. In addition, design knowledge such as methods to analyze object models and design requirements such as cost performance are also regarded as constraints, from the single view of the constraint concept. However, not many of the existing tools that support the construction of expert systems provide an environment that makes it easy to express the constraint concept explicitly; the person constructing the system must use the language depending on the tool to realize mechanisms for applying constraint representations which depend on the design object. This section discusses the characteristics of constraints in design problems [Nagai 88b].

(1) Static and dynamic constraints
Many existing constraint solvers consider constraints as static entities. In design problems, however, not all constraints are given in the initial stages of a design process; many are added or deleted during the design process. Furthermore, there are suggestive constraints as described below; constraints are dynamically changed in design problems.

(2) Obligatory and suggestive constraints
Not all the constraints are selected and executed on an equal basis in design problems. In other words, priorities are assigned to constraints, and the priorities are based on design requirements and designers' intentions. All obligatory constraints must be satisfied, and these are generally given explicitly. Suggestive constraints, however, are used as guides in choosing the optimum branch at a node in the search tree, and they are given lower priorities than obligatory constraints. Thus, if an obligatory constraint cannot be satisfied, suggestive constraints may be changed so that the obligatory constraints are satisfied.

(3) Local and global constraints
Many design problems are divided into subproblems when an attempt is made to solve the problems. Thus, it is necessary to distinguish whether the applicable scope of a constraint closes locally within a subproblem or is globally related to other subproblems. In addition, interactions among local constraints within a subproblem and interactions between local and global constraints must be considered.

(4) Propagation of values and interval bounds
Some constraints in design problems are represented by inequalities. Therefore, not only do constraints propagate values, they also propagate over interval bounds in which variables that can take certain values must be considered.

When considering practical design problems, one constraint may belong to multiple types of these characteristics.

3.5 Architecture of the Building Tool

As stated above, we divide design knowledge into object models and knowledge about problem solving. This enables us to maintain knowledge and to modify knowledge flexibly. Viewing knowledge and requirements as constraints, constraint based problem solving is employed. To help designers to build an expert system suitable for a design problem, we propose a building tool that regards inputs of design knowledge as constraints, generates design plans by analyzing their dependencies, and provides an interface between design knowledge and a constraint solver. We used a constraint analyzer to obtain facilities for this building tool. The expert system which is the output of the tool can efficiently obtain solutions that satisfy the design requirements, according to the design plan generated by the tool. Fig. 3.1 shows the architecture of the building tool. An expert system building tool, MECHANICOT [Terasaki 88], is being developed now. MECHANICOT is a tool for a mechanical parametric design. It analyzes dependencies between structures of a design object and parameters, produces a design plan, and builds a specialized design expert system.

4 HYPOTHETICAL REASONING

4.1 Problem Solving with Hypothetical Reasoning [Inoue 88c]

Hypothetical reasoning [Inoue, ed. 88] is a type of inference which is desirable to have when dealing with alternatives among knowledge, or incomplete knowledge (knowledge that may not always be true) in problem solving. It assumes that unclear or insufficient knowledge is true (establishes hypotheses), and attempts to have the inference proceed based on the hypotheses. Because the hypotheses and formulas derived from the knowledge and hypotheses are not guaranteed to be true, it is necessary to check consistency through constraints or other means. If a contradiction occurs in the reasoning process, we must remove the original hypotheses and select other ones instead. For this reason, hypothetical reasoning can be interpreted as a kind of *non-monotonic reasoning*, and belief revision technology is required. Hypothetical reasoning is in fact inference as practiced by humans, and is one key to implementing advanced inference mechanisms such as commonsense reasoning and learning. Conventional research into hypothetical reasoning, however, has concentrated on establishing the basic inferential mechanisms, and there has been little work done *from the viewpoint of application* in problem solving. This section discusses a prototype system called APRICOT/0 of the APRICOT project [Inoue 88c] as a basic software tool for next-generation knowledge-based systems.

A variety of frameworks for handling hypotheses and incomplete knowledge has been proposed [Doyle 79, de Kleer 86a, Poole 88, Reiter 80], but from the viewpoint of application, they have been faced with major problems in that (1) there has been no integrated handling of the generation, selection, and verification of hypotheses, and (2) architecture has not taken problem solving into account. As the basic standpoint for the construction of APRICOT, we stressed the following two points :

(1) By using domain-dependent knowledge, especially deep knowledge (such as structure and function knowledge), commonsense knowledge (knowledge of physical laws, etc.) and constraints, APRICOT will automatically generate and enumerate hypotheses. It will be more intelligent than the conventional approach stressing heuristic rules.

(2) Positioning the inferential control mechanism between the hypothetical reasoning mechanism and a domain-dependent problem solver will enhance efficiency [Inoue 88a].

There are two points to be considered in the use of hypotheses. The first is a dependency of what knowledge is established on the basis of what hypotheses. *Truth maintenance systems* (TMSs) manage dynamically contradiction-free characteristics in a database (working memory) including the hypotheses, and have been proposed by [Doyle 79] and in the *assumption-based TMS (ATMS)* [de Kleer 86a]. The second is called *abductive reasoning*, where hypotheses that do not contradict the database explaining the observed events are selected. A hypothetical reasoning system of this type has been proposed in [Poole 88]. Both have in common management of *consistency*, however, and can be unified model-thoretically [Inoue 88b]. In other words, the former maintains the contradiction-free style of the database from the input hypotheses, and the latter determines goal hypotheses from the input observations.

APRICOT provides a basic framework for using hypothetical reasoning in problem solving, but as the problem solver is dependent on the problem domain, only one of the above approaches may be stressed, depending on the problem domain. For example, the inferential strategies for *diagnosis* and *constraint satisfaction* would be as outlined below.

4.1.1 Problem Solver for Diagnosis

If components are assumed to be working correctly, and predictions from those assumptions are inconsistent with behavioral observations, the conflict set (the set of disjunctions of negated literals of assumptions) are determined, and possible combinations of faulty components can be calculated theoretically by converting the conjunctive normal form (conflict set) into the disjunctive normal form. This means that it is sufficient to find a consistent set of assumptions that explain the observations and goals through backward reasoning.

4.1.2 Problem Solver for Constraint Satisfaction

Assumptions are regarded as assignments of values to some variables, and so when results derived from them through forward reasoning are inconsistent with the specifications, the combinations of assumptions that support the observations or their negations are determined. In planning, a set of parameter values satisfying various kinds of constraints is collected as a context. In design, multiple design models are maintained, structured with hierarchical contexts so that the upper layers are the design model assumptions and the lower layers the parameter assumptions.

4.2 Hypothetical Reasoning System APRICOT/0

The APRICOT/0 system for hypothetical reasoning consists of the ATMS [de Kleer 86a], which maintains consistency based on combinations of assumptions (called *environments*), and a rule-based problem solver. APRICOT/0 is implemented in ESP [Iijima & Inoue 88, Fujiwara & Inoue 88].

APRICOT/0 treats the ATMS and a rule-based problem solver called the assumption-based

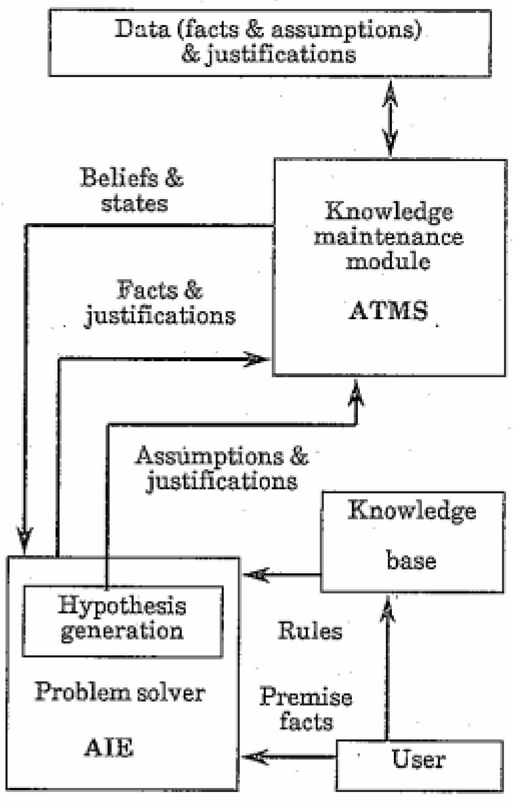


Fig 4.1 Configuration of APRICOT/0

inference engine (AIE) as independent modules. As shown in Fig. 4.1, AIE provides the ATMS with *data* (*facts* and *assumptions*) and their *justifications*, and the ATMS efficiently determines all *contexts* (sets of all data which hold in each consistent environment). AIE proceeds with inferential processing while checking whether the ATMS data holds in some contexts.

In multiple worlds of the ATMS, each time a new fact is inserted it causes other events such as the addition of a justification and the occurrence of a contradiction. To express these activities accurately, APRICOT/0 expresses components such as assumptions, justifications and contradictions as ESP objects. Attribute information for each is contained in the object slot, and the truth maintenance algorithm is implemented through inter-object message passing.

AIE is the actual problem solving mechanism, operating within the hypothetical reasoning system to link the user-input premise facts and rules from the knowledge base with the belief states from the ATMS. To avoid firing unnecessary rules and generate only the minimum essential number of justifications, an inference control mechanism similar to the Rete algorithm is used. The ESP unification function is used in matching assumptions and facts to rule conditions. A *rule* consists of the *condition* part, which is matched with obtained facts and assumptions taking all contexts into account, and the *action* part, which provides additional facts and assumptions, and their justifications to the ATMS. Each condition of the condition part is an ESP predicate, variable or atom, whose valuation is true if the ATMS node corresponding to the object fact or assumption is believed (called *IN*), that is, the ATMS node holds in some environments, or, a method call or ESP built-in, whose valuation is true if its ESP execution, such as a numerical calculation, succeeds. If all conditions of the condition part are true, a justification of the form : <condition part>⇒<action part> is passed to the ATMS. Rules without an action part indicate that they generate contradictions if they are executed, and they will be executed with maximum priority within the same environment in action part queue scheduling. Several AIE rule examples are given below.

Example 4.1

1) `rule91:: temperature (X,Y), {Y>=25} -> cooler_ON (X).`
   %If the temperature Y of room X is 25° or higher, turn on cooler X.

2) `birdfly:: bird(A) -> assume(fly(A)).`
   %If A is a bird, assume it can fly.

3) `contradiction:: not(X), X -> [].`
   %If both affirmative and negative of X exist at the same time, it is contradictory.

4.3 Knowledge Compiling on APRICOT/0

As discussed in 4.1, the basic concept of APRICOT is to utilize various kinds of knowledge, including incomplete knowledge, linked together functionally, to solve problems effectively. Model knowledge expressing principles (called deep knowledge) is combined with constraints and heuristics to generate powerful knowledge that is directly helpful in the domain task; this is called *knowledge compiling*. While the problem solver of APRICOT/0, namely, AIE, is a rule engine and handles AIE rules only, APRICOT/0 can simulate the function implementation of knowledge compiling, only if all knowledge such as default knowledge, logical inference rules and constraints are converted into AIE rules, and are passed through a sophisticated scheduler.

4.3.1 Dynamic Hypothesis Generation and Default Reasoning

When humans solve problems, they perform inference as establishing a succession of assumptions depending on their circumstances. In this process, all possible hypotheses are not listed beforehand; rather, hypotheses can be generated or deleted as required. To implement this process, in APRICOT/0, a function is provided that *dynamically* introduces hypotheses.

