ICOT Technical Memorandom: TM-1157

TM-1157

Constraint Analysis and Plan Generation of Constraint Compiler for Design Problems

by
Y. Nagai (Toshiba) & S. Terasaki

February, 1992

© 1992, ICOT

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

Constraint Analysis and Plan Generation of Constraint Compiler for Design Problems

Yasuo Nagai
Information Systems Engineering Lab., Toshiba Corp.
70, Yanagi-cho, Saiwai-ku, Kawasaki, 210, Japan

Satoshi Terasaki
Research Center, Institute for New Generation Computer Technology
Mita Kokusai Building, 21F
1-4-28, Mita, Minato-ku, Tokyo, 108, Japan

SUMMARY This paper deals with constraint analysis and plan generation of a constraint compiler for design problems. [1] Knowledge compilation is a technique by which knowledge about a domain (facts and theories) is stored in declarative form and is applied using problem solving mechanisms. We introduce constraint compilation as a knowledge compilation technique that focuses on the concept of constraint. Our constraint compilation for design problems is designed to transform the input design specifications into the design plan, assuming that the structure of the design object has been determined. First, we describe the basic concepts of knowledge compilation, knowledge compilation for design problems, and design plan generation through this method, assuming the target architecture of the design expert system. Next, we describe an actual constraint compiler that applies the concept of constraint compilation to mechanical design, MECHANICOT. In particular, we describe an overall flow of the constraint compiler, concentrating on algorithms that analyze constraints and generate a design plan, and describe the process of design plan generation through constraint compilation. Finally, we give details of design plan generation using this compiler, giving as an example the problem of gear unit design.

1 Introduction

A large quantity of knowledge that depends on specific design objects is required to develop knowledge-based systems for design problems. Therefore, knowledge representation specific to the design problem and efficient problem solving method that uses this knowledge are key points for research on a tool for building design expert systems [14].

In this case, we want to avoid describing design knowledge as existing procedural programs and avoid forcing system builders to describe design knowledge in a form which is dependent on the specific problem solver. In most design knowledge, relations between objects may be represented using symbolic descriptions such as mathematical formulas. When we compare declarative and procedural representation of design knowledge, the former provides a higher-level and more natural description than the latter. In short, procedural representation must be given explicitly, and declarative descriptions need not be.

Here, relations between objects are represented as constraints to conveniently handle design knowledge as a declarative representation. Accordingly, declarative knowledge representation is a more sophisticated representation than existing procedural knowledge representations and is effective for building knowledge-based systems for design problems.

Therefore we can expect to reduce the time spent in unnecessary searching and to improve the efficiency of our problem solver because the solution space to be searched can be restricted by means of the effective utilization of constraints.

We will focus on this advantage of constraints and consider the realization of constraint-based problem solving for design problems, while regarding knowledge compilation as the most important technical issue.

We shall realize knowledge compilation by focusing on the concept of constraint; we define this technique as constraint compilation. In design problems, constraint compilation can generate an efficient design plan automatically, using design knowledge represented declaratively. Before discussing knowledge compilation in detail, we describe a formalization of knowledge-based systems for design problems and the architecture of design expert systems in Section 2.

Section 3 gives an overview of knowledge compilation and the knowledge compiler, with emphasis on design problems.

Section 4 describes constraint analysis and design plan generation of a constraint compiler. We describe the characteristics of the mechanical design problems which have guided us in analyzing the problem. Then we define

[1] This paper describes research done mainly at the Fifth Research Laboratory (Intermediate stage from 1985 to 1989) of the Institute for New Generation Computer Technology (ICOT).

in detail the process of design plan generation through constraint compilation and describe the constraint compiler. Next we describe an actual constraint compiler that applies the concept of knowledge compilation to mechanical design, MECHANICOT. We give the details of constraint analysis and design plan generation using MECHANICOT, giving as an example the problem of gear unit design.

Finally, related works are provided in Section 5 and current research and future research are described in Section 6.

2 Target Architecture of Design Expert System

2.1 Formalization of Knowledge-Based Systems for Design Problems

Tong [22] presents a process for developing knowledge-based systems that can be reconstructed. This is based on factoring the space of detailed problem solving system specifications into three levels of specification: the knowledge level, the function level, and the program level. Newell [18] describes and distinguishes the knowledge level and the symbol or program level. He views these as levels for describing computer systems.

According to their formalization, we will divide a formalization of knowledge-based system for design problems into three levels: the knowledge level, the problem solving level, and the implementation level, as shown in Fig. 1. Our problem solving level corresponds to Tong's function level.

At the knowledge level, after the type of design problem is determined, the corresponding solution space, design theory, and design specification are formalized. At the problem solving level, the constraint-based problem solving mechanism is determined according to the design process. This problem solving mechanism can be realized by applying various solution methods such as problem decomposition, constraint propagation, failure recovery, (hierarchical) generate & test, least commitment, and (linear) approximation method [14]. At the implementation level, knowledge-based systems are realized using expert shells composed of knowledge representations, such as rule and frame descriptions, and programming languages, such as logic programming and constraint logic programming languages, according to the fixed architecture of these systems.

Furthermore, consideration of a modeling of the design process is required to realize a problem solving mechanism for design problems using a constraint satisfaction process [5], [14]. Fig. 2 shows a model of the design process for routine mechanical design. This routine mechanical design problem consists of editing pre-existing designs, and is based on this design process model [5]. The structure of the design object is determined by combining the components or is according to predefined design styles of the design object. In this case, the structures are determined by retrieving the appropriate design style from the knowledge base. The components are implemented using standard parts found in catalogs or non-standard parts from the design.

