TM-1270

A Constraint-Based Knowledge Compiler for Parametric Design Problem in Mechanical Engineering

by
Y. Nagai (Toshiba) & S. Terasaki (Matsushita)

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

Institute for New Generation Computer Technology

A Constraint-Based Knowledge Compiler for Parametric Design Problem in Mechanical Engineering *

Yasuo Nagai † † **Satoshi Terasaki** † ‡

†Institute for New Generation Computer Technology, Tokyo 108, Japan.

Keywords: knowledge compiler, knowledge compilation, parametric design, routine design, constraint, constraint satisfaction, mechanical design.

Summary

Knowledge compilation is a technique by which task-specific knowledge is represented in declarative form and by which procedures specific to the task domain using derived problem-solving mechanism effectively is generated. This paper describes a constraint-based knowledge compiler for parametric design problem in mechanical engineering. The purpose of our research is to realize tools allowing designers to build knowledge-based systems for parametric design in mechanical engineering simply by using a knowledge compilation approach. Our knowledge compilation approach seeks to automate 1) the process of producing knowledge-based design systems from higher level design specifications, and 2) the process of restructuring existing software systems to produce new systems that exhibit a) an increase in efficiency or usability, b) a change in representation level, and c) a reduction of reasoning. Consequently, we realize knowledge compilation that adopts the advantages of the constraint-based formalization of parametric design systems; we define this technique as constraint-based knowledge compilation. Constraint-based knowledge compilation automates the process of producing knowledge-based design systems for each design object, from input design specifications, by regarding 1) a design problem in mechanical engineering as parametric design, 2) an explicit declarative knowledge of this design as a constraint, and 3) the design process as a constraint satisfaction problem. The advantage of this technique is that it is easy to modify and reuse in order to realize problem-solving mechanisms (i.e. constraint satisfaction) for other design tasks. We have developed MECHANICOT, a tool which implements our constraint-based knowledge compilation technique for building knowledge-based systems for parametric design in mechanical engineering. We also demonstrate its effectiveness, especially constraint analysis and plan generation, on a mechanical design problem.

1 Introduction

In many design problems, the designer performs a design task for each design object. Many design systems have been developed and used in various fields to perform specialized design tasks for design objects, or to help designers to perform efficiently part of the design tasks. For example, domain-specific design systems have been developed and utilized to efficiently perform design tasks for well-defined and well-structured design problems [Nagasawa 87]. However, developing such systems is an expensive and time-consuming task. In addition, it is difficult to modify and extend these systems since it is not feasible to reuse them for other design tasks. To allow designers or end users to build specialized systems for each design object (domain-specific design systems), it is necessary to provide a methodology that could help automate the construction of new design systems by combining modularized software components that are easy to maintain and extend [Goel 91].

Knowledge compilation is a common term in artificial intelligence research on problem solving, model-based reasoning, machine learning, software engineering, and automatic programming [Goel 91]. We shall adopt a knowledge compilation approach, which is expected to provide a methodology for the construction of special-purpose design systems for each design object by integrating a design process model with a design object model. The purpose of our research is to realize tools allowing designers to build knowledge-based systems for

*This research was carried out mainly at the Fifth Research Laboratory (Intermediate stage from 1985 to 1989) of ICOT.

†Systems & Software Engineering Laboratory, Research & Development Center, Toshiba Corp., Kanagawa 210, Japan.

‡Tokyo Information Systems Laboratory, AV & CC Systems Research and Development Center, Matsushita Electric Industrial Co., Ltd., Tokyo 140, Japan.

parametric design in mechanical engineering simply by using a knowledge compilation approach. Our knowledge compilation approach seeks to automate the process of producing knowledge-based design systems from higher level design specifications, and the process of automatically restructuring existing software systems to produce new design systems that exhibit 1) an increase in efficiency or usability, 2) a change in representation level, and 3) a reduction of reasoning. This approach can facilitate knowledge reuse because design knowledge such as knowledge about the design object and knowledge about the problem solving methods can be encoded separately and explicitly and can be represented in a general form. Furthermore, it can also help knowledge acquisition to the extent that the it is possible for users to construct new knowledge-based systems easily [Mizoguchi 88].