Use of this function allows *default rules* to be represented that produce results as long as no evidence contradicts them. For example, a normal default "a(x) : Mb(x)/b(x)" [Reiter 80] can be expressed as an AIE rule "`a(x) -> assume(b(x))`". Internally, an assumption $\Gamma_{b(\lambda)}$ (λ is a ground term) is introduced, and b(λ) is expressed as an assumed node supported by $\Gamma_{b(\lambda)}$. The justification "$a(\lambda) \wedge \Gamma_{b(\lambda)} \Rightarrow b(\lambda)$" [de Kleer 86b] is passed to the ATMS. This allows inference to proceed using $\Gamma_{b(\lambda)}$ as a default assumption.

Example 4.2

When Tom Sawyer met Huckleberry Finn, who watched movies on a weekday, he wondered if Huck was a delinquent. He then remembered that Huck's friend (himself) was not that type of person, and thought that therefore Huck could not be, and denied the belief that "that day was a school day". (See Fig. 4.2)

```
rule1:: not(X), X ->[]. %Logical contradiction.

rule2:: day(weekday) ->
        assume(not(close(school)))).
        %Typically, go to school, school not closed
        on weekdays.

rule3:: not(close(school)), see(X,movie)
        -> go_slow(X,school).
        %School not closed, so people watching
        movies are skipping school.
```

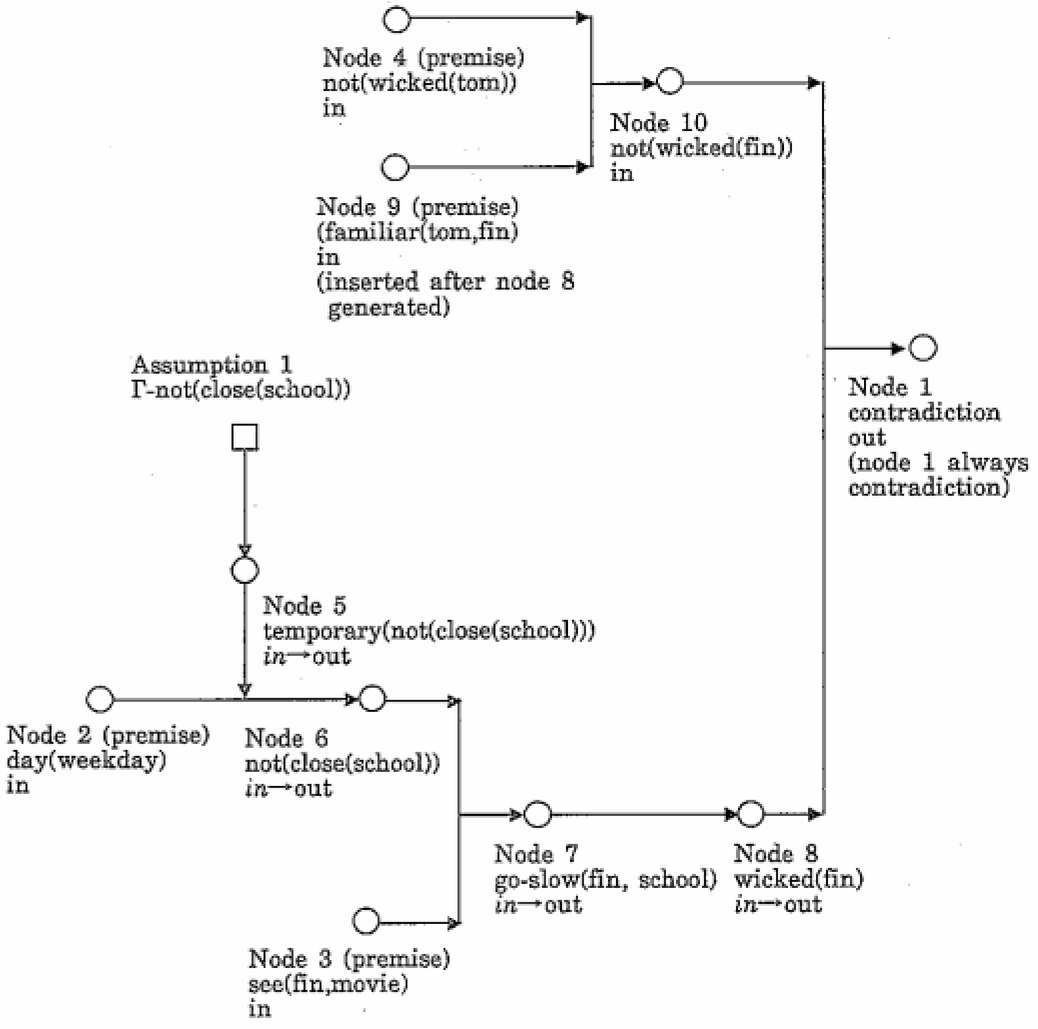


Fig. 4.2 Huckleberry Finn justification network

```
rule4:: go_slow(X,school) -> wicked(X).
        %People skipping school are juvenile
        delinquents.

rule5:: familiar(Man1,Man2),not(
        wicked(Man1))->not(wicked(Man2)).
        %Man not a delinquent, so his friend not a
        delinquent.
```

4.3.2 Inference Rules for Natural Deduction

Production rules, with which inference is executed through Modus Ponens, have been conventionally used in heuristic rule description. With this framework alone, handling more logical structures such as circuits leads to a situation where logical completeness cannot be assured. If logical inference rule descriptions are expressed by the AIE rules, it becomes possible to describe And Elimination, Or Elimination, Modus Tollens, and so on.

4.3.3 Handling Constraints

One of the uses of the ATMS in constraint-based problem solving is *constraint satisfaction*, where solutions satisfy the set of all constraints. This regards assignments of values to variables as assumptions, and determines consistent sets of assumptions that do not violate the constraints to make them solutions. A solver equipped with an assumption generator and a constraint checker can allow the ATMS functions to obtain all solutions. This type of assumption-based problem solving can be applied to combinatorial problems and design problems [Inoue 88].

If some mechanism of *constraint propagation* can be incorporated organically, problem solving becomes even more flexible. The interface between the ATMS and a problem solver handling constraints has been proposed in the form of the *consumer architecture* (CA) [de Kleer 86c] incorporating the data-flow mechanism. In this concept, consumers are unit problem solving steps attached to the corresponding ATMS nodes through an analysis of the set of constraints (called *precompiling*). The consumer generation procedure (precompiling) is as follows. First, consumers are shaped through the data-flow analysis of the set of constraints (input by the user as relational expressions among variables) based on heuristics related to the usage of constraints. Then, the action parts are attached to the ATMS nodes related to each condition part. The consumer execution procedure is as follows. When the ATMS node becomes IN during the inference

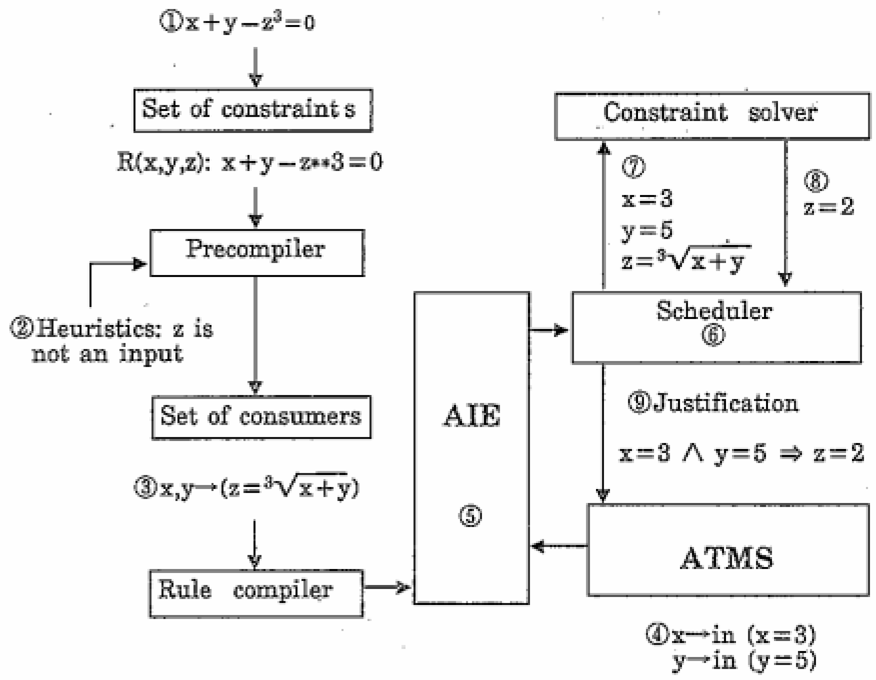


Fig. 4.3 CA by APRICOT/0

process, the attached consumers are passed to the scheduler, and passed to some solver when appropriate, and then the results are passed to the ATMS along with its justification.

The implementation of CA using APRICOT/0 is as follows and as shown in Fig. 4.3. Consumers in [de Kleer 86c] are attached to the ATMS nodes, but here they are converted to AIE rule formats. This enables simultaneous handling of heuristic rules and constraints through the common framework, that is, AIE and a scheduler. In this architecture, if a value of a variable, say "a", is assigned or updated, then the predicate "`equal(a, A)`" is introduced (indicating that "a" has a value, say λ, and is unified with ESP variable "`A`"). In AIE, this predicate is interpreted as IN when variable "a" is bound by λ in some context, and so delayed execution of consumers is possible. This means that constraint propagation is possible as a data-driven evaluator, and CA can be implemented.

Example 4.3 (A process of consumer generation and execution (See Fig. 4.3))

(1) Relationship among x, y, z : "$x+y-z^3=0$" is input.

(2) In the way of using the constraint, it can be seen that "z" is only used as the output.

(3) Data-flow in the variable set is determined, and the constraint is converted to a rule.

(4) In an inference process, x and y become IN.

(5) Rule conditions are satisfied, and the action part is sent to the scheduler.

(6) The consumer is scheduled to the queue according to a certain strategy.

(7) The consumer is picked up from the queue and passed to the solver when appropriate.

(8) The result given by the solver is returned to the CA.

(9) The CA registers the justification "$x=3 \wedge y=5 \Rightarrow z=2$" in the ATMS.

Example 4.4

The following constraints (and heuristics related to their usages) areconverted into AIE rules below, taking constraint analysis into account.

$g= -6*a*a+0.7*b+c$
(g is only used as an output)
$g> 3*a+b*c$
(this is used as a test after calculating the value of g)
$g< a*b*c$
(this is used as a test after calculating the value of g)

```
rule101::equal(a,X), equal(b,Y), equal(c,Z)
            -> {G is -6*X*X+0. 7*Y+Z},
               equal(g,G).
rule102::equal(a,X), equal(b,Y), equal(c,Z),
            equal(g,G), {G<=3*X+Y*Z} -> [].
rule103::equal(a,X), equal(b,Y), equal(c,Z),
            equal(g,G), {G>=X*Y*Z} -> [].
```

4.3.4 Scheduling

The knowledge compiling function handles various types of knowledge (such as constraints, heuristic rules, defaults, and logical inference rules) under the common framework, so it is not enough for its implementation merely to convert that knowledge into the single AIE rule representation; it must schedule action parts of invoked rules. This can be accomplished by adding the following functions to the action part queue.