2.2 Architecture of a Design Expert System

Research has been conducted on architectures consisting of primitive tasks for routine design, called generic tasks for design [4], [17]. These architectures provide the ability to structure knowledge for the various design descriptions and provide problem solving for the design to reduce the gaps between the functions for the design process at the knowledge level and the functions supported by expert system building tools at the program level. However, it seems that they are insufficient to handle the constraint representation [20] for this generic task. Therefore, we investigate the architecture of an expert system for routine design using the constraint-based problem solving mechanism [14]. The architecture of design expert systems which use the constraint-based problem solving mechanism is composed of the following primitives: the generator, the propagator, the tester, and the failure recovery module, as shown in Fig. 3. This problem solving mechanism is extended based on a generate & test method.

3 Knowledge Compilation

3.1 Knowledge Compilation

Knowledge compilation is a common term in artificial intelligence research on problem solving, model-based reasoning (MBR), machine learning, software engineering, and automatic programming [2], [9], [11]. In Ref. [9], "knowledge compilation" takes on a broader meaning and can be classified into at least four alternatives:

- The process of shifting from a declarative to a procedural form of knowledge representation.
- The process of automatically transforming explicit but inefficient knowledge representations into implicit but more efficient form.
- The process of producing knowledge-based systems from higher level specifications.
- The process of automatically restructuring existing software systems to produce new systems that exhibit an increase in efficiency or usability, a change in representation level, and a reduction of reasoning.

3.2 Knowledge Compilation for Design Problems

In this section we define knowledge compilation for design problems [14], [15].

In mechanical engineering there are many cases when design systems or tools are provided for each design object. In fact, the individuality of the design object makes it difficult to abstract, arrange, and utilize the design systems or tools because the corresponding model and analysis method for the object often change when the structure of the design object changes.

At present the individualized design systems or tools for mechanical design implemented using a typical procedural language such as Fortran are inconvenient to use and inefficient for designers.

To reduce the inconvenience and inefficiency of existing design environments, we need to provide an environment in which the designer can apply the approach used in the existing compiler [1] to perform mechanical design and to construct design systems or tools easily.

By dividing design knowledge into knowledge about the design object and knowledge about problem solving we may handle both types of knowledge effectively and improve the efficiency of the problem solving mechanism of the whole system. Viewing knowledge and requirements as constraints, we can regard design problems as constraint satisfaction problem [5]. Knowledge transformation is a very important technique for handling these independent kinds of knowledge uniformly, to generate design plans according to primitives of constraint-based problem solving, and to improve the efficiency of problem solving.

We will consider two methods for transformations. The first method is knowledge compilation, which generates design plans by analyzing and compiling knowledge about design objects and about problem solving and which builds design systems. It is suitable for parametric design where the structure of the design object is fixed, and an efficient problem solving mechanism can be realized [11], [15], [23].

The second method is to translate knowledge about object models and knowledge about problem solving into intermediate descriptions and to interpret these intermediate descriptions. This is an interpretation approach, which corresponds to the interpretation of a design plan. Furthermore, it is possible to interpret a design plan generated during problem solving efficiently. This is done by reusing knowledge derived beforehand through knowledge transformation techniques (including the design plan and its intermediate description).

At first, we focus a design problem where the structure of the design object and knowledge about problem solving are fixed. Therefore, we adopt knowledge compilation as a knowledge transformation technique. In our research, knowledge compilation means to transform knowledge representation (problem specifications) at an abstract level to representation at a more concrete level; from knowledge level to problem solving level and from knowledge level to implementation level [14], [15]. The purpose of this method is to improve the efficiency of the utilization of various kinds of knowledge and the problem solving mechanism.

We consider knowledge compilation a technique by which knowledge in declarative form, such as facts and theories, about the domain is stored and by which knowledge linked to an efficient problem solving task is generated. This stored or generated knowledge is applied and utilized by interpretive procedures for performing a task in that domain. Therefore, more efficient procedures specific to the task domain can be generated using the knowledge compilation technique. Furthermore, in our opinion, knowledge compilation can help knowledge acquisition to the extent that it helps users to construct new knowledge-based systems easily [13] and can facilitate knowledge reuse.

4 Constraint Compiler for Design Problems

We define knowledge compilation techniques that focus on the concept of constraint as constraint compilation and describe an actual constraint compiler that applies this concept. In this case, we represent a set of constraints in terms of a constraint network. Constraint networks involve a set of n variables or parameters and their domains of finite or infinite values [7].

4.1 Overview of Constraint Compiler

In Ref. [8], a constraint compiler is intended to optimize constraint propagation and to improve the efficiency of constraint propagation for large constraint networks by performing a dependency analysis of constraints. In other words, it makes existing paths of constraint processing (constraint propagation) more efficient rather than enabling new paths.

Our constraint compiler transforms the input design specifications into the design plan by compiling design knowledge and problem-solving heuristics, assuming that the structure of the design object has been determined [14].

Since the constraint compiler can generate a design plan by analyzing dependencies among constraints, design knowledge can also be represented declaratively. The general flow of the constraint compiler is shown in Fig. 4. The compiler contains four main procedures: *lexical and syntax analysis*, *inheritance relations analysis* (by *generation of class definition tables*), *constraint analysis* (by both *generation and update of tables* and *determination of a constraint analysis sequence*), and *design plan generation*.

Inputs to this constraint compiler are the design requirements, object models, and knowledge about problem solving. The output is a design plan. Reference to the results of previous designs and the designers' heuristics about searching for alternatives are also represented

as knowledge about problem solving.