Previous work on the application of knowledge compilation techniques to design problems focused primarily on design plan generation methods and consistency maintenance methods of knowledge-based design systems. Araya & Mittal [Araya 87] presented a method by which design plans can be automatically generated by compiling knowledge about artifacts, problem-solving heuristics, and characteristics of specific problems. Serrano & Gossard [Serrano 90] presented a constraint-based environment for Mechanical Computer Aided Design, where methods for maintaining consistency are provided to designers with assistance during the early stages of design and help to close the gap between novice and experienced designers. Furthermore, Feldman [Feldman 88] presented a constraint-based knowledge compiler, which is used to optimize constraint propagation and to improve the efficiency of constraint propagation for a constraint language embedded within a frame system by performing a dependency analysis of constraint networks, as work on a constraint-based knowledge compilation technique.

We treat parametric design in mechanical engineering, where both the knowledge and problem-solving techniques are known in advance and the problem space needed to find solutions is reduced. In such knowledge, relations between the attributes of design objects as well as the relations between the attributes within a design object may be represented using symbolic descriptions such as mathematical formulas. Common knowledge in handbooks and other sources of engineering knowledge usually appear as design formulas or relations which are combinations of other engineering knowledge rewritten for the designer's benefit. We need to represent this knowledge in an explicit, declarative, and domain-independent form because it should be easy to modify and reuse this knowledge for the realization of other design tasks. Thus, considering a constraint-based formalization that regards this knowledge as constraints, much of the engineering design process can be viewed as the identification, formulation, and satisfaction of constraints. We will realize knowledge compilation that adopts the advantages of the constraint-based formalization of parametric design systems; we define this technique as constraint-based knowledge compilation.

In Section 2, we describe the target architecture and constraint-based formalization of knowledge-based systems for parametric design in mechanical engineering. Section 3 describes our constraint-based knowledge compiler. The current status and future research are described in Section 4. Finally, we conclude with Section 5.

2 Target Architecture of Knowledge-Based System for Parametric Design in Mechanical Engineering

2.1 Parametric Design in Mechanical Engineering

Parametric design is subclasses of routine design. Routine design is a class of design problem where both the knowledge and problem-solving techniques are known in advance so the problem space needed to find solutions is reduced [Chandrasekaran 90]. Parametric design determines or optimizes the structural parameters of the design object, when its structure is fixed and standard methods of designing various components are known.

Fig. 1 shows a model of the design process for parametric design. First, the design plan templates are searched for knowledge base according to a design specification. These templates represent the structure or configuration of the design objects obtained from previous designs corresponding to the input specification.

Next, using the design plan templates, the structure of the design object is determined by searching the struc-

tural model knowledge base for an appropriate design style, or by configuring or combining predefined components. The engineering model is then determined based on the structure of the design object. The engineering model of the design object consists of the analysis and evaluation knowledge about the design object (e.g. design formulas) and public knowledge (e.g. parts catalogs and the knowledge in textbooks). Refinement of the engineering model corresponds to constraint satisfaction [Mackworth 92] or optimization. It can be regarded as either the search for attribute parameters so that constraints can be satisfied, or as the optimization of the attribute parameters of the structures of the design object. If possible, during optimization, the structure of the design object is transformed or modified locally without changes to the input design specifications.

In evaluation of engineering model, the engineering model is analyzed and checked whether it satisfies the design specifications or not. In this case, there are two methods: domain-specific calculations and simulation.

In modification of engineering model, information about the failure of evaluation is used to modify the engineering model without changing the structural model.

In evaluation of structural model, the structural model is evaluated according to some evaluation criteria and the causes of the design's failure are analyzed.

In modification of structural model, information about the failure of a candidate design is used to modify the design to better match the specifications.

2.2 Constraint-Based Formalization of Knowledge-Based Systems for Parametric Design

In parametric design, first the structural model of the design object is determined by searching a knowledge base for a design style template according to given design specification. Therefore, the problem space of a parametric design problem is known in advance and is restricted because a class of this design problem is determined by assuming the structure of the design object. We can regard a parametric design problem as a problem with a sufficient solution space where constraint satisfaction [Mackworth 92] can be done without excessive search. This problem is to find a consistent set of parameter values, in which case numeric or symbolic optimization techniques and constraint satisfaction techniques are effective. It can be viewed as a search problem, an algebraic equation-solving problem, or an optimization problem. Thus, in constraint-based formulations of parametric design, the solution may lie in a space determined by a set of constraints [Chandrasekaran 90]. The advantages of constraint-based formalization are as follows: A constraint is a more sophisticated representation which is effective for building knowledge-based design systems because it is a declarative representation. In addition, a constraint can be used in many ways; it can be added incrementally; it can be expressed in a variety of ways; it can be used to solve problems efficiently; it can be used to support local maintenance; and it can also provide a global view of the search space. Furthermore, constraint-based techniques have been shown to improve problem-solving capabilities. 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, such as checking of a solution or a partial solution, generation of a partial solution, modification of a solution, propagation of decisions to modify future decisions, and computation of search space metrics.