(1) Sort the queued action part list in ascending order of environment size.

(2) If consumer execution causes a contradiction to be detected and then some action parts to be no longer IN, they are removed from the queue. This prevents unnecessary justification generation and consumer execution.

(3) Add some kind of priority as heuristics. For example, higher priority is given to rules introducing contradictions. Set the priority according to types of knowledge and circumstances where to use it.

(4) Incorporate various kinds of search algorithms [Inoue 88a].

4.4 Conclusions and Future Research

This section discussed the architecture of the APRICOT/0 hypothetical reasoning system, composed of the ATMS maintaining concurrent representation of all contexts and the AIE rule-based problem solver, as well as the techniques used to implement knowledge compiling. Application is currently being considered for design and planning problems such as the design problem of the main spindle head in a lathe [Inoue et al. 88] and the problem of automatic generation of the disassembly sequence of machine tool head stocks (see 8.2), as well as distributed cooperative problems such as a delivery planning problem (see 8.3). Future plans include extension and generalization of the ATMS, and parallel implementation of the ATMS and AIE in GHC.

5 DISTRIBUTED COOPERATIVE PROBLEM SOLVING SYSTEM

A cooperative problem solving system solves problems for which optimum solutions are difficult to obtain. A typical model of such a system is the blackboard model. To improve automation system performance, cooperative problem solving functions may be useful particularly in the field of designing. Instance where they are useful are large-scale objects such as LSI circuits, because (1) combinations of constraints must be considered in phases of a design process (including the verification phase) to solve design problems; and (2) a conventional design process is divided into phases which are executed separately; thus it may not be able to produce the best product or design the required production this field. A typical conventional system is the HEARSAY-II system developed in the early '70s by the speech understanding project at CMU.

This system was designed to obtain solutions from ambiguous incomplete data (including noise) and knowledge. The architecture of the system was very promising [Nii 86]. However, it gave rise to difficulties in representing knowledge about inference control and in processing a large volume of data. For this reason, it has been left unused without finding out its full advantages. However, the recent progress in LSI and network technologies is spotlighting this architecture again. The system has turned out to be able to exhibit the originally expected performance if the architecture is expanded to cover a multiprocessing environment. Theoretical researche has also progressed in problem solving with uncertain incomplete data and knowledge. Distributed cooperative problem solving techniques have been developed, mainly by the members of Distributed AI workshops in the United States. Target environments pursued by the members for system models are versatile; they range from an environment for connectionist models (massively parallel machines) to a conventional computer environment [Davis 80, Fehling 83, Gasser 87, Smith 85]. Nevertheless, the environments lack clear basic concepts that serve as criteria or assessing their features. This fact darkens the outlook of the research. We are now trying to clarify the concepts of distributed cooperative problem solving, enumerating technical objectives, and probing possible methods.

5.1 Definition of distributed cooperative problem solving

Distributed cooperative problem solving is defined typically by Smith as follows [Smith 85]. "Distributed cooperative problem solving is cooperative solving of a problem by a group of decentralized and loosely coupled knowledge sources. Knowledge sources here mean knowledge systems described by some knowledge representations in various processors. They are cooperative because none of them has the necessary information or information processing capability to solve the whole problem. They are said to be decentralized if no global control and no global data storage site exist. They are said to be loosely coupled if they spend more time on computation than on communication".

As in above, distributed cooperative problem solving is irrelevant to a specific knowledge representation form and inference method. It stipulates a coarse system architecture for problem solving. It does not determine what sorts of knowledge sources are decentralized or how they are decentralized. Nor does it clarify how knowledge sources cooperate. A distributed AI system like the connectionist model includes tightly coupled problem solvers which are assigned small tasks. However, they are not regarded as distributed cooperative problem solvers [Decker 87].

5.2 Structure of a problem solver

The system consists of multiple problem solvers. It divides a problem, solves subproblems, and synthesizes the solutions. Various architectures can be considered for this system. We present only the structure common to problem solvers making up the system. Each problem solver consists of the following components (Fig. 5.1):

(1) Communicator: Exchanges processing results with other problem solvers.

(2) Controller: Borders retrieval spaces for tasks and performs focus control to reduce communications. Focus control selects the least costly, most efficient subtask when subtasks are connected by OR logic. When subtasks are connected by AND logic, the controller analyzes parallelism.

(3) Reasoner: Performs inference.

(4) Knowledge base: Contains knowledge of experts. Knowledge is dispersed to problem solvers, and no problem solver has the necessary knowledge to solve the whole problem.

(5) Working memory: Stores processing results of tasks.

5.3 Advantages of distributed cooperative problem solving

A distributed cooperative problem solving system improves, as do existing distribution systems, in performance. It heightens its processing speed and reliability. Routine programs have difficulty in performing knowledge processing subtasks. Therefore, if the subtasks were distributed to existing data processing subsystems, communication overhead would increase and thus would the advantage of distribution be offset. Cooperation functions are necessary in an environment where these subtasks can be efficiently distributed. Introduction of cooperation functions also improves the expandability of the system. When the system is expanded, cooperation between modules eliminates the need to change existing system resources. Another advantage of cooperative problem solving is the capability to obtain appropriate solutions. A feature of problems now under discussion is uncertainty. Uncertainty here means lack of data and lack of guarantee for completeness, correctness, and consistency of processing results supplied to a problem solver from others. Cooperative problem solving may obtain justifiable solutions under this uncertainty.

5.4 Features of a distributed cooperative problem solving system

Distributed cooperative problem solving can be regarded as a framework of inference control over multiple problem solvers to solve a problem cooperatively by using inference functions rather than knowledge representations. The optimum framework may depend on problems. An inference control frame has the following facilities:

(1) Integration mechanisms
Multiple problem solvers in a distributed cooperative problem solving system work in harmony solve a problem. An integration mechanism is thus necessary in the system to integrate the actions of problem solvers.

(2) Communications between problem solvers
Suitable communications facilities are important resources for a distributed cooperative problem solving system. The facilities may take various forms from the perspectives of: communication paradigms, communication contents, and communication protocols.
Communication paradigms refer to the following two communication forms:
(a)Communication through global memory (blackboard model)
(b)Message passing

(a) may be asynchronous communication, and (b) synchronous. Synchronous communication lowers processing speed, whereas asynchronous communication makes it difficult to guarantee data compatibility between problem solvers.

5.5 Technical objectives

Advantages of distributed cooperative problem solving can be divided into the following two groups:

(1) Advantages given by distribution processing, that is, ease of system construction, high execution speed, and high reliability

(2) Advantage given by cooperation processing, that is, generation of appropriate solutions by using limited data, knowledge, and processing time

In a distributed cooperative problem solving system, tasks and intermediate solutions are exchanged through communications. The communication speed is generally slower than computation speeds in problem solvers. Nevertheless, working towards a better solution increases communication frequency and quantity. Therefore, improvement in efficiency of communications is a major technical objective. How to obtain appropriate solutions through cooperative processing is another major technical objective. To achieve these objectives, we are now studying the following techniques:

(1) Efficient communication in a distribution environment
In a distributed problem solving environment, each problem solver assumes self-control over its inference function. In this environment, problem solvers share tasks, processing results, or resources, and thus must be coordinated functionally. A technique for satisfying this requirement is the inference control technique proposed by Durfee and Lesser, called partial global plans [Durfee & Lesser 87]. This technique makes each problem solver create tactics for solving a problem, which arises in the whole network but is viewed from the local standpoint, and exchange the tactics with other problem solvers. Whether one problem solver should employ tactics offered by another depends on evaluation standards implemented by the problem solver. The above technique may be a meta-communication technique. Another technique for efficient communication is the expectation-driven communication, which is performed by anticipating the actions of partner problem solvers. For one problem solver to anticipate the action of another, it must know the intention and action plan of the latter. The situation theory may be used to find out the intention action plan. This theory was first used by SRI to devise a cooperative work plan for multiple robots. Concerning this plan, Georgeff proposed an action theory, and Konolige presented a belief mode[Konolige 85].

(2) Cooperation for obtaining an appropriate solution
A problem solving technique used in an environment where applicable knowledge and input data fall short is inference based on evidence. Human beings copy flexibly with problems by assuming the presence of exceptions in incomplete knowledge about the real world. The knowledge assuming exceptions are called default knowledge. Inference based on evidence uses default knowledge and instances supporting the correctness of default knowledge. When an inference process encounters conflicting assumptions, it selects an appropriate one based on supporting values[Shastri 85]. Through the above studies and analyses, we will propose a framework of distributed cooperative problem solving.

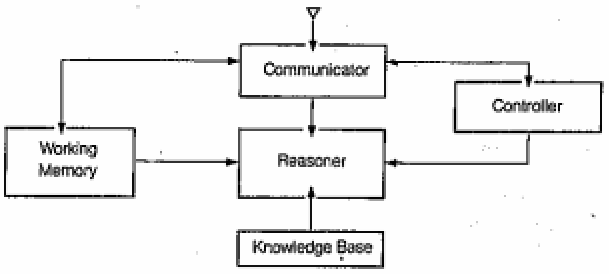


Fig. 5.1 Structure of a problem solver

6 USE OF DEEP KNOWLEDGE AND QUALITATIVE REASONING

6.1 Use of Deep Knowledge and Its Effects

One problem with conventional expert systems involves the complete inability of the system to solve a problem if it does not have inference rules for the problem, Because the basic ability of the conventional expert systems for solving a problem is based on the range of the inference rules in the problem domain. Moreover, as the expert system does not in essence understand the knowledge in problem domain, there are limitations in the intelligent problem solving and the explanation of reasoning processes. As one solution to overcome this problem, reasoning using deep knowledge (deep reasoning) has been proposed. Shallow and deep knowledge are defined as follows.

Shallow knowledge: Knowledge directly related to the tasks performed by an human expert.

Deep knowledge: Basic knowledge close to the universal laws or principles in the problem domain, or knowledge of general validity such as representations of the structure and function of the object.

Use of deep reasoning is anticipated to have the following effects.

(1) Completeness of knowledge :
As each item of deep knowledge is expressed at a more basic level, the range covered by a single piece of knowledge is more broad. Hence, deep knowledge can cope with situations which cannot be predicted in advance on the basis of direct cause-effect relations, such as production rules.

(2) Understanding and explaining causality :
As the knowledge in the problem domain is in a form which is close to physical laws and principles or the structure of the object, the system is far more capable of explaining the results of reasoning processes.

(3) Automatic generation of shallow knowledge :
By compiling and storing reasoning results for various situations in a rule format, deep knowledge can be used in the automatic generation of shallow knowledge. General deep knowledge in the domain can be used in the construction of various expert systems for different applications in the domain (for instance, design knowledge can be used to generate diagnostic rules). That is, deep knowledge acquired can be used efficiently. On the other hand, shallow knowledge generated can be used for high-speed inference.

One means to achieve deep reasoning is qualitative reasoning. Its basic procedure is to express physical quantities and constraints existing among them qualitatively, and to reason about the system behavior. Methods for current qualitative reasoning systems (simulators) may be broadly divided into two categories :

(1) Qualitative modeling type :
In this type of simulator, variables and constraints representing the system dynamics (a set of simultaneous qualitative differential equations) are given and fixed. All of these variables and constraints are used to reason the qualitative behavior of the whole system. An example is QSIM [Kuipers 85].