In *constraint analysis sequence determination*, relations between components are analyzed and a directed-acyclic graph (DAG) that represents part-whole relations and abstract-concrete relations is generated using the results of analysis. The *constraint analysis sequence* is determined according to this graph. In *constraint analysis*, dependencies among constraints inside components and functional blocks are analyzed according to this determined sequence. Inheritance relations between functional blocks and between components are analyzed in *inheritance analysis*.

In *design plan generation*, a design plan that enables efficient problem solving mechanisms is generated according to the dependencies between constraints assuming the architecture of the design expert system mentioned in Section 2.

4.2 Constraint Analysis and Design Plan Generation

(a) Constraint Analysis

Handling the constraint network in constraint analysis is difficult, because the structure of the network is quite complicated for complex systems. Therefore it is necessary to structure the network with this hierarchical level of abstraction and to combine subnetworks, instead of handling the whole network as a flat structure. Thus, an effective utilization of this structural information of the problem space extracted using the constraint analysis will lead to improvements in constraint-based problem solving. In the process of constraint propagation, local and non-local (global) propagations must be taken care of [12]. In local propagation, known values are propagated along the arcs of the constraint network by using local information to nodes [20]. On the other hand, when there is a cyclic dependency in the constraint network using a local propagation, the problem can not solved by local constraint propagation only. In this non-local (global) propagation, global information such as an equation solving technique, is required to deduce values of variables.

Therefore, to improve the constraint-based problem solving mechanism, especially constraint propagation, it is necessary to separate the constraint network description of the design plan into parts that can be processed by local propagation (tree description) and parts that require non-local propagation (cyclic description), in constraint analysis. Consequently, each description should be dealt with using the appropriate solver for constraint handling by splitting up the constraint network. In order to deal with constraint analysis for structured networks with hierarchical levels, the analysis phase is divided into two phases. Fig. 5 shows the whole algorithm of constraint analysis. In this algorithm, only analysis for the constraint-based problem solving mechanism based on local propagation is realized. The analysis is executed according to the tree description composed of *part-whole* relations and *is-a* relations of the system. This constraint analysis executes the same procedure as the dataflow analysis in optimization phase of the compiler [1]. In other words, phase one corresponds to the local optimization, i.e. optimization of a basic block and phase two corresponds to the global optimization, i.e. global rearrangement of the dataflow graph generated during phase one.

Phase one proceeds from the bottom up; it performs dataflow analysis [1], [8] and reduction (merging) for each component (line 3-5) and functional block (line 6-7).

Phase two instead starts from the root and proceeds from the top down towards the leaves: the dependencies of the constraint network [7] are determined by reanalyzing constraint dependencies among functional blocks, and between functional blocks and components. Finally the dependencies among the constraints of the whole system are determined (line 8-9).

Concretely, constraint analysis begins at the lowest level of the class hierarchy and proceeds towards the highest level class. If there are inheritance relations, the constraints are not processed along the class hierarchy between parent classes and children classes, but are treated as a flat set of constraints included in both parent classes and their children classes.

(b) Design Plan Generation

After dependencies among constraints inside components and functional blocks are obtained by constraint analysis, design plan generation proceeds towards the higher levels of the hierarchy of the design object using the results of the analysis. Thus, a design plan is generated by grouping the blocks analyzed above and reanalyzing them.

4.3 MECHANICOT

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. By regarding knowledge and design requirements as constraints, we employ constraint-based problem solving. To help designers build an expert system suitable for a design problem, we propose a building tool that regards design knowledge as constraints, generates design plans by analyzing their dependencies, and provides an interface between the design knowledge and the constraint solver. We used a constraint compiler 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. 6 shows the architecture of the building tool. An expert system building tool, MECHANICOT, is being developed [14], [15]. MECHANICOT is a tool for 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. In this case, the main spindle head of a lathe is selected as the design object.

4.4 Example of Constraint Analysis and Design Plan Generation Using MECHANICOT

First, we will explain the design object of MECHANICOT, the main spindle head of a lathe. Next, we will describe in detail the constraint analysis and design plan generation process, using a gear unit as an example.

4.4.1 Design Problem of Main Spindle Head

The design object of MECHANICOT is the main spindle head of a lathe, shown in Fig. 7. It consists of a main spindle to grip and rotate a workpiece, a motor as a power source, V-belts and a pair of pulleys to transmit power from the motor to a pulley-shaft, bearings to support both the main spindle and the pulley-shaft, and two pairs of gears to change the main spindle speed.

This design is a typical example of parametric design. It can be considered as the problem of determining the output (design) parameters so that the design requirement and input parameters are satisfied when the structure of the design object is assumed to be fixed [10].

In Table 1, there are two types of input parameters (design requirements): cutting capacity requirements and evaluation criterion. Their default values are shown at the right side of Table 1. Table 2 shows examples of output (design) parameters.

4.4.2 Example of MECHANICOT

In this case, we give an example of constraint analysis and design plan generation using a gear unit as an explanation of MECHANICOT. Fig. 8 is a schematic description of a gear unit used in the reduction system of a main spindle unit. Fig. 9 shows an example of MECHANICOT system appearing in PSI windows; (a) shows a compilation process of a gear unit design problem, (b) shows a hierarchy of design knowledge (design object), and (c) shows a class definition of design knowledge.

Fig. 10 shows a constraint network for this design problem formalized from the viewpoint of the concept of a constraint, especially an algebraic constraint. Forms of many constraints consist of equalities and inequalities. There are some alternatives when generating the values for the variables. Since there are several possible solution methods (strategies), solving the problem can be efficient or inefficient according to the solution method used for a problem. Although the problem can be formalized as a constraint network, the available strategies may not lead to an efficient solution. Types of problem-solving primitives to insure efficient problem solving must be provided to deal with practical problems.