Problem-solving mechanisms can be realized to solve the constraint satisfaction problem, the optimization problem, or the equation-solving problem by applying various solution methods. These solution methods involve search methods such as constraint propagation, generate & test, refinement, or least commitment [Nagai 88, Nagai 89a], mathematical programming methods such as linear programming and integer programming, and equation-solving methods. Because our target design is a parametric design and can be regarded as a constraint satisfaction problem, the search for the structural parameters is executed through a constraint network [Dechter 92] derived from the determined structure of the design object.

Our problem-solving mechanism for a knowledge-based system for parametric design is based on a generate & test method and is composed of the following primitives: the generator, the propagator, the tester, and the filter shown in Fig. 2 [Nagai 88, Nagai 89a]. The controller controls a hierarchical generate & test by selecting

and executing the problem-solving primitives according to the structure of the constraint network. The generator assigns values to parameters or maps functional blocks or components to physical blocks or components. This generator can take either continuous or discrete values. In the former case, the generator assigns parameters of the attributes by local modification based on the predetermined components. In the latter case, the generator assigns parameters by retrieving the standard parts for implementing components from a catalog using a table look-up method. The propagator assigns parameters by actively evaluating functions or constraints, and propagating and solving constraints. The tester checks the constraints; testing is also considered as the passive handling of constraints. In general, inequality constraints can be regarded as the tester, but in some contexts, they can also be considered as equality constraint. The filter produces, as output, values that satisfy the constraints, by removing the unsatisfied values from input values or adjusting input values to standard values.

Finally, knowledge-based design systems having an inference engine, where a problem-solving mechanism for performing a specialized design task is realized using problem-solving primitives and embedded, are generated using knowledge-based system building tools or programming languages.

3 Constraint-Based Knowledge Compiler for Parametric Design Problems

In this section, we describe a constraint-based knowledge compiler which adopts our constraint-based knowledge compilation technique.

3.1 Overview of Constraint-Based Knowledge Compiler

The general flow of our constraint-based knowledge compiler is shown in Fig. 3. The compiler contains four main procedures: retrieval of design plan template, determination of structural model of design object, determination of engineering model, and constraint analysis and plan generation.

Inputs to the compiler are a design specification, which includes the functional description and constraints such as performance and a resource limitation in the form of a parametric description. The compiler searches a knowledge base for a design plan template according to given design specification. It determines the structure of the design object by searching the knowledge base for the structural model of design object (e.g. the instances of the configuration and mechanism of the design object) using this design plan template.

Next, this compiler determines the engineering model (e.g. the analysis and evaluation methods) corresponding to this structure by searching the knowledge base for knowledge about the design object and knowledge about problem-solving. Consequently, the compiler is able to generate a set of constraints since most engineering models can be expressed in terms of constructs such as algebraic equations and data input descriptors. This set of constraints is regarded as a constraint network involving a set of parameters and their domains of finite or infinite values [Mackworth 92].

3.2 Constraint Analysis and Plan Generation

Fig. 4 shows the general flow of constraint analysis and plan generation. In order to deal with constraint analysis and plan generation for networks with a hierarchical structure, five main procedures are used. These procedures are: *inheritance analysis of design object* (by generation of class definition tables), *determination of a dataflow analysis sequence*, two *dataflow analyses*, and *plan generation* (by both generation and update of tables).

Handling the constraint network in constraint analysis and plan generation is difficult because the structure of the network is quite complicated for a complex design object. Therefore it is necessary to structure the network with a hierarchical level of abstraction and to combine subnetworks, instead of handling the whole network as a flat structure.

(a) Constraint Analysis

Effective utilization of the structural information of the problem space extracted using constraint analysis will lead to improvements in the solution methods for constraint satisfaction (e.g. constraint propagation), constraint-directed search (e.g. dependency-directed backtrack search), and hierarchical generate & test [Mackworth 92, Dechter 92]. In the process of constraint propagation, local and non-local (global) propa-

gations must handled [Sussman 80]. In local propagation, known values are propagated along the arcs to the nodes of the constraint network by using local information [Sussman 80]. When there is a cyclic dependency in the constraint network, the problem can not solved by using local propagation only. In non-local (global) propagation, global information such as an equation-solving technique, is required to deduce the values of the parameters. In our constraint analysis, the analysis for the local constraint propagation and constraint-directed search mechanism is realized.