(2) Qualitative process theory type :
Basic knowledge representations of this type of simulator are objects and, process or physical rules. Process and physical rules contain the constraints to change the states of each object. Process or physical rules which are currently active are identified to construct a set of constraints representing the system. System behavior changing over time is reasoned using the constraints. Not only are qualitative states determined, but an understanding of causality is also sought. QPT [Forbus 84] is an example of such a system.

6.2 Qualitative Reasoning Mechanism

The Fifth Research Laboratory is studying qualitative reasoning mechanisms from the following two approaches.

(a) Research and development of Qupras [Ohki 88], a QPT-type qualitative reasoning system, which aims to deal with physical laws in their original form (without qualitative modeling).

(b) Research for improving the efficiency of the reasoning processes of the two types of qualitative reasoning systems mentioned above.

6.2.1 Outline of Qupras

As stated above, QPT is closer to the reasoning using deep knowledge than the approaches oriented to qualitative simulation. However, the QPT framework is in some respects inadequate to express knowledge at the level of general physical laws as basic knowledge. Hence, Qupras (for the qualitative physical reasoning system), a qualitative reasoning system which overcomes these difficulties is under development. As shown in Fig. 6.1, Qupras consists of a knowledge representation supporting subsystem and a reasoning subsystem.

(1) Knowledge representation in Qupras

Knowledge representation in Qupras involves descriptions of the object, physical rules and initial states. Objects are described in terms of (i) attributes (definition of attributes which describe the object), (ii) parts (definition of parts of the object), (iii) conditions (the object becomes active only when these conditions are satisfied), and (iv) relations (relations among physical quantities which are valid when the object is active). Fig. 6.2 is an example of an object describing a boiler class.

Physical rules are expressed in terms of (i) objects (to which the physical rule can be applied), (ii) conditions (conditions under which the physical rules may be applied), and (iii) relations (between the attributes of objects or other quantities). The physical rules are active only when both (i) and (ii) are active. Fig. 6.3 gives an example of a physical rules describing heat flow.

The initial state defines the state of the target system at the beginning of reasoning (that is, instances of objects and definition of facts).

Conditions and relations of objects and physical rules are expressed as equalities (using addition, subtraction, multiplication and division), inequalities, or terms which describe facts such as positions. Further, Qupras is capable of handling these formulas not only qualitatively, but also quantitatively.

Knowledge about objects and physical rules is given in the form of templates. Prior to the inference process, the knowledge representation supporting subsystem of the Qupras applies template knowledge to instances in the initial state, to generate instances of the object and physical rule, then converts them to an intermediate format which the reasoning subsystem can understand.

(2) Qualitative reasoning in Qupras

The qualitative reasoning subsystem in Qupras has the structure shown in Fig. 6.4. Beginning from the given initial state, the succeeding behavior of the target system is reasoned. The qualitative reasoning is performed by two processes, intra-state analysis (propagation) and limit analysis (prediction), in turn.

In intra-state analysis, the reasoning subsystem searches for active objects or physical rules whose conditions are satisfied, and collects constraints in them to construct the simultaneous differential equations describing the target system at that time. The subsystem propagates known attribute values to undetermined attribute values through constraints.

In limit analysis, the reasoning subsystem predicts a qualitative value at the next time for each attribute changing with time. It is selected from the nearest limit points of the present value searching for the equalities and inequalities of conditions of objects and physical rules.

```
object boiler:Boiler
  parts__of
    container—container;
    heat__source—heat__source;
  relations
    on (Container! Boiler, heat__source! Boiler);
    melting__point @ container! Boiler
    < temperature @ heat—source! Boiler;
  end.
```

Fig. 6.2 Definition of a Boiler

(3) Features of Qupras

(a) Knowledge related to physical laws can be represented in a single formulation. (QPT handles dynamic and static phenomena separately.)

(b) Formulas describing physical laws may be represented without qualitative modeling: further, physical quantities may be handled in either a qualitative or a quantitative manner, as the situation demands.

(c) No statements of the partial ordering between the values of varying physical quantities, or of quantity spaces, are required.

(d) Representation and reasoning about states changing of physical variables is possible. In

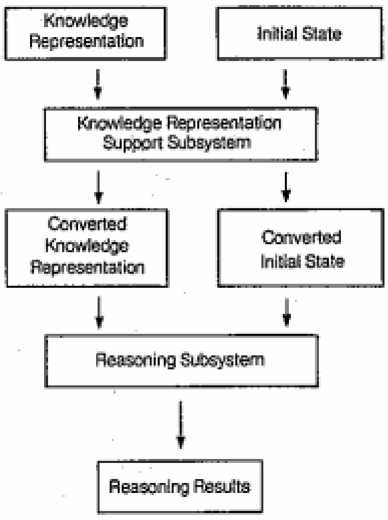


Fig. 6.1 Outiline of Qupras

```
physics heat__flow
  objects
    Heat__source—heat__source;
    Container—container;
  conditions
    on (Container, Heat__source);
    temperature@Heat__source<>temperature@Container;
  relations
    ddt (heat Container) : = :
      temperature@Heat__source—temperature@Container;
  end.
```

Fig. 6.3 Definition of Heat Flow

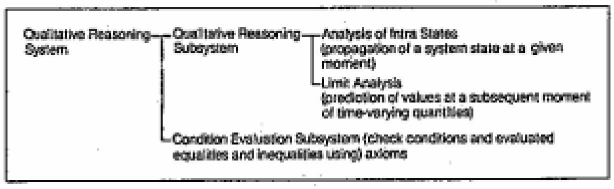

Qualitative Reasoning System
Qualitative Reasoning Subsystem
Analysis of Intra States (propagation of a system state at a given moment)
Limit Analysis (prediction of values at a subsequent moment of time-varying quantities)
Condition Evaluation Subsystem (check conditions and evaluated equalities and inequalities using) axioms

Fig. 6.4 Structure of the Qupras Qualitative Reasoning System

other words, Qupras expresses physical laws in a more primitive form.

(4) Remaining subjects for study

(a) Studies of better primitives for representing such physical laws as the conservation of energy

(b) Improvement of the condition evaluator
We have partially enabled the solution of non-linear simultaneous inequalities by combining the Sup-Inf method for linear simultaneous inequalities [Shostak 77] and the Groebner-base method for non-linear simultaneous equation

[Buchberger 83]. We have also enabled control of the sequence of evaluation of equalities and inequalities (freeze function). However, we have to increase the execution speed.

(c) Constructing a hierarchical structure of object definition, and generalizing definitions of physical rules. At present, an improved version, Qupras ver.2,which adopts these functions and features, is under development.

6.2.2 Improvement of Qualitative Reasoning System

One of the problems with the current qualitative reasoning system is that it cannot predict the behavior of large target systems because of the limitations of computation time and memory capacity. Even the QSIM algorithm, which presently performs reasoning at the highest speed of all the qualitative reasoning systems, can handle only a small continuous system. One method to realize a qualitative reasoning system which can reason the behavior of large target system including multiple physical domains is partition of the target system. We propose two methods for partition based on the heuristics with the structure and properties of the target system. We estimate the effect of the partition methods for computational complexity of qualitative reasoning and discuss the applied conditions of the methods.

(1)Partition method of target system based on heuristics with their structure and properties

(a) Method 1 : Partition method of variables according to the independence of each subsystem.
Assume a target system which consists of many parts that have close interactions between internal components and only weak interactions with external components. Such a target system can be divided into loosely connected subsystems corresponding to the parts. All variables belonging to only one part are assigned to the subsystem as its internal variables. Each external variable (that is. an output from a part and/or an input to the other) is shared by both of these subsystems as a common variable. The behavior of each subsystem can be simulated independently as response to the input state. The simulated results of each subsystem are communicated to others via common variables. They are integrated to constitute the behavior of the whole target system.

(b) Method 2 :Partition method of a system by the field of applicable rules.
Assume another target system, each of whose components is designed and operated according to the rules of a different physical field (for example, electronic circuit, thermodynamics, electrostatics, and quantum mechanics) to satisfy a different function. In this case, the target system can be partitioned into subsystems. Each subsystem consists of parts whose fields of dominant rules are identical. As the range of rules applicable to each object is limited, the number of objects contained in each subsystem is also reduced. Because the rules of different physical fields have little effect, the target system is divided into independent subsystems. Therefore, the behavior of each subsystem can be simulated independently.

(2)Effect of partition methods on the computational efficiency of Qupras

This section estimates the effects of applying partition methods 1 and 2 to the qualitative reasoning system, Qupras. (For more details, refer to [Sakane 88].)

(a) Effect on propagation process

Suppose that an object system is partitioned into W subsystems of the same size by partition methods 1 and 2 . We estimate the computational complexity required for simulating the system behavior using the equally partitioned model. Because the number of instance rules generated is reduced to 1/W, the computational complexity required to find active physical rules is also reduced to 1/W. Thus, the computational complexity in the propagation process is reduced to approximately 1/W times under the following conditions:

(i) application to large target systems without feedback loops consisting of relatively independent subsystems.

(ii) Expression of each physical rule in a general form.

(b) Effect on prediction process

The cost required to predict the next value of each variable changing with time is proportional to the number of instance rules generated. The cost is reduced to 1/W times using the equally partitioned model. However, if there are many variables changing with time, all the combinations of their next values must be checked for consistency. The total computational complexity in the prediction process increases sharply as the number of variables changing with time, Y, increases. Then, the computational complexity is not reduced unless Y decreases. To reduce the number of such variables, some knowledge to control the order of changes among variables is needed.

The partition methods also have advantages in acquiring this kind of knowledge. When the simulator simulates the system behavior as a whole, only the knowledge of the order of changing among variables is available. However, it is very rare in practice that sufficient knowledge is given to specify a variable to be changed first. When the system is partitioned into subsystems by the partition method, the knowledge with the order of variable changes among subsystems is also available. This knowledge is likely to be known, even if the orders of changing among variables are not known.

(3)Remaining subjects for study

(a) Studies of efficient methods of partition of target systems other than the above two methods

(b) Study of a method of mapping the qualitative time of each subsystem to real time

(c) Dealing with discontinuous change which occurs at the beginning and/or end of the interval of qualitative time in each subsystem.

(d) Controlling reasoning: controlling the order of changes among variables and controlling the order of propagation using the dependency among variables.

6.3 Systems Which Apply Deep Knowledge

The range of application of deep reasoning (and qualitative reasoning in particular) may consist of analytic problems and synthetic problems; here we consider a malfunction diagnostic system employing deep knowledge as an example of an analytical problem. The following two types of use are possible.

(1)Generation of diagnostic rules (knowledge compilation)

Qualitative differential equations of the faulty system are constructed for each candidate of malfunction. Qualitative behavior of the faulty systems is acquired, using the model to obtain malfunction symptoms. By connecting the malfunction to the symptoms, the system generates diagnostic rules in the form of "If (symptom) Then (malfunction)". Because the diagnostic rules are generated for all the malfunction candidates, this method is not efficient with regard to the execution time. Therefore, it is suited to off-line generation of diagnostic rules.

(2)Identifying faults based on symptoms

First, the qualitative value of the variable representing the malfunction symptom is propagated to other variables through the constraints. When propagated qualitative values contradict at a certain variable, the variable is considered to be the point where the malfunction causes. In this method, contradiction may be detected at many variables. Therefore, it is necessary to eliminate secondary symptoms by some knowledge and select only the direct cause of malfunction. Because only a set of differential equations must be considered, this method is suited for on-line use with respect to the execution efficiency.