Therefore, each constraint is assigned a type such as *generator, tester, design_method, filter, and constraint*. For example, *design_method* shows the functions and search methods from catalogs and tables which have no alternatives, and *generator* expresses the functions and search methods from catalogs or tables which have alternatives. *Tester* represents equalities and inequalities used for testing solutions from *generator* or for evaluating solutions. *Filter* is used for adjusting solutions to standard values. For example, a generator *m_gen* generates m from the standard values set, a generator *Pd_gen* generates Pd from the minimum and maximum values, and *tester* tests the gear ratio R_g. In the middle we show the dataflow description of the *design method* for calculating an input shaft diameter. In this design method, a twisting moment T_{in},a shearing strength G_{in}, and a torsion angle θ_{in} are input. Input shaft diameter D_{in} is the output.

In MECHANICOT, constraint analysis consists of two phases. In phase one, dataflow analysis [1] is executed for the *input shaft*, *output shaft*, and *pair of gears* that correspond to leaf parts of the tree description (Fig. 8). In this dataflow analysis, each constraint is interpreted as a function or relation according to assigned types. Next, traversing toward the root of the tree, dataflow analysis proceeds in the functional block (*gear unit*) in the upper part of the tree description. After that, we perform dataflow analysis for the *gear unit* and its components, including the *input shaft*, *output shaft*, and *pair of gears*. The constraint analysis terminates when the root of the tree is reached.

In phase two, the analysis is executed from the root of the tree to the leaves. In this phase, dependencies between the functional block (*shaft*) and the components (*input shaft*, *output shaft*, and *pair of gears*), and dependencies between the components themselves, are reanalyzed; finally the dependencies of the whole system are analyzed.

In design plan generation, subgoals are assigned to *constraint*, *generator*, *tester*, *filter*, and *design_method*, so that the name of each subgoal is unique. Subgoals are integrated into goals based on the input-output dependencies of parameters generated by dataflow analysis. Names are assigned to goals in exactly the same way as they are assigned to subgoals. Finally, an execution sequence for goals is determined based on their input-output dependencies. This sequence is managed in a goal that is one level higher than the included subgoals.

Fig. 11 shows a design plan generated according to

this analyzed result. This figure consists of dataflow descriptions for each gear and for the pair of gears. We assume that the relationships among the goals and subgoals correspond to the hierarchical relationships of the design object shown in Fig. 9 (b); the relationships between the components and subcomponents and the design method for the components and subcomponents are formalized and given in advance as the model description of the design object in the knowledge base. From the input design specifications including the design requirements, object models, and knowledge about problem solving shown in Fig. 12 (a), we generate a design plan using constraint compilation and finally obtain a plan written in ESP code, shown in Fig. 12 (b). Fig. 13 shows the hierarchical relationships between goals and subgoals of this generated design plan. Table 3 shows the number of constraints used in the plan generation.

5 Related Works

Araya & Mittal [3] present a method by which the design plans can be automatically generated by compiling knowledge about artifacts, problem solving heuristics, and characteristics of specific problems. Design plans tailored to particular problems are generated and the potential benefits of maintaining a knowledge base are provided, by reusing the same knowledge for different purposes and by providing a framework for more systematic knowledge acquisition. They propose a method for design plan generation, while our approach concerns a design plan generation environment and its implementation by focusing on a constraint compilation.

Keller [11] shows the compiler that compiles redesign rules from a device model through several levels of abstraction, including the structure/behaviour device model, the qualitative equation model, the causal dependency structure, the redesign goal tree, and the abstract redesign plan. His approach aims to improve efficiency by modifying the general but inefficient model used on MBR. In our approach, we focus on assisting knowledge acquisition and knowledge reuse facilities.

Serrano & Gossard [19] present methods for maintaining consistency in systems of constraints. These provide designers with assistance during the early stages of design and help to close the gap between novice and experienced designers. Effective constraint handling is very important for the development of a knowledge-based conceptual design. They have implemented a constraint-based environment for mechanical computer aided design and their aim is similar to our aim of constraint (knowledge) compilation. However, the problem solving mechanism is not focused on as much as in our approach, from the points of view of constraint solving and constraint-based problem solving.

6 Current State and Future Research

The first implementation of the design plan generation environment, MECHANICOT, including the constraint compiler, is being carried out using the Extended Self-contained Prolog (ESP) language [6] on the personal sequential inference (PSI) machine [21].

MECHANICOT is an automated system with no user interaction. It receives design requirements and design object representations written in an ESP-like language as input, and generates the design plan written in ESP as output. The execution mechanism of the generated design plan is realized using the inference mechanisms in the ESP language, such as unification and backtracking mechanisms. As a result, a design calculation is performed and the figure of the main spindle head with dimensions is shown on the graphic display.

So far, we only handle static and obligatory (hard) constraints [14]. For example, the interpretation of a constraint is fixed, because the role of a constraint such as a generator and tester on a constraint-handling mechanism is predetermined. The handling of suggestive (soft) constraints [14] and of dynamic constraints such as the addition, deletion and modification of constraints during design process has not yet been investigated. Both static and dynamic analysis for constraints, including constraint relaxation [14], are required to realize dynamic constraint handling, when we consider a current constraint compiler.

Next, we will consider the following extensions of constraint analysis to utilize the structure of the problem space effectively and to improve constraint-based problem solving. We need to properly treat the different priorities of the constraints and to interpret the direction of an information flow of constraints dynamically. Furthermore, design plan generation requires the scheduling of goals and subgoals according to a rough prediction of the necessary cost of problem solving gained during constraint analysis.