Next, each procedure of constraint analysis is described. Because our design knowledge is represented using an object-oriented language, flexible modification and reusability of this knowledge can be achieved by an inheritance mechanism. *Inheritance analysis* analyzes inheritance relations between functional blocks and between components in design knowledge. In *dataflow analysis sequence determination*, the part-whole relations and inheritance relations of the design object are analyzed and a directed-acyclic graph (DAG) is generated using the results of the analysis. The dataflow analysis sequence is determined according to the DAG graph and it is represented as a tree description composed of part-whole relations and inheritance relations of the design object. The two dataflow analyses (phase one and phase two) are executed according to this tree description. These analyses execute a procedure similar to that of the dataflow analysis in the optimization phase of a conventional compiler [Aho 77]. Phase one corresponds to the local optimization, i.e. optimization of a basic block of the compiler, and phase two corresponds to the global optimization, i.e. global rearrangement of the dataflow graph generated during phase one.

Fig. 5 (a) shows the main algorithm of the dataflow analysis composed of phase one (line 3-8) and phase two (line 9-11). Phase one proceeds from the bottom up: it performs dataflow analysis [Aho 77] and grouping of analyzed dataflow information for each component as shown in Fig. 5 (b) and each functional block as shown in Fig. 5 (c). In the dataflow analysis for each component, constraints are analyzed according to the problem-solving types assigned to each constraint and the dataflow information is extracted. Thus, the dataflow graph is built for the constraints by serializing and grouping the extracted dataflow information (Fig. 5 (e)). In the dataflow analysis for functional block, the same dataflow analysis as one for each component is executed using the result of the dataflow information of each component as shown in Fig. 5 (c). Concretely, dataflow analysis begins at the lowest-level node of the tree description of the design object and proceeds towards the highest-level node of the tree. If there are inheritance relations, the constraints are not processed along the node hierarchy between parent nodes and children nodes, but are treated as a flat set of constraints included in both parent nodes and their children nodes.

Phase two starts from the root and proceeds from the top down towards the leaves of the tree as shown in Fig. 5 (d): the dependencies among the constraints are determined by re-analyzing dataflow information extracted from constraints among functional blocks, and between functional blocks and components. This dataflow analysis is equivalent to the inter-procedural optimization of a conventional compiler [Aho 77]. Finally the dependencies among the constraints of the whole system are determined.

(b) Plan Generation

After dependencies among the component and functional block constraints are obtained by constraint analysis, plan generation proceeds towards the higher levels of the hierarchy of the design object using the results of the constraint analysis as shown in Fig. 5 (f) and (g). Thus, a design plan is generated by grouping the dependencies analyzed above and re-analyzing them. Finally, a problem-solving mechanism based on the problem-solving primitives suitable to the given problem is generated by assuming tasks necessary to solve the problem from the analyzed dependencies.

3.3 MECHANICOT

To help designers build a knowledge-based system for parametric design in mechanical engineering, we propose a tool MECHANICOT in which design knowledge is treated as constraints and this design process is treated as a constraint satisfaction problem. MECHANICOT is a constraint-based knowledge compiler with libraries for describing design knowledge. The designer inputs a design specification, knowledge about the design object,

and knowledge about problem-solving to MECHANICOT, by specifying system libraries of knowledge bases, or by modifying these libraries. MECHANICOT analyzes parameters, constraints, and structures of a design object, extracts the dataflow information, generates a design plan by analyzing dependencies among constraints and parameters, and provides an interface between the design knowledge and the inference engine. The output from MECHANICOT is a specialized knowledge-based design system including designers' heuristics. As a result of an execution of this generated system, a design calculation is performed and the functional machine unit with dimensions is described on the graphic display.

Fig. 6 shows the architecture of MECHANICOT, the tool we are developing. The libraries consist of knowledge about the design object which includes functional blocks and components (e.g., shafts, bearings, and gears) including design formulas, catalogs, and tables and knowledge about problem-solving.