7 KNOWLEDGE ACQUISITION SUPPORT SYSTEMS

A major problem which tends to arise when constructing knowledge-base systems concerns bottlenecks at the knowledge acquisition stage. Knowledge acquisition involves the collection of knowledge from human experts, arrangement and systematization of this information, and construction of a knowledge base for use in a knowledge-base system. At present, this task is performed by knowledge engineers (KEs); that is, knowledge acquisition relies entirely on human efforts. And, since systematic methods of knowledge acquisition have not yet been established, the process of knowledge acquisition is an extremely troublesome one for the knowledge engineer. Our goal is to improve such bottlenecks. Somewhat more concretely, we are responsible for the clarification of the types and structures of knowledge possessed by human experts, and with the establishment of effective methods for the extraction and organization of expert knowledge. In considering knowledge acquisition support systems, we may analyze the work of the knowledge engineer in the following four broad phases.

(1) Problem analysis:
In this phase, the tasks of system to be developed are determined, and the system feasibility and significance of development are analyzed.

(2) Expert model building:
The technical terminology, task procedures (problem-solving strategies) and conceptual structure used by human experts are clarified, and the methods and environments of expert system use are determined.

(3) Expert model instantiation:
Expert knowledge is elicited and organized in the form of the expert model, and the knowledge base is built.

(4) Knowledge-base management:
The knowledge base is modified, added to, or deleted to correct any contradictions, redundancies, deficiencies, or unnecessary (isolated) data.

Knowledge acquisition support systems capable of supporting each of these phases are required. Below, we discuss the basic technology required for such knowledge acquisition support systems, and briefly sketch some systems (CATS and EPSILON/One) which are now being researched.

7.1 Basic Technology for Knowledge Acquisition Support Systems

Knowledge acquisition support systems are hybrid systems, consisting of a number of basic technologies. We list some of these here.

(1) Knowledge acquisition interface (interface with knowledge source)
Interfaces for knowledge acquisition are divided into interpretive and interactive interfaces, according to the manner in which information is exchanged with the knowledge source. In interactive interfaces, knowledge is obtained directly from human experts through that interaction. Conversational representations

employ symbols, numbers, or. words, etc.), tables (spread-sheets, for instance) and other means of expression. Interpretive interfaces, such as those used in protocol analysis and text analysis, on the other hand, involve the direct one-way flow of information from the knowledge source to the knowledge acquisition support system. Here, techniques for interpreting knowledge representations used by the knowledge source (i.e. techniques for natural language understanding) are required.

(2) Interviewing techniques
In interactive knowledge acquisition, only the necessary information must be extracted from the human expert if the acquisition process is to be efficient. Further, knowledge which is to be acquired must be educed through association. Methods for prompting associations include pairwise comparison, personal construct theory (used with CATS, discussed below) and the pre-post method (used with EPSILON/One, below).

(3) Building of conceptual structures (domain models, expert models and knowledge representation)
In general, the knowledge representation supported by expert shells is extremely basic (for instance, rules and frames). Because of this, it is difficult for experts to express their specialized knowledge. Knowledge acquisition support systems can facilitate the extraction of knowledge, by supporting knowledge representation in specialized operations or tasks. Such representation may take the form of domain models (basic conceptions of the objects with which the expert deals, and relations between objects) or of expert models (expressions of the task performed by the expert in terms of basic operations; used with EPSILON/One).

(4) Refinement of task models (expert and domain models)
During the process of acquisition, acquired knowledge contains deficiencies, redundancies and contradictions. A refinement method appropriate to the model structure is thus used to perfect the task model.

(5) Knowledge base evaluation The acquired knowledge (task representation knowledge) is translated to the knowledge representation of an expert shell, and the inference engine of the expert shell is used to evaluate the extent to which the system is capable of the intellectual activity of the expert.

7.2 Classified Task Acquisition Support (CTAS) System

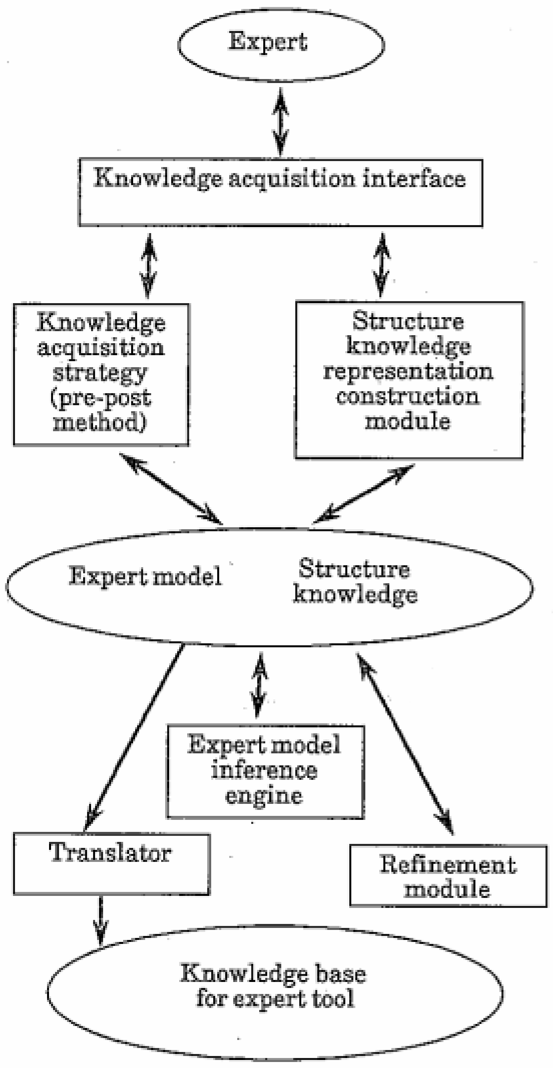
CTAS [Yamazaki et al. 87] is a knowledge acquisition support system which builds an initial knowledge base for classification-type problems. Knowledge-base systems may be broadly classified into a synthetic class and an analytical class; of these, CTAS is applied to the analytical class. In general, problem solving for the analytical class consists of a hierarchical classification task and an ordering task. The hierarchical classification task classifies the elements (items to be classified) on the basis of broad, clear traits. The ordering task, on the other hand, classifies elements that can no longer be classified by clearly distinguishable traits, according to the strength of correlations between traits and elements. CTAS enables the acquisition of knowledge bases in which these two tasks are done using knowledge acquisition methods based on George Kelly's Personal Construct Theory. Using the elements, traits by which elements are classified, and scaled ratings(a scaled rating is the correlation between elements and traits), CTAS generates production rules with a certainty factor. CTAS consultation consists of five stages – elicitation, arrangement, refinement, rule generation, and testing. In the elicitation process, elements for classification, traits and scaled rating are elicited; the arrangement stage involves making graphs and tables using the elicited information so that the acquired knowledge can be perceived visually. The refinement process comprises refinement of elements, traits and scaled ratings, while the rule generation process is concerned with the generation of production rules. In the testing process, the rules thus generated are evaluated. Consultation for each of these processes is explained below. In the elicitation process, support of hierarchical classification task and ordering tasks relies on the elicitation of classification elements, traits, and scaled ratings. In hierarchical classification tasks, traits which can be used to classify elements into hierarchical levels are elicited, and elements are thereby organized into groups; this process may be performed either top-down or bottom-up. In top-down classification, division into hierarchical levels is first performed; this method is effective when experts have already organized elements (into a hierarchy). In bottom-up classification, classification is performed after elicitation of elements; this method is advantageous when elements have not been organized, or have not been organized throughly, by experts. An ordering task involves the elicitation of traits, and scaled ratings between elements and traits for every group. Four types of graphs and a table are prepared in the arrangement process - hierarchical trees, rating grids, implication graphs, and cluster trees. All except hierarchical trees are prepared for each group in a hierarchy. The hierarchical tree indicates the hierarchical relationship between groups of classified elements. A rating grid indicates, in table form, the elements, traits and scaled ratings contained in a group. An implication graph shows the implication relationships between traits in a group. A cluster tree is prepared for each of the elements and traits contained in a group. Each cluster tree indicates the similarity of relationships between items (classification elements and traits). The graphs and table facilitate visual inspection and verification of elicited knowledge by human experts. In refinement processes, several types of refinement methods based on the Personal Construct Theory are used in the refinement of elements, traits and scaled ratings. Additions, deletion, integration and renaming are performed for the different items (elements and traits). Scaled ratings are also corrected. In the rule generation process, three types of production rules - hierarchical rules, conclusion rules and intermediate rules - are generated. Hierarchical rules are used for hierarchical classification, and express the hierarchical relationships between elements. Conclusion and intermediate rules are used for the ordering tasks; and the correlations between traits and elements contained in a group are expressed using the certainty factor. In the testing process, rules thus generated are evaluated through use. In this process, the hierarchical rules are invoked, and groups of elements which are to be assessed are determined. Then the conclusion rules and intermediate rules are first invoked, and result in classified elements which are displayed in order of the certainty factor. The advantage of CTAS lies in its support of knowledge acquisition focusing on the structure of tasks which the knowledge-base systems perform. Also, the graphs and table enable efficient refinement of the acquired knowledge base. By using the CTAS system, a high-quality knowledge base for classification-type problems can be obtained, and the knowledge base can be refined efficiently. Possible applications of CTAS lie in classification tasks in the various areas of diagnostics and planning. CTAS was developed using ESP running on the PSI, and knowledge bases thus acquired are performed using the CTAS inference engine.

7.3 Knowledge Acquisition Support System Based on Expert Model (EPSILON/One)

The EPSILON/One [Tsubaki et al. 88] [Ohsaki et al. 88] is a knowledge acquisition support system which gathers intelligent expert work in small units called operations and builds a knowledge base. We designed an expert model [Taki et al. 87] to represent a knowledge base consisting of these operations. We also developed the pre-post method as a means of acquiring knowledge for the expert mode. Fig. 7.1 shows the configuration of the EPSILON/One. The knowledge acquisition strategy of the pre-post method acquires knowledge through the knowledge acquisition interface and builds an expert model. In addition, as a knowledge representation model of structural information, the structural information knowledge representation construction module acquires structural information knowledge through the knowledge acquisition interface. Then, the refinement module refines the expert model based on structural information knowledge. The expert model and structural information knowledge are converted into a knowledge base for expert shells by the knowledge representation translator. In the following, we discuss the basic concepts employed in expert models, representations of the structure and functions of the expert model, and pre-post method.

7.3.1 Basic concepts of the expert model

We propose an expert model based on two different ideas - a simplified expert task model, and analysis and grouping of diagnostic expert knowledge for production expressions.



Expert
Knowledge acquisition interface
Knowledge acquisition strategy (pre-post method)
Structure knowledge representation construction module
Expert model
Structure knowledge
Expert model inference engine
Translator
Refinement module
Knowledge base for expert tool

Fig. 7.1 EPSILON/One overview

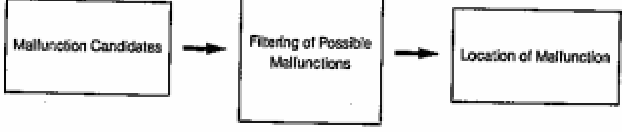


Fig. 7.2 Simplified diagnostic task model

(1) Simplified expert task model

The operations performed by an expert system may be divided into a number of task types, such as diagnostics, design and control. We attempted to represent these task types using simple models. We show this approach because simplified expert tasks allow us to provide images of expert system operations to human experts, to enable them to give expression to their own knowledge. We here present one example of this (Fig. 7.2). On studying simplified expert task models, we found that such models (that is, operations) could be conceived to consist of the object (source element group), processing (evaluator), and processing results (destination element group).