When a constraint compiler is used as a building tool, the designer needs to be able to generate design plans automatically, recognizing underconstrained and overconstrained states in the constraint network. However dataflow analysis seems to be unsuitable for a constraint analysis that checks this state. To detect overconstrained and underconstrained states (involving cyclic description in a constraint network), we must consider a structural analysis method that analyzes dependencies between variables of constraints (topology information of the constraint network) by considering the constraint network as a bipartite graph [16].

7 Conclusion

This paper has considered a method of constraint analysis and design plan generation of the constraint compiler

for design problems, a form of knowledge compiler. It has been conducted while assuming the architecture of expert systems based on constraint-based problem solving.

We have demonstrated the effectiveness of the constraint compilation technique on a mechanical component, a main spindle head of a lathe, by developing an expert system building tool MECHANICOT. Currently, MECHANICOT does not provide a friendly user interface, i.e. one where the designer can give knowledge about design requirements and the design object in the form of a schematic description as an input, and also interact with the system.

Our future plan is to provide such an environment in which designers can acquire design knowledge by arranging and coordinating their knowledge and in which they can construct design systems or tools easily by representing design knowledge in a natural form and by recognizing underconstrained and overconstrained states of the systems.

Acknowledgments

I would like to express thanks to Mr. Yuiti Fujii (NTT Corp.), Mr. Kenji Ikoma (NTT Data Corp.) and the other members of the Fifth Research laboratory (Intermediate stage) Mr. Takanori Yokoyama (Hitachi Corp.), Mr. Katsumi Inoue, Mr. Eiiti Horiuchi (Mechanical Engineering Lab.) and Dr. Hirokazu Taki (Mitsubishi Corp.) for their helpful comments. I would also like to thank Mr. Masahiro Hoshi, JIPDEC, for implementation of the MECHANICOT system and Prof. Isao Nagasawa, Kyusyu Institute of Technology, for useful suggestions and comments on the needs of knowledge compilation for mechanical design. I would like to thank Dr. Ryuzo Hasegawa, Chief of the Fourth and Fifth Laboratories and Dr. Kouichi Furukawa, Deputy Director of the ICOT Research Laboratories for helpful comments and suggestions. Finally, I would like to express special thanks to Dr. Kazuhiro Fuchi, Director of ICOT Research Center, who has given me the opportunity to carry out research in the Fifth Generation Computer Systems Project.

References

(1) Aho A. V. and Ullman J. D. : "Principles of Compiler Design", Addison-Wesley Publishing Company (1977).

(2) Anderson J. R. : "Knowledge Compilation: The General Learning Mechanism (Chapter 11)", Machine Learning, An Artificial Intelligence Approach, **2**, Michalski R. S., Carbonell J. G. and Mitchell T. M. (ed.), pp.289-310, Morgan Kaufmann Publisher, Inc. (1986).

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

(4) Chandrasekaran B. : "Generic Tasks in Knowledge-based Reasoning: High-Level Building Blocks for Expert System Design", IEEE EXPERT, **1**, 3, pp.23-30 (1984).

(5) Chandrasekaran B. : "Design Problem Solving: A Task Analysis", AI Magazine, **11**, 4, pp.59-71, Winter (1990).

(6) Chikayama T. : "Unique Features of ESP", Proc. of International Conference on Fifth Generation Computer Systems, pp.292-298 (1984).

(7) Dechter R. and Pearl J. : "Network-based Heuristics for Constraint Satisfaction Problems", Artificial Intelligence, **34**, pp.1-38 (1987).

(8) Feldman R. : "Design of a Dependency-Directed Compiler for Constraint Propagation", Proc. of 1st International Conference on Industrial and Engineering Application of Artificial Intelligence and Expert Systems (IEA/AIE-88), **1**, pp.141-146 (1988).

(9) Goel A.K. (ed): "Knowledge Compilation : A Symposium", IEEE EXPERT, pp.71-93, April (1991).

(10) 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 - ", (in Japanese), ICOT-Technical Memorandum (1988).

(11) Keller R.M. *et al.* : "Compiling Redesign Plans and Diagnosis Rules from a Structure/Behavior Device Model", Technical Report KSL89-50, Knowledge Systems Laboratory, Stanford Univ. (1989).

(12) Leler W. : "Constraint Programming Languages", Addison-Wesley Publishing Company (1988).

(13) Mizoguchi R., Yamaguchi T., and Kakusyo O. : "Towards Establishment of a Methodology for Building Expert Systems" (in Japanese), Technical Report of Japanese Society for Artificial Intelligence, SIG-KBS-8801-2 (1988).

(14) Nagai Y., Taki H., Terasaki S., Yokoyama T., and Inoue K. : "A Tool Architecture for Design Expert Systems", Journal of Japanese Society for Artificial Intelligence (in Japanese), **4**, 3, pp.297-303 (1989).

(15) Nagai Y. and Terasaki S. : "Towards Constraint Analysis and Plan Generation of Constraint Compiler for Design Problems", (in Japanese), Procs. of the 3rd Annual Conf. of JSAI, 11-43, pp.693-696 (1989).

(16) Nagai Y. and Ikoma K. : "Constraint Analysis and Plan Generation Based on Graph Theory", (in Japanese), Procs. of the 40th Annual Convention of IPSJ, 1D-4, pp.218-219 (1990).

(17) Nagasawa I. : "Design Expert System", (in Japanese), IPSJ, **28**, 2, pp.187-196 (1987).

(18) Newell A. : "The Knowledge Level", Artificial Intelligence, **18**, pp.87-127 (1982).

(19) Serrano D. and Gossard D. : "Constraint management in MCAE", Artificial Intelligence in Engineering: Design, (Jero, J.S. (ed.)), pp.217-240, Elsevier (1990).

(20) Sussman G. J. and Steel Jr. G. L. : "CONSTRAINT - A Language for Expressing Almost-Hierarchical Descriptions", Artificial Intelligence, **14**, pp.1-39 (1980).