Design knowledge, such as knowledge about the design object and knowledge about the problem-solving, is encoded separately and explicitly so that knowledge reuse can be facilitated. Knowledge about the design object is represented in a modularized form; i.e. design plan templates, functional blocks, and components are described as class modules. A class hierarchy represents inheritance relations (*inherit_from*) and part-whole relations (*consist_of*) among classes. This knowledge consists of design parameters, part-whole or inheritance relations, and constraints. Constraints include structural constraints, design formulas, and relationships among design parameters. Structural constraints represent relationships between components, between functional blocks and components, and between functional blocks; these relationships are determined according to the part-whole relations of the design object.

Knowledge about problem-solving describes solution methods to realize problem-solving mechanisms for design tasks using an extension of generate & test. In MECHANICOT, such knowledge is classified into four problem-solving primitives such as the generator, tester, design method, and filter shown in Table 1, according to the problem-solving types. The design method represents the functions or methods, and search methods from catalogs and tables which have no alternatives, while the generator represents the functions and search methods from catalogs or tables which have alternatives such as inequalities. The generator is used for restricting the search space. The tester represents equalities and inequalities used for testing solutions from the generators or for evaluating solutions. The filter is used for adjusting solutions to standard values.

Generally several alternative values may be generated for a parameter so that constraints can be satisfied because many constraints are represented as multidirectional relations and no notions such as assignment, procedure, or control flow are introduced. Since there are several possible solution methods, it may be efficient or inefficient to solve the problem depending on the solution method used, or it may be difficult to solve the problem enough using only given constraints.

MECHANICOT uses a dataflow implementation of constraints [Borning 81, Sussman 80]. Each constraint is represented as a set of procedures that compute values for the remaining parameters and a procedural test for whether or not the constraint is satisfied. The former procedures are implemented using functions or methods and are given a problem-solving type *design_method*. The latter test is implemented using relations such as equalities and inequalities and it is given a problem-solving type *tester*. In addition, problem-solving types such as *generator*, *constraint*, and *filter* can be assigned to each constraint. The problem-solving type *constraint* is assigned to the structural constraint and a direction of this constraint can be determined using input information. For example, the constraint $a = b + c$ has three *design_method* type constraints $a := b + c$, $b := a - c$, and $c := a - b$ (where $:=$ means an assignment) and the *tester* type constraint $a = b + c$ (where parameters a, b,and c have a value), if necessary. Consequently problem-solving mechanisms are realized according to these types through a constraint analysis and plan generation.

3.4 Example of Constraint-Based Knowledge Compilation Using MECHANICOT

First, we will explain in detail MECHANICOT's constraint-based knowledge compilation process, using a gear unit as an example of the design object. Next, we will describe a result of the compilation using the main

spindle head of a lathe.

3.4.1 The Gear Unit Design Problem

In this section, we use, as an example, the gear unit. Fig. 7 (a) shows the gear unit, a subsystem of the power unit of the main spindle head of a lathe transmission unit [Inoue 88]. Fig. 7 (b) shows the part-whole relation of the gear unit composed of the output shaft, input shaft, and gears. The gear unit design is a typical example of parametric design. It can be considered as a problem of determining the design parameters so that the specification parameters, performance parameters, and constraints are satisfied when the structure of the design object is assumed to be fixed. In this problem, it is necessary to consider that there are two strategies for the implementation of the physical components using standard parts and data obtained by searching catalog and tables, and non-standard parts obtained by calculating design formulas and determining design parameters and their combinations.

Input parameters and output parameters are shown in Fig. 7 (c). Input parameters are the twisting moment of the input shaft T_{in}, the shearing strength G_{in}, the tolerant torsion angle θ_{in}, the number of input revolutions R_{in}, the twisting moment of output shaft T_{out}, the shearing strength G_{out}, and the tolerant torsion angle θ_{out}. Output parameters are the gear ratio R_g, the pitch diameter of gears *Pd_wheel* and *Pd_pinion*, the shaft diameters D_{in} and D_{out}, and the number of output revolutions R_{out}.