(2) Rule analysis in diagnostic expert systems
Production rules are the general means of representing knowledge in expert systems. The knowledge engineer must possess knowledge representation techniques having a production rule form. We assumed that such techniques appear in rule forms, and discovered the following seven description forms in sets of production rules: selection, classification, sort, combination, translation, input and output forms. The result of combining these two ideas is generic operation.

7.3.2 Structure of expert models

As just explained, expert models consist of types of operations and the relations between operations. These operation relations contain information on the order of execution of operations. Next, as an example of an operation, we consider a classification into types (Fig. 7.4). In the figure, group animals-1 is the source element group, while animals-2, animals-3 and animals-4 are destination element groups. Suppose that the evaluator's task is to divide the elements into three groups according to size. In this example, the elements are dog, cat, wolf, porpoise, rat, whale, elephant, and the size of each animal is its attribute.

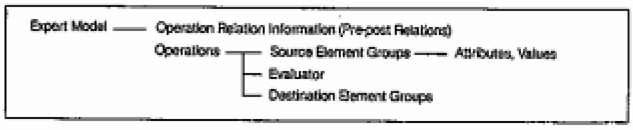

Expert Model
Operation Relation Information (Pre-post Relations)
Operations
Source Element Groups
Attributes, Values
Evaluator
Destination Element Groups

Fig. 7.3 Expert model structure

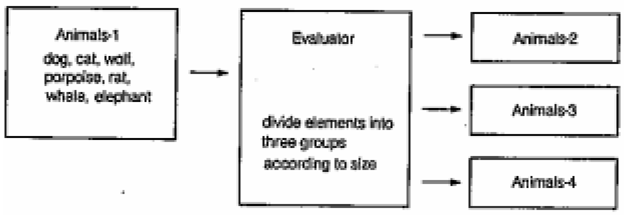

Animals-1
dog, cat, wolf, porpoise, rat, whale, elephant
Evaluator
divide elements into three groups according to size
Animals-2
Animals-3
Animals-4

Fig. 7.4 Example of an operation type (classification)

7.3.3 Method of knowledge acquisition (pre-post method)

The main strategy of the pre-post method stimulated the expert to remember the associative operations before and after (pre- and post-) a given operation. It is relatively easy for the expert to state what operations are necessary before and after a given operation. For instance, when a car will not run, if a human is asked "What should be done before checking the engine?", it is easy to answer "Check if there's gasoline in the tank" or "Check the battery". Further, in the pre-post method, the details of operations are determined, and the operation management structure (meta-script) is configured. The knowledge acquisition process by the pre-post method proceeds as follows.

(1) Collection of operations serving as the starting-point of knowledge acquisition Arbitrary operations are extracted from the expert. (An example is the "engine check" just mentioned.)

(2) Extraction pre- and post- operations
Extraction of operations preceding and following each of the above operations (in the above example, "Check the gasoline" and "Check the battery").

(3) Checking of pre-post relations
Graphic illustration of the operations preceding and following each operation, and checks for differences in these relations.

(4) Determination of operation types
Types are assigned to operations; when assignments cannot be made, operations are further divided into smaller units. Operation types are chosen from among seven types, such as selection and sort.

(5) Operation merging operations in which the same processing is performed are merged.

(6) Evaluator determination
By extracting the necessary information for operation types, the details of the evaluator are decided. (When the type of operation is selection, selection criteria are extracted.)

(7) Determination of source element groups
The elements to be processed by the evaluator are determined. (When the type is selection, objects for selection are extracted.)

(8) Determination of element attributes and values
The attributes and attribute values for each element, to be evaluated by the evaluator, are determined.

(9) Grouping of operations into blocks
Operations which rely on identical reasoning methods are combined.

(10) Determination of reasoning method
The operation reasoning method is determined sequential or parallel, full-solution or partial-solution search.

7.3.4 Refinement method by using different knowledge representation models

Human knowledge consists of a variety of domain knowledge. Accordingly, it is desirable that each piece of knowledge should be acquired in a format suited to its representation. However, a knowledge base of a single knowledge representation must be refined from many points of view. Besides the expert model of the EPSILON/One, there is a knowledge representation model which can hierarchically represent the structural information of the system. It seems useful to refine the expert model by comparing this knowledge representation model to the expert model, so we are presently studying this method.

7.3.5 Future reseach

We have introduced an expert model derived from analysis of production rules in diagnostic expert systems and the simplified expert task model, and acquisition method (pre-post method) for this model. We are developing the EPSILON/One by ESP on the PSI. At present, it is equipped with a sequential inference engine for the expert model. The expert model is a framework which can represent parallel operation knowledge, so we will have to study the possibility of applying it to knowledge acquisition suited to parallel inference. Moreover, we will have to develop parallel inference engines for the expert model.

8 EXPERIMENTAL EXPERT SYSTEMS

8.1 Expert System for VLSI Logic Circuit Design

A cooperative expert system for logic circuit design, called co-LODEX, accepts input of the VLSI behavioral algorithm specifications, datapath structure and constraints to support automatic design of CMOS standard cells. Constraints on gate count (or more precisely, basic cell count) and constraints on delay time can also be input. As indicated in Fig. 8.1.1, co-LODEX performs overall design through cooperation between the agents designing the finite-state machine and control circuit, and those implementing the datapath structures (registers and multipliers) in the CMOS standard cell. Cooperative operation is handled by requesting one agent to alter something when an other agent prevents the constraints from being satisfied. For example, if a datapath satisfying the constraints cannot be implemented under the controls generated by the control design agent, the data path design agent request changes the control of the control system design agent. In design work, there are cases where redesign is unavoidable. Especially in cooperative systems, where changes may be requested from the outside, efficient redesign is essential. In co-LODEX the problem is resolved through assumption-based reasoning.

(1) Alternatives occurring in design are regarded as assumptions, and change is managed through use/non-use of assumptions in redesign.

(2) Violation of constraints is a contradiction, and redesign cancels the contradiction.

(3) Constraints can also be treated as assumptions, considering that design may be repeated while changing constraints.

(4) Redesign must be handled through operations on the conjunction of delay time conditions, conditions related to basic cell count, and assumptions expressing the cause of the violation of constraints.

8.2 An Expert System for Automatic Determination of Disassembly Sequence of a Head Stock

This system aims at studying efficient inference control mechanisms. To accomplish the aim, problems in automatic generation of disassembly sequence of machine tool head stocks are employed as specific examples. An example of machine tool head stocks is shown in Fig. 8.2.1. This example consists of 80 components, which are represented by the connective relations based on a fitting tolerances and by three-dimensional data using generalized cylinder expression. This system generates the disassembly sequence with the connective relations and from (1) knowledge to extract candidate components for disassembly, (2) knowledge to select the component to be disassembled from the list of candidates, (3) knowledge to evaluate the disassembly cost and to check the inter-component interference, and (4) knowledge to update connective relations. The processing sequence is given in Fig. 8.2.2. The disassembly sequence is generated as follows. First, extraction knowledge is applied to connective relations, and components that can be disassembled are listed. Next, selection knowledge is used to determine the component in the list that is the easiest to disassemble. Selection knowledge is given a priority and ordered, based on ease of disassembly derived from connective relations. Components with the same disassembly priority are evaluated by disassembly cost and checked by inter-component interference knowledge. When a component is removed, connective relations are renewed by knowledge to update connective relations. This process is repeated until all components are disassembled. Even with evaluation knowledge, the sequence will have alternatives because ease of disassembly and cost are equivalent for multiple components. The sequence will also be forced to change by inter-component interference. To solve these problems, this system employed the following approach.

(1) Alternatives of disassembly sequence are handled as multiple contexts.

(2) A hypothetical reasoning mechanism is applied, assuming that alternatives of disassembly sequences are hypotheses, and the evidence for disassembly sequences being inappropriate as shown by the cost evaluation and inter-component interference check is a contradiction.

This research is based on research performed in cooperation with the Mechanical Engineering

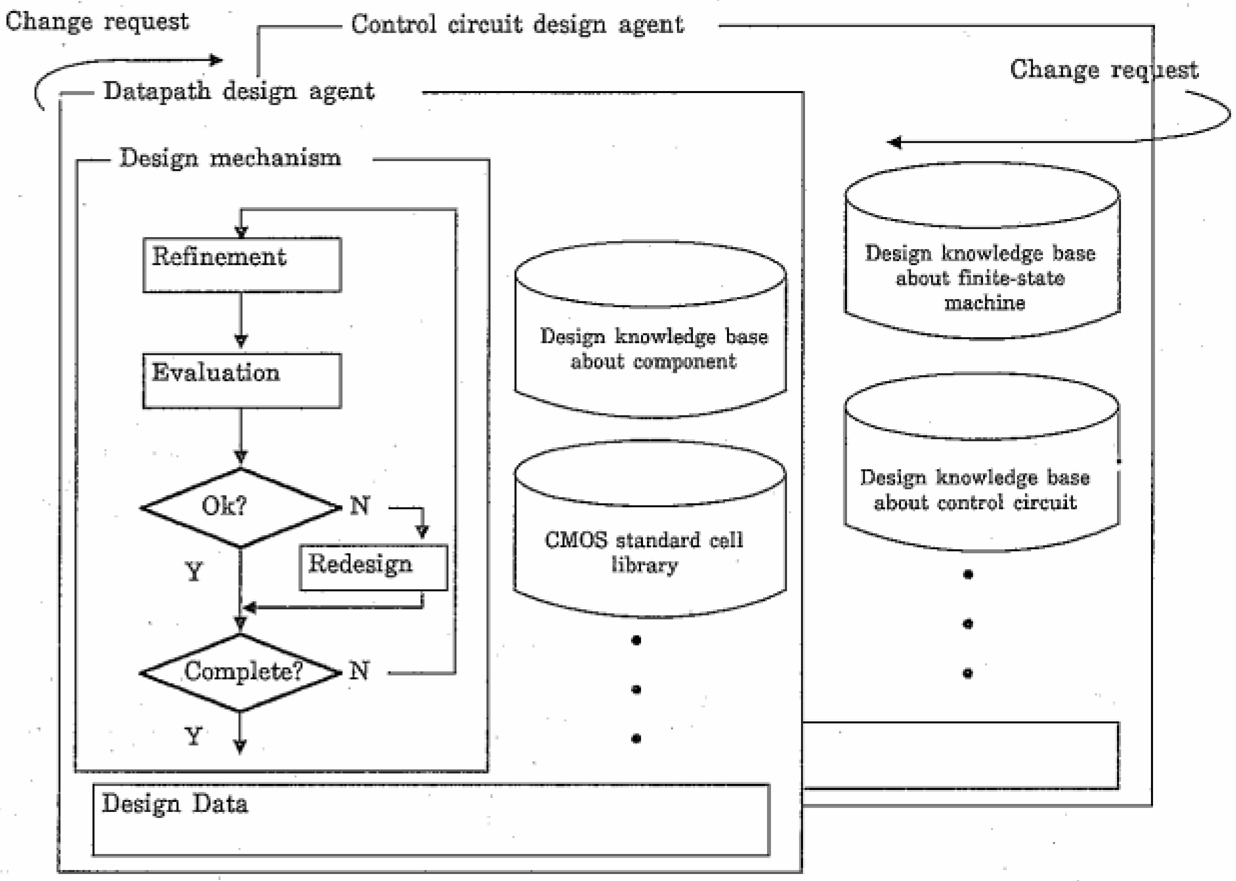

Change request
Control circuit design agent
Change request
Datapath design agent
Design mechanism
Refinement
Evaluation
Ok?
N
Y
Redesign
Complete?
N
Y
Design knowledge base about component
CMOS standard cell library
Design knowledge base about finite-state machine
Design knowledge base about control circuit
Design Data

Fig. 8.1.1 System Structure

Laboratory, Agency of Industrial Science and Technology, Ministry of International Trade and Industry [Sekiguchi 83], [Sekiguchi 87].