(21) Taki K., Yokota M., Yamamoto A., Nishikawa H., Uchida S., Nakajima N., and Mitsui M. : "Hardware Design and Implementation of the Personal Sequential Inference Machine (PSI)", Proc. of International Conference on Fifth Generation Computer Systems, pp.398-409 (1984).

(22) Tong C. : "Towards an Engineering Science of Kowledge-based Design", Artificial Intelligence in Engineering, **2**, 3, pp.133-166 (1987).

(23) Tong C. : "The Nature and Significance of Knowledge

Compilation", In Knowledge Compilation: A Symposium (Goel, A.K, *et al.*), IEEE EXPERT, pp.88-91, April (1991).

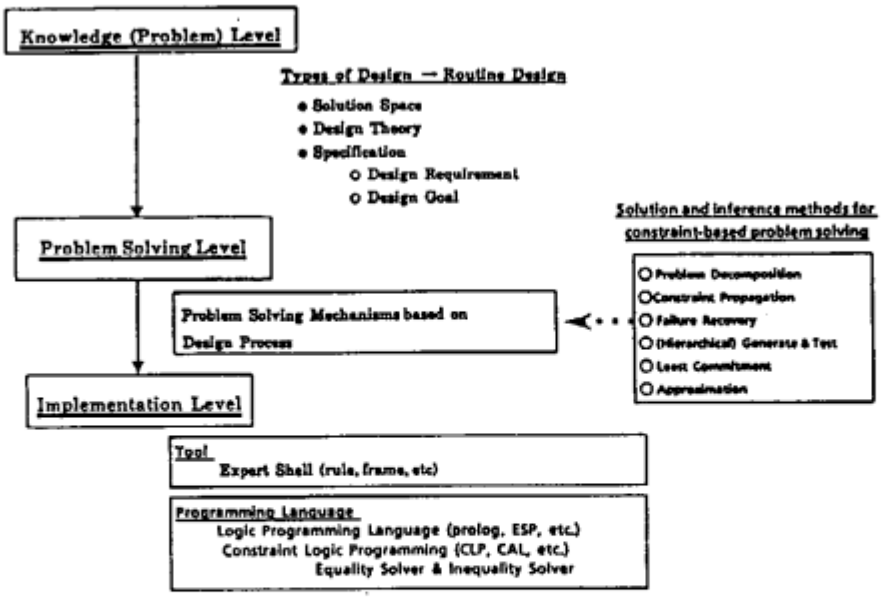


Fig.1 Formalization of Knowledge-Based Systems

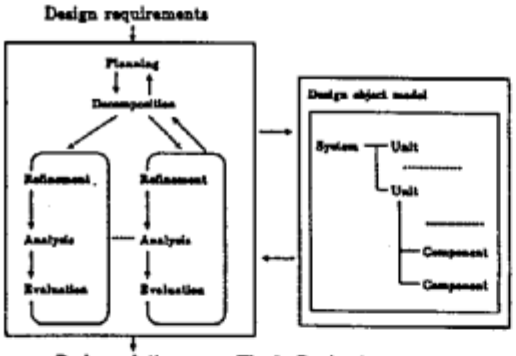


Fig. 2 Design Process Model

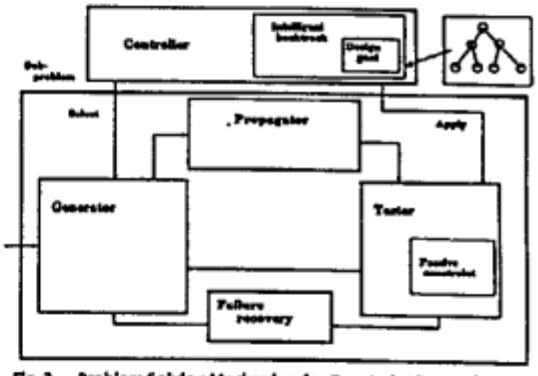


Fig. 3 Problem Solving Mechanism for Constraint Reasoning

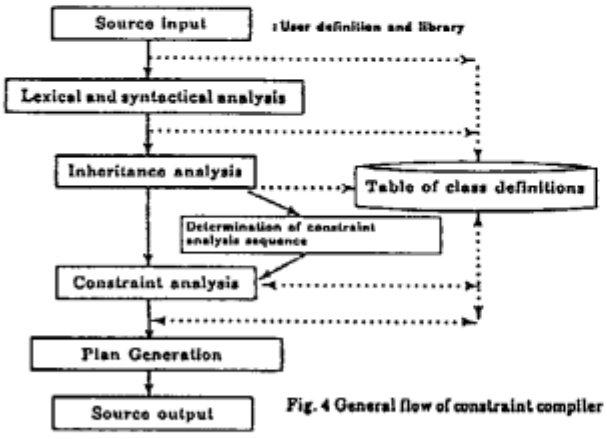


Fig. 4 General flow of constraint compiler

```
1  procedure constraint_analysis
2    begin
3      while (component = 'terminal')
4        data_flow_analysis_and_merge ;
5    end;
6    for V ∈ a list of ordered functional blocks do
7      data_flow_analysis_for_block (V) ;
8    for V ∈ determined tree description of the design object do
9      top_down_analysis(V) ;
10 end;
```