Fig. 8 shows constraints required to design a gear unit. Based on problem-solving types assigned to these constraints, a problem-solving mechanism is realized so that these constraints can be satisfied. Among the constraints for each gear design, equation (1) describes the relationship among a gear module m, a number of gear teeth Z, and a gear pitch diameter Pd. In this case, because the module m takes a discrete value set, the problem-solving type *generator* is assigned to the parameter m of this equation. The equation can be interpreted as a procedure with the generator. Inequality (2) means that the minimum value of the gear pitch diameter is obtained from the values of the shaft diameter *Ds* and the value of the module m. Inequality (3) means that the maximum value of the gear pitch diameter is determined from the value of shaft revolutions *Rpm*. In MECHANICOT, inequalities are transformed to equations with generators. Based on the problem-solving type *design_method* assigned to these equations, these transformed equations are regarded as procedures (i.e. assignment statements) and handled by generate & test problem-solving mechanism. For example, both inequalities (2) and (3) for wheel and pinion gears are required and they are transformed to the procedures: $Pd_{min}\_wheel := D_{in}+7m$, $Pd_{min}\_pinion := D_{out}+7m$, $Pd_{max}\_wheel := 2000/\pi R_{in}$, $Pd_{max}\_pinion := 2000/\pi R_{out}$ and the two generators: a generator for a pinion gear that generates positive integer *Pd_pinion* by increments 1 from the minimum value Pd_{min}_pinion and the maximum value Pd_{max}_pinion and a generator for a wheel gear that generates positive integer *Pd_wheel* by increments 1 from the minimum value Pd_{min}_wheel and the maximum value Pd_{max}_wheel. Equation (4) describes a relationship among the gear pitch diameters *Pd_wheel* and *Pd_pinion*, and the gear ratio *Rg* for a pair of gears. The problem-solving type *tester* is assigned to this constraint. Then the constraint is regarded as a tester which tests the gear ratio R_g. Equation (5) is the design formula representing a relation with parameters twisting moment T, tolerant torsion angle θ, shearing strength G, and shaft diameter D. Based on the assigned problem-solving type *design_method*, a procedure is selected from a set of procedures using methods or functions corresponding to the equation. The equation is regarded as a procedure. Equation (6) describes a structural constraint on this unit: the gear hole diameter *Dwheel* must be equal to output shaft diameter *Dout* in order to assemble the gear unit. Based on the problem-solving type *constraint* assigned to this constraint, dataflow information can be analyzed using given input information.

3.4.2 MECHANICOT's Solution

We explain MECHANICOT's constraint analysis and design plan generation using the gear unit as an example.

The user inputs design specification including design parameters shown in Fig. 9 (a) to MECHANICOT. MECHANICOT determines the structure of the gear unit. After inheritance analysis and constraint analysis sequence determination, MECHANICOT begins constraint analysis. First, it searches the knowledge base

for knowledge about functional blocks and components shown in Fig. 9 (b); i.e. it determines the engineering model according to the determined structure. This collected knowledge can thus constitute a set of constraints. Fig. 10 shows a constraint network corresponding to this constraint set.

Next, dataflow analysis consists of two phases. In phase one, dataflow analysis [Aho 77] is executed to the constraints of the *input shaft*, *output shaft*, and *gears* (a pair of gears) that correspond to the leaves of the tree description shown in Fig. 7 (b) (i.e. *input shaft* → *output shaft* → *gears* → *gear unit*). In this case, when the problem-solving type *design_method* is assigned to the constraint, an appropriate procedure using functions or methods corresponding to the constraint to be analyzed is selected from a set of procedures stored as knowledge about the design object. For example, if the constraint $a = b + c$ has three *design_method* type constraints $a := b+c$, $b := a-c$, and $c := a-b$ as knowledge about the design object, an appropriate constraint is selected from them and is regarded as a procedure. When the problem-solving type *generator* is assigned to a constraint, the constraint is regarded as a generator whose input and output parameters are determined. When the problem-solving type *filter* is assigned to the constraint, the constraint is regarded as a filter. When the problem-solving type *tester* is assigned to a constraint, the constraint is regarded as a tester. In the case where there is only one parameter of this constraint, the generator sharing the same parameter is modified so that it enumerates only values that satisfy the tester. In the design method for calculating an input shaft diameter, dataflow information is extracted from functions or methods corresponding to the constraint and input parameters such as the twisting moment T_{in}, the shearing strength G_{in}, and the torsion angle θ_{in}. As a result, the input shaft diameter D_{in} can be considered as the output parameter. Then, traversing toward the root of the tree, dataflow analysis is done in the functional block (*gear unit*) in the upper part of the tree description. After that, we perform dataflow analysis for the functional block *gear unit* and its components the *input shaft*, *output shaft*, and *pair of gears*. The dataflow analysis terminates when the root of the tree is reached.