8.3 Delivery planning

The delivery planning support system supports the assignment of trucks and drivers, and the selection of routes. The delivery task is that packages from a distribution center are delivered on multiple trucks to multiple destinations (retailers). This system generates satisfactory plans. This problem has some constraints, for example, they are numbers of trucks and drivers as resources, required visiting time intervals for the destinations, and trucks' capacity. The primary target of this system is a feasibility study for distributed cooperative problem solving systems. First, delivery requests (packages) are divided into groups called areas, through heuristics. This means dividing the problem into subproblems. A delivery plan for each area can be generated autonomously by the distributed problem solver called the area agent, but each generated plan is not complete. Fig. 8.3.1 shows a simple example of area agents and delivery requests. Resources for delivery (trucks and drivers) can be shared between areas, so resources for an area need to be allocated in cooperation with other areas. The flexibility of the generated plans is also enhanced by giving a package to multiple areas and through cooperation between the areas. System input consists of route information, truck and driver information, and delivery orders. System output is the delivery plan for a single day. Fig. 8.3.2 shows a simple delivery plan. Multiple candidates are evaluated by the total cost. This system accepts user input with the guide in the event of loops or dead ends. It has a function to explain the plan generation process.

8.4 Troubleshooting Expert System for Electronic Switching Systems

This system infers the probable cause from the symptoms of an electronic switching system fault. The diagnostic process is as given below.

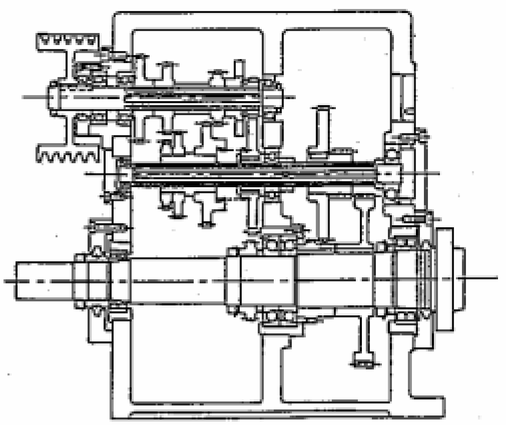


Fig. 8.2.1 Example of a mchine tool headstock

(1) Symptom data is analyzed, and the components that could be causing the problem (suspects) determined.

(2) The suspects are represented as a group of functional blocks.

(3) Effective tests are selected for each suspect and executed. The number of suspects is reduced from the results. This process is repeated.

(4) Finally, the remaining suspect is replaced, and a check made to see if the fault disappears.

Inferences in narrowing down suspected components by test results are frequently indeterminate. In inferences of this type, the final indicated suspect is in error through errors in judgment. If the suspected component is replaced and the fault is still present, it indicates an error (contradiction) in a judgment made up to that point, so that judgment is deleted and a different conclusion is reasoned (a different suspect is selected). A truth maintenance technique such as an ATMS is incorporated in order to realize this kind of indeterminate inference. An ATMS can handle multiple contexts simultaneously. In this case, restrictions on the number of environments handled simultaneously were imposed to prevent a drop in processing efficiency. As an example of an imprecise judgement, it can be assumed that the power supply is normal is the power supply alarm is not active, but if the power supply alarm indicator lamp is broken, there will be no alarm even if the power supply is abnormal. In this situation, the assumption that the power supply is normal is made.
The rule may be expressed as:

Table 1. Connective relations

Connective relations		Code	Level
Fit	Pressure fit	Pr	↑
	Push fit	Pu	
	Screw fit	Sc	
	Taper fit	Ta	
	Spline fit	Sp	
	Position fit	Po	
	Movable fit	Mo	
	Gear coupling	Ge	
	Ring fit	Ri	
	(Key fit)	Ke	
Contact	Clamp contact	Cl	↑
	Taper contact	Ta	
	Plane contact	Pl	
	External contact	Ex	
	Gear meshing	Ge	
	(Gap plane)	Ga	

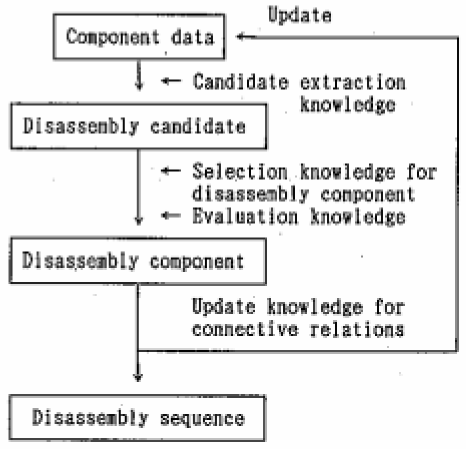

Update
Component data
← Candidate extraction knowledge
Disassembly candidate
← Selection knowledge for disassembly component
← Evaluation knowledge
Disassembly component
Update knowledge for connective relations
Disassembly sequence

Fig. 8.2.2 Processing procedure

if power__supply__lamp__not__on ,
asm__alarm__ok % power supply alarm is normal
then power__supply__is__normal (cause is other than power supply)

In this rule asm__alarm__ok is an assumption. If the truth of this assumption becomes doubtful, a test is executed. If no contradiction is generated, then the test may be omitted, which is an advantage in system performance.

8.5 Computer Layout

This system aims at a feasibility study of parallel processing in problem-solving, using the problem of laying out geometrical shapes within a limited space (that is, computer layout). ESP is used, and the pilot system is implemented on the PSI. The basic configuration of ESP objects is given in Fig. 8.5.1, and the following hierarchical problem-solving approach is tried, taking parallel processing into consideration. The parallel problem-solving object invoked by user request requires an upper-level processing object. The upper-level processing object divides given equipment into groups, and the room into zones so that the problem can be reduced to subproblems by allocating each group to an appropriate zone. Through pseudo-parallel processing realized by the above problem division, a layout satisfying semantical constraint conditions (that is, equipment maintenance areas and pillars may not overlap) is obtained. Pseudo-parallel processing and the reduction of possible answer sets by semantical constraint conditions makes the search for an answer more efficient. Constraint conditions imposed between parallel processes consist of constraints to align the front of units of equipment, and limitations on distances between specified equipment pairs. These are relatively weak constraints, so parallel processing is extremely efficient. However, if there are strong constraints such as the requirement for the front of one zone to be aligned with that of another zone, interference between parallel processes often reduces parallel processing efficiency. Future research is required on the allocation of information acquisition rights to minimize inter-process interference.

8.6 Intelligent secretary

The most common method of building a knowledge base for expert systems is for a knowledge engineer (KE) to interview a domain expert and extract his/her knowledge. In this method, however, a new expert (the KE) is required, and there are not enough KEs to spread expert systems. Much of the knowledge of experts is already available in printed form, in various forms as dissertations and books. As human beings acquire knowledge, they acquire vast quantities of knowledge already arranged systematically through books, and experts and KEs can also form knowledge bases for expert systems from such basic knowledge and experience. If it were possible to convert the natural language found in such documentation into a knowledge base suitable for an expert system, it would be extremely easy to develop expert systems. The goal of this research is to establish methods for converting knowledge expressed in natural language into knowledge bases for expert systems. The field is limited to secretarial work, primarily scheduling, and we present a knowledge acquisition method by using the text knowledge primitive (SKIP) to be problem solution elicited from documentation. A knowledge acquisition method using SKIP requires a function to draw concrete actions from general concepts, and a function to systematize text using multiple jargon and expressions in a single form suitable for a system. These are called the expression systematization function and the structural systematization function. We are studying and developing a knowledge compiler which links these functions. The overall structure of the knowledge acquisition support tool prototype, including the intelligent scheduling system, is shown in Fig. 8.6.1. Current knowledge acquisition support tools process documentation in SKIP format (which restricts scheduling) into a knowledge base for scheduling systems, and then its knowledge base is used for intelligent scheduling.

8.7 Plant Control

This study aims to demonstrate the potential of a logic programming language for application to the plant control domain. A prototype plant control system was constructed and applied to a steam power plant as an example of a control object. Knowledge representation techniques were investigated for describing the structural and dynamic characteristics of the plant in the control domain. Basic functions needed for controlling the power plant are:

(1) Monitoring the states of the plant;

(2) Selecting and/or determining the timing and amount of control action;

(3) Planning the sequence of control actions responding to malfunctions detected or aiming to improve the control performance;

(4) predicting the transition of the states of the plant by simulation.

Here, (3) and (4) were selected as the main subjects of the prototype system.

In the prototype system, the plant model is described in the form of deep knowledge such as physical laws and the structure of the plant. Using deep knowledge, the control system enables control actions to be generated even if an unexpected situation occurs where heuristic control knowledge (shallow knowledge) does not exist. The behavior of the plant responding to the specified control action is predicted. This result is used for evaluating the validity of the sequence of control actions generated.

We employ qualitative reasoning mechanism to realize deep reasoning. In qualitative reasoning, the object plant is modeled qualitatively with the variables and constraints contained in it expressed qualitatively. The behavior of the plant is acquired in the form of the transition of qualitative states. Both of the qualitative reasoning systems based on the methodologies of [Kuipers 86] and [deKleer 84] have been explored and compared. Due to the ambiguity caused by qualitative values, a great number of states are generated which actually never occur. Therefore, neither method can be applied directly to plant control. To avoid combinatorial explosion, pruning the unsuitable candidates of the states based on heuristics proved to be more effective than introducing the full order among the landmarks of different variables.

In future study, the mechanism of generating control actions using deep reasoning must be examined in more detail. Parallel processing and knowledge compilation of the reasoning results are considered to be effective to improve the efficiency of deep reasoning processing.

8.8 Portfolio planning support system

This system is designed to support portfolio planning, where a specific amount of capital is divided among multiple investment options. The user inputs the portfolio problem by specifying profit and safety targets, and the portfolio plan is generated and refined by analysis from multiple viewpoints to output the final portfolio plan, The objective is the establishment of cooperation technology which can reach a solution to the overall problem by exchanging planning information between individual agents in a multi-agent problem resolution process. A knowledge base module of this system stores the knowledge essential for problem resolution by a standard portfolio generation module and a portfolio improvement module. For the standard portfolio, agents are structured primarily on investment options, and for portfolio improvements primarily on evaluation viewpoints for generated investment plans or aggressive investment strategies. The knowledge module is implemented on a parallel logic language (FGHC), in the POOL (Parallel Object Oriented Language for cooperative problem solving) parallel object-oriented language. The POOL program is a set of class descriptions defining objects. Class definitions consist of inheritance, slots and default values, methods and local predicates. One-to-one and broadcasting communication functions are supported as message handling functions. Fig. 8.8.1 shows class hierarchical structuring for the investement option agents.