Fig. 5 General algorithm of constraint analysis

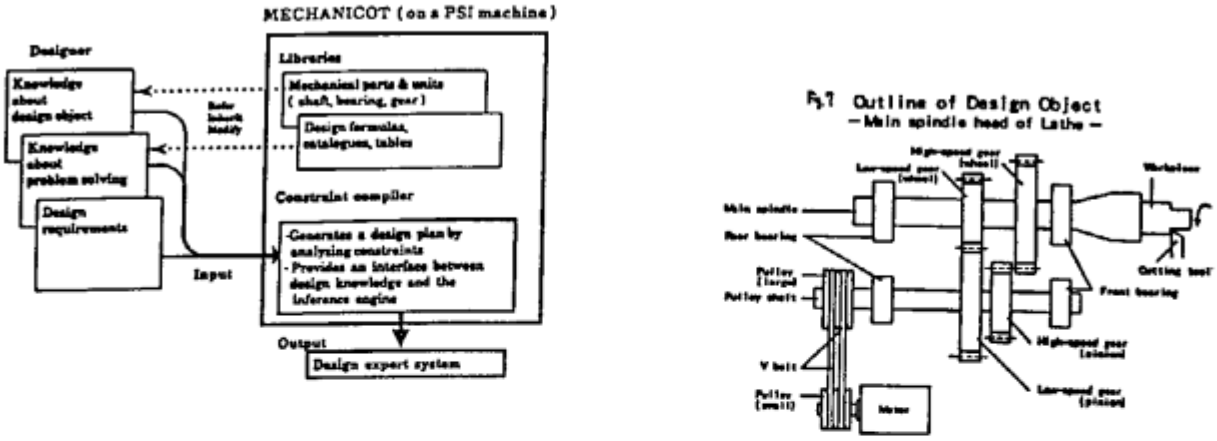
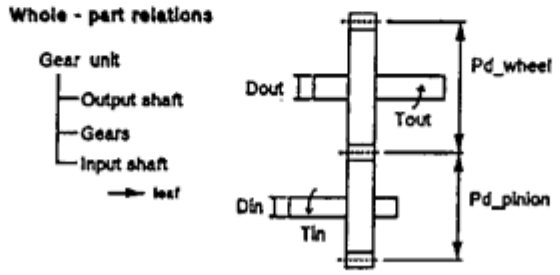