In phase two, the dataflow analysis is executed from the root of the tree to the leaves in Fig. 7 (b) (i.e. *gear unit* → *input shaft* → *output shaft* → *gears*). In this phase, dependencies between the functional block (*gear unit*) and the components (*input shaft*, *output shaft*, and *gears*), and dependencies between the components themselves, are re-analyzed using dataflow information; finally the dependencies of the whole system are analyzed.

In plan generation, the problem-solving primitives shown in Table 1 are integrated into subgoals according to the problem-solving type (*generator*, *tester*, *design_method*, *constraint*, and *filter*) assigned to each constraint so that the name of each subgoal is unique. Subgoals are arranged and integrated into goals based on the input-output dependencies of the parameters generated by dataflow analysis shown in Fig. 11. Unique 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 class that is one level higher than the class in which the goal is included. When generator & tester loops are included in execution statements, a generator corresponding to a tester is found by analyzing dependencies of constraints and is moved to execute the generate and test loop efficiently. Execution of these statements can be considered equivalent to the realization of test movement [Dietterich 86]. Test movement regresses tests back into the generator to achieve early pruning without affecting the correctness of the problem solver.

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. 7 (c); the structural relationships between the functional blocks and components and the design methods for the functional blocks and components are formalized and given in advance as the structural and engineering model descriptions of the design object in the knowledge base. From knowledge about the design object and knowledge about problem-solving derived from the input design specifications shown in Fig. 9, we generate a design plan using constraint-based knowledge compilation and finally obtain a plan written in the Extended Self-contained Prolog (ESP)

[Chikayama 84] code shown in Fig. 12. Fig. 13 shows an example of the MECHANICOT system appearing in the Personal Sequential Inference machine (PSI) II [Taki 84] windows: (a) shows the compilation process of a gear unit design problem. (b) shows the hierarchy of design knowledge (design object), and (c) shows the class definition of design knowledge.

3.4.3 The Main Spindle Head Design Problem

As another example, we examine the main spindle head design problem, which is a large and practical parametric design problem in mechanical engineering [Inoue 88]. The main spindle head of a lathe, shown in Fig. 14, 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 problem is to determine the dimensions of each part and find each part number by searching catalogs.

Table 2 shows an example of two input parameters (design specifications) composed of cutting capacity requirements and evaluation criterion and an example of output (design) parameters. The default values of input parameters are shown at the right side of the table 1. Table 3 shows the number of constraints used in the constraint analysis of MECHANICOT. Although the manual construction of design plans (i.e. knowledge-based design systems) for this problem takes a few hours by search and choice of parts catalogs and tables, design formulas, or physical laws, MECHANICOT compiles this problem constraints derived from the input design specification and generates a design plan in roughly 10 minutes CPU time on a PSI II machine.

4 Current Status and Future Research

The first implementation of a constraint-based knowledge compiler MECHANICOT is carried out using the ESP language on the PSI II machine.

MECHANICOT does not provide a friendly user interface, i.e. one where the designer can give design specifications and design knowledge in the form of a schematic description as an input. In other words, it receives design specifications, design object representations written in an ESP-like object-oriented language, and problem-solving primitives executing solution methods as input, and generates the design plan written in ESP as output. The execution mechanism of the generated design plan is realized using ESP's inference mechanisms, such as unification and backtracking.

To analyze both local and non-local constraints in constraint analysis, it is necessary to separate the constraint network into parts that can be processed by local propagation (tree description) and parts that require non-local propagation (loop description) [Henley 73]. Consequently, each description should be dealt with by using the appropriate problem solver for constraint handling to split up the constraint network thereby reducing the search space of the problem. However, since only local constraints are handled in our constraint analysis, problem-solving types such as *generator* and *tester* must be assigned to constraints so that the role of a constraint in the problem-solving mechanism is fixed. In future, the problem-solvers that can handle the local and the non-local constraint propagations (i.e. constraint solvers) are required. Furthermore, we handle only obligatory (hard) and static constraints [Nagai 88]. The handling of suggestive (soft) constraints [Nagai 88] 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 of constraints, including constraint relaxation [Nagai 88, Nagai 89a], are required to realize dynamic constraint handling.

When a constraint-based knowledge 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 these states. To detect overconstrained and underconstrained states, we must consider a structural analysis method that analyzes dependencies between parameters of constraints (topology information of the constraint network) by considering the constraint network as a bipartite graph [Nagai 91].