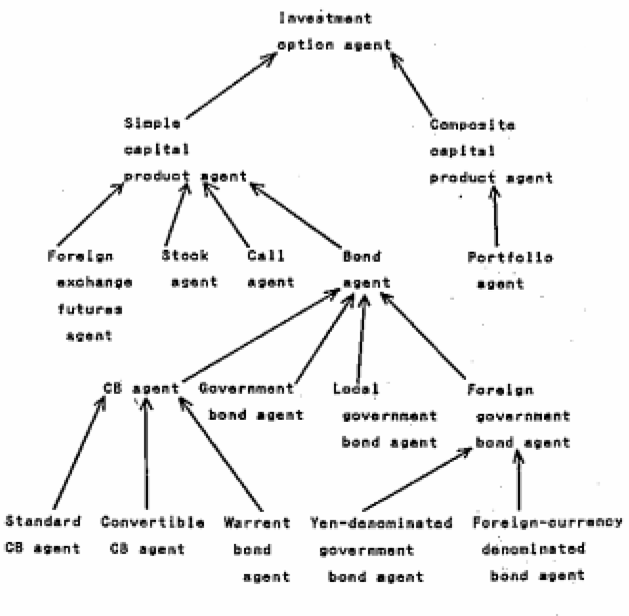

Investment option agent
Simple capital product agent
Composite capital product agent
Foreign exchange futures agent
Stock agent
Call agent
Bond agent
Portfolio agent
CB agent
Government bond agent
Local government bond agent
Foreign government bond agent
Standard CB agent
Convertible CB agent
Warrent bond agent
Yen-denominated government bond agent
Foreign-currency denominated bond agent
: Superclass-Subclass

Fig. 8.8.1 Class Hierarchy for Investment Option Agent

9 CONCLUSIONS

This paper reviewed the current state of research and development on experimental knowledge processing systems. The next steps forwards the final stage will consist of enhancements of functions and technologies, through use and verification of experimental expert systems for individual component technologies. Parallel processing will be introduced, and integration as a next-generation tool will be promoted.

REFERENCES

[Araya 87] Araya, A.A. and Mittal, S., "Compiling Design Plans from Descriptions of Artifacts and Problem Solving Heuristics", Proc. of IJCAI 1987, pp.552-558

[Buchberger 83] Buchberger, B., "Groebner Bases: An Algorithmic Method in Polynomial Ideal Theory", TR CAMP-LINTS (1983)

[Chandrasekaran 86] Chandrasekaran, B., "Generic Tasks in Knowledge-Based REasoning: High Level Building Blocks for Expert System Design", IEEE Expert, Vol. 1 (1986), pp. 23-30

[Davis 80]Davis, R., "Report on the Workshop on Distributed AI", SIGART Newsletter, No. 73, October, 1980

[Decker 87]Decker, Keith S., "Distributed Problem-Solving Techniques: A Survey", IEEE Trans. on System, Man, and Cybernetics, Vol.SMC-17, No. 5 Sep./Oct. 1987

[deKleer 84] de Kleer and Brown, "A Qualitative Physics Based on Confluence", Artificial Intelligence Vol, 24 (1984), pp.7-83

[de Kleer 86a] de Kleer, J., "An Assumption-based TMS", Artificial Intelligence 28 (1986), pp.127-162.

[de Kleer 86b] de Kleer, J., "Extending the ATMS", Artificial Intelligence 28 (1986), pp.163-196

[de Kleer 86c] de Kleer, J., "Problem Solving with the ATMS", Artificial Intelligence 28 (1986), pp.197-224

[Doyle 79] Doyle, J., "A Truth Maintenance System", Artificial Intelligence 12 (1979), pp.231-272

[Durfee&Lesser 87]Durfee, Edmond H. and Lesser, Victor R., "Using Partial Global Plans to Coordinate Distributed Solvers", Proc. of IJCAI 87, pp.875-883

[Fehling 83]Fehling, M., "Report on the Third Annual Workshop on Distributed Artificial Intelligence", SIGART Newsletter, No. 84, April, 1983

[Feldman 88] Feldman, R., "Design of a Depandency-Directed Compiler for Constraint Propagation", Proc. of 1st International Conference on Industrial & Engineering Application of Artificial Intellignece and Expert Systems, IEA/AIE '88, (1988)

[Forbus 84]Forbus, Kenneth D., "Qualitative Process Theory", Artificial Intelligence 24(1984), pp.85-168

[Fujiwara & Inoue 88] Fujiwara, T. and Inoue, K., "A Hypothetical Reasoning System in ESP - ASTRON-"(An ATMS implemented in ESP (Ver. 2)), ICOT Technical Memorandum No. TM-587, ICOT, 1988(in Japanese)

[Gasser 87]Gasser, Les, "The 1985 Workshop on Distributed Artificial Intelligence", AI Magazine, Summer, 1987

[Iijima & Inoue 88] Iijima, K. and Inoue, K., "An ATMS implemented in ESP (Ver. 1), ICOT Technical Memorandum No. TM-467,ICOT, 1988 (in Japanese).
[Inoue 88a] Inoue, K., "Pruning Search Trees in Assumption-based Reasoning", Proc. Avignon '88: The 8th International Workshop on Expert Systems & their Applications (1988), pp.133-151
[Inoue 88b] Inoue, K., "On the Semantics of Hypothetical Reasoning and Truth Maintenance", ICOT Technical Report No. TR-357, ICOT, 1988.
[Inoue 88c] Inoue, K., "Problem Solving with Hypothetical Reasoning", FGCS '88: International Conference on Fifth Generation Computer Systems (1988), in these Proceedings
[Inoue, ed. 88] Inoue, K., (ed.), "Expectations and Images for Hypothetical Reasoning", KSS-WG HYR-SWG Report for 1987, ICOT Technical Memorandum No. TM-487, ICOT, 1988 (in Japanese).
[Inoue et al. 88] Inoue, K., Nagai, Y., Fujii, Y., Imamura, S. and Kojima, T., "Analysis of the Design Process of Machine Tools - Example of a Machine Unit for Lathes -", ICOT Technical Memorandum No. TM-494, ICOT, 1988 (in Japanese)
[Kobayashi 86] Kobayashi, s., "Knowledge Engineering", Shokado, 1986
[Konolige 85] Konolige, Kurt, "Research on distributed artificial intelligence", Stanford Research Institute, AI Center, 1985
[Kuipers 86] Kuipers, B., "Qualitative Simulation", Artificial Intelligence Vol. 29(1986), pp.289-338
[Kuipers 85] Kuipers, B., "Qualitative Simulation of Mechanisms", MIT LCS TM-274, 1985
[Kuipers 87] Kuipers, B., "Abstraction by Time-scale in Qualitative Simulation", Proceedings of AAAI-87(1987), pp.621-625
[Nagai 88a] Nagai, Y., "Towards Desgin Plan Generation for Routine Design using Knowledge Compilation -Forcusing on Constarint Representation and its Application Mechanism for Mecahnical Design-", ICOT Techincal Memorandum, TM-504, (1988)
[Nagai 88b] Nagai, Y. "Towards an Expert System Architecture for Routine Design -Focusing on Constraint Representation and an Application Mechanism for Mechnical Design-", ICOT Technical Memorandum, 1988
[Nii 86] Nii, H.Penny , "Blackboard Systems",Technical Report No. STAN-CS-86-1123, June 1986
[Ohki 88] Ohki, M., "Towards Qualitative Physics", ICOT-TR-221, 1988, (to appear)
[Ohsuga 85] Ohsuga, S. "Conceptual Design of CAD Systems Involving Knowledge Bases", Knowledge Engineering in Computer-Aided Design(Gero, J.S. (ed.)), North-Holland, pp.29-50, 1985
[Oosaki et al. 88] Oosaki, H., Tsubaki, K. and Taki, H., "Knowledge Acquisition Support System EPSILON/One (2)", Proceeding of 8th SICE Knowledge Engineering Symposium, 1988 (in Japanese)
[Poole 88] Poole, D., "A Logical Framework for Default Reasoning", Artificial Intelligence 36 (1988), pp.27-47
[Reiter 80] Reiter, R., "A Logic for Default Reasoning", Artificial Intelligence 13 (1980), pp.81-132
[Sakane 88] Sakane, K., "Methods for Partition of Target Systems in Qualitative Reasoning", Proceedings of FGCS '88 (1988)
[Sekiguchi 83] Sekiguchi, H., Kojima, T. and Inoue, K., "Study on Automatic Determination of Assembly Sequence", Annals of the CIRP, Vol. 32, No. 1 (1983), pp.371-374
[Sekiguchi 87] Sekiguchi, H., Imamura, S., Kojima, T. and Inoue, K., "Method of Developing Part Specifications from Assembly Drawing of Machine Unit (2nd Report) -Automatic Determination of Assembly/Disassembly Sequence-" Journal of the JSPE, Aug.1987, pp.1183-1188 (in Japanese)
[Shastri 85] Shastri, L., "Evidential Reasoning in Semantic Network: A Formal Theory and its Parallel Implimentation", TR166, The University of Rochester,Sept. 1985
[Shostak 77] Shostak, R. E., "On the SUP-INF Method for Proving Presburger Formulas", Journal of ACM, 24 (1977), pp.529-543
[Smith 85]Smith, R.G., "Report on the 1984 Distributed Artificial Intelligence", AI Magazine, Fall, 1985.
[Taki et al. 87] Taki, H., Tsubaki, K. and Iwashita, Y., "EXPERT MODEL for Knowledge Acquisition", IEEE Expert Systems in Government Conference, 1987
[Taki 88] Taki, H., "Knowledge Acquisition by Observation", Proceedings of FGCS '88, 1988
[Tanaka 88] Tanaka, H., "Temporal-hierarchical Qualitative Reasoning and its Application to Medicine", Proceedings of Logic Programming Conference '88, pp.11-17 (1988)
[Terasaki 88] Terasaki, S. Nagai, Y. Yokoyama, T. Inoue, K. Horiuchi, E. and Taki, H., "Mechanical Design Expert System Constructing Tool, MecanICOT", SIG-KBS, JSAI, Oct. 1988 (in Japanese)
[Tomiyama 85] Tomiyama,T. and Yoshikawa,H. "Requirements and Principles for Intelligent CAD Systems" Knowledge Engineering in Computer-Aided Design(Gero,J.S. (ed)), North-Holland, 1985 pp.1-23
[Tsubaki et al. 88] Tsubaki, K., Oosaki,H. and Taki,H., "Knowledge Acquisition Support System EPSILON/One (1)", Proceeding of 8th SICE Knowledge Engineering Symposium, 1988 (in Japanese)
[Yamazaki et al. 87] Yamazaki, T., Taki, H., Tsubaki, K., "Classification Task Acquisition System Based on Generic Tasks CTAS", Proceeding of 1st JSAI Knowledge Base Research Meeting, 1987 (in Japanese)
[Yokoyama 88] Yokoyama, T., "FREEDOM: A Knowledge Representation System for Design Object Modeling"(in Japanese) WGAI, IPSJ, Preprints, 88-60 1988, ICOT Technical Memorandum No. TM-467, ICOT, 1988 (in Japanese)