Fig. 6 Overview of MECHANICOT



Parameters

Input

Tin, Gin, θ_{in}, Rin, Tout, Gout, θ_{out}

Output

Rg, Pd_wheel, Pd_pinion, Din, Dout, Rout

Fig. 8 Schematic description of a gear unit

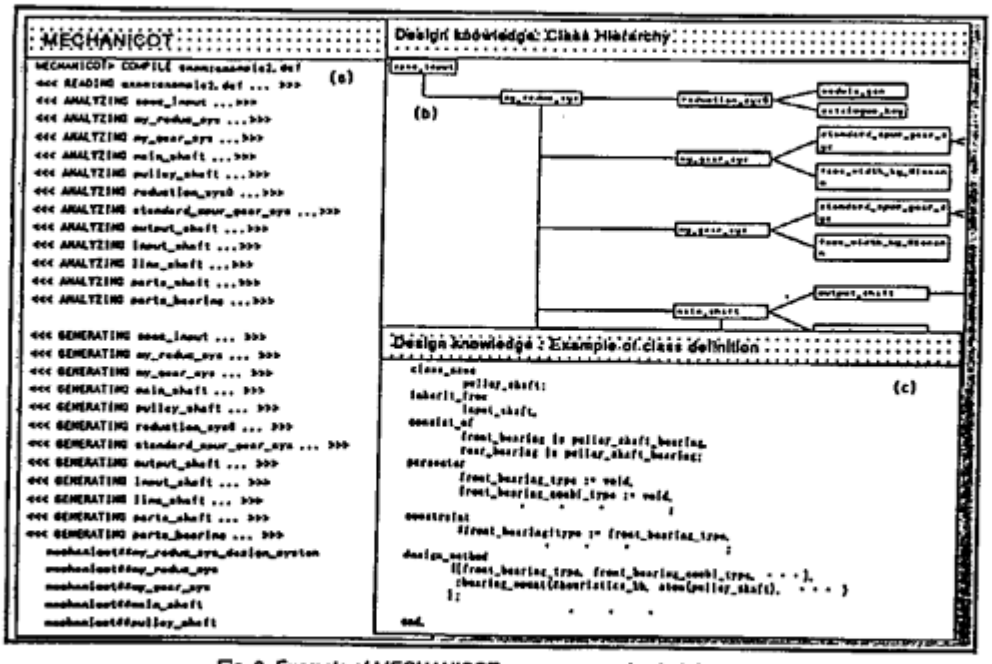


Fig. 9 Example of MECHANICOT system appearing in PSI window

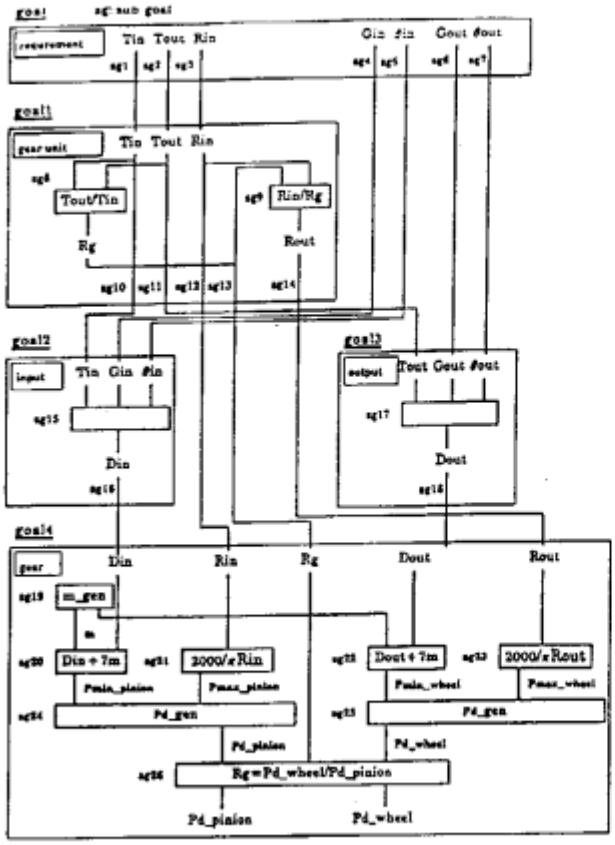


Fig. 10 Constraint Network Description

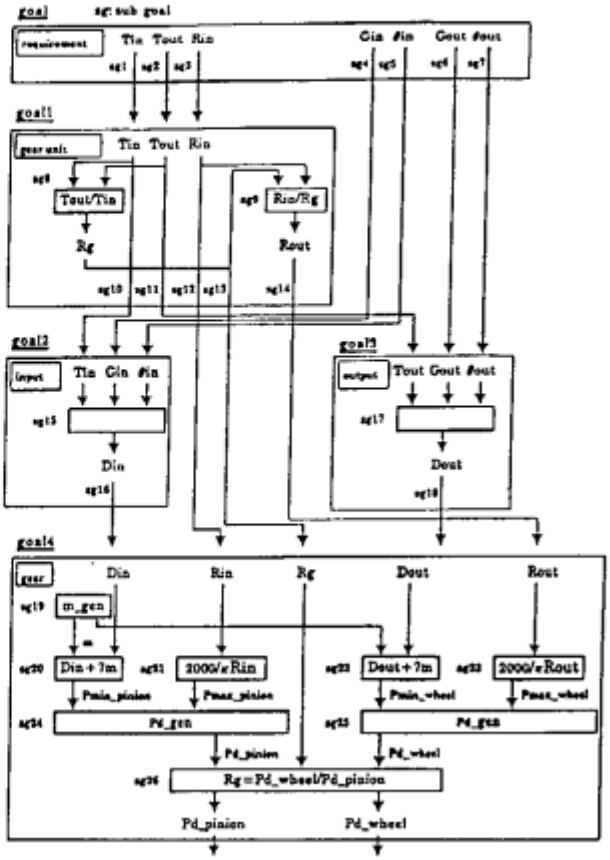


Fig. 11 Generated Design Plan

Fig. 12 (a) Input design specification

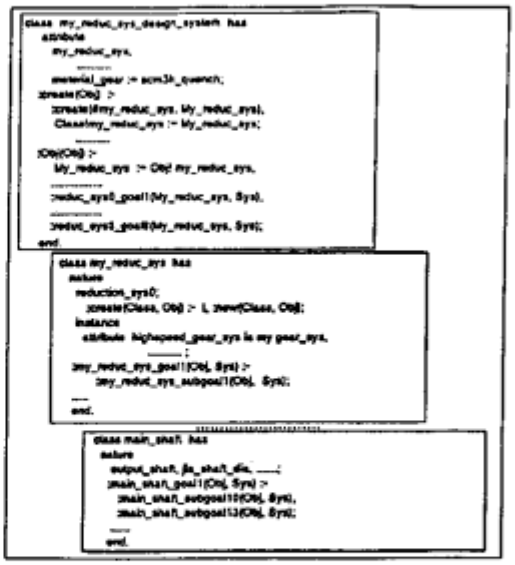


Fig. 12 (b) Design plan written in ESP code

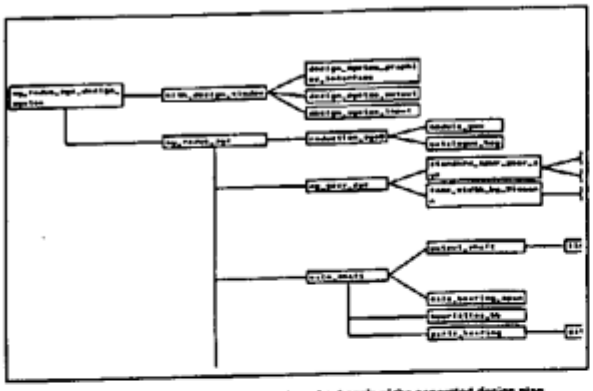


Fig. 13 Hierarchical relationships between goals and subgoals of the generated design plan

Table 1 Example of input parameters of a main spindle head design

Input parameters (design requirements)		Example of values
Cutting capacity	Workpiece material	S45C
	Tool material	Special hardness
	Workpiece diameter (max.)	165.0 [mm]
	Main_spindle speed (max.)	6000 [rpm]
	Cutting depth (max.)	2.0 [mm]
	Feeding speed (max.)	0.5 [mmpr]
	Drill diameter (max.)	40.0 [mm]
	Drill speed (max.)	30.0 [mpm]
Evaluation	Life of bearings	30000 [hours]

Note

mm: *millimeters*, rpm: *revolutions per minute*,

mmpr: *millimeters per revolution*, mpm: *meters per minute*.

Table 2 Example of output (design) parameters of a main spindle head design

Output (design) parameters	
Determined by calculation	Main_spindle diameter
	Pulley_shaft diameter
	Gears & pulleys ratio
	Number of gear teeth
	Gears pitch diameters
Result of previous design	Bearing mount type
	Bearing span
Search from catalogs or tables	Bearing part number
	Motor part number
	V_belt part number
	Pulleys part number

Table 3 The number of constraints used in the plan generation

Constraint		*C*	*DM*	*G*	*T*	*F*
Problem	Gear unit	75	28	3	2	2
	Spindle head	192	106	11	9	2

Note

C: *constraint*, DM: *design_method*, G: *generator*,

T: *tester*, F: *filter*.