For the future, we will consider extending the constraint analysis to utilize the structure of the problem space effectively and to improve constraint propagation

and constraint-directed search. These extensions include 1) incremental analysis to handle constraints that change dynamically, 2) handling of the different priorities of the constraints, and 3) dynamic interpretation of the direction of an information flow of constraints. Furthermore, plan generation requires the scheduling of goals and subgoals according to the rough prediction of the necessary cost of problem-solving gained during constraint analysis [Nagai 89a].

5 Conclusion

In this paper, we discussed a constraint-based knowledge compiler for parametric design in mechanical engineering. We regarded the design process of this parametric design as a constraint satisfaction problem and assumed a problem-solving architecture of knowledge-based design systems using an extension of the generate & test method.

We have demonstrated the constraint-based knowledge compilation technique on the functional machine units, a gear unit and a main spindle head of a lathe, by developing a knowledge-based system building tool MECHANICOT. MECHANICOT generates a design plan that embeds the constraint-incorporated problem-solving mechanisms realized using problem-solving primitives and that solves the parametric design problem in mechanical engineering. MECHANICOT can be also regarded as a software CAD tool.

The constraint-based knowledge compilation technique will be expected to be applied to various routine design tasks such as configuration tasks and circuit design tasks because it is described in a general form similar to a conventional compiler and reused so that design plan can be generated easily for other design tasks. This technique will lead to an improvement of the development of knowledge-based design systems and an enhancement of the productivity of designers or end users.

Acknowledgments

I would like to express thanks to Mr. Y.Fujii (NTT Corp.), Mr. K.Ikoma (NTT Data Corp.) and the other members of the Fifth Research laboratory (Intermediate stage) Mr. T.Yokoyama (Hitachi Corp.), Dr. K.Inoue, Mr. E.Horiuchi (MEL) and Dr. H.Taki (Mitsubishi Corp.) for their helpful comments. I would also like to thank Mr. M.Hoshi (JIPDEC) for implementation of the MECHANICOT system and Prof. I.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. R.Hasegawa of the ICOT Research Laboratories for helpful comments and suggestions. Finally, I would like to express special thanks to Dr. K.Fuchi, Director of ICOT Research Center, who has given me the opportunity to carry out research in the Fifth Generation Computer Systems Project.

References

[Aho 77] Aho, A. V. and Ullman, J. D.: *Principles of Compiler Design*, Addison-Wesley Publishing Company Inc. (1977).

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

[Borning 81] Borning, A.: The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory. *ACM Trans. on Programming Languages and Systems*, Vol. 3, No. 4, pp.353-387, October (1981).

[Chandrasekaran 90] Chandrasekaran, B.: Design Problem Solving: A Task Analysis, *AI Magazine*, Vol.11, No.4, pp.59-71, Winter (1990).

[Chikayama 84] Chikayama, T.: Unique Features of ESP, *Proc. of International Conference on Fifth Generation Computer Systems*, pp.292-298 (1984).

[Dechter 92] Dechter, R.: Constraint Networks, *Encyclopedia of Artificial Intelligence, Second Edition*, Vol. 1, pp.276-285 (1992).

[Dietterich 86] Dietterich, T. and Bennett, J.: The Test Incorporation Theory of Problem Solving (Preliminary Report), The Workshop on Knowledge Compilation, AAAI (1986).

[Feldman 88] 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)*, Vol.1, pp.141-146 (1988).

[Goel 91] Goel, A.K. (ed): Knowledge Compilation : A Symposium, *IEEE EXPERT*, pp.71-93, April (1991).

[Henley 73] Henley, E. J. and Williams, R. A.: *Graph Theory In Modern Engineering, Computer Aided Design, Control, Optimization, Reliability Analysis*. Academic Press (1973).

[Inoue 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 - (in Japanese). *ICOT-Technical Memorandum* (1988).

[Mackworth 92] Mackworth, A. K.: Constraint Satisfaction, *Encyclopedia of Artificial Intelligence, Second Edition*, Vol. 1, pp.285-293 (1992).

[Mizoguchi 88] 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).

[Nagai 88] Nagai, Y., Terasaki, S., Yokoyama, T., and Taki, H.: Expert System Architecture for Design Tasks, *Proc. of Int'l Conf. on Fifth Generation Computer Systems 88*, pp.296-317, ICOT (1988).

[Nagai 89a] 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) , Vol. 4 No. 3, pp.297-303 (1989).

[Nagai 89b] Nagai, Y. and Terasaki, S.: Towards Constraint Analysis and Plan Generation of Constraint Compiler for Design Problems (in Japanese), *Proc. of the 3rd Annual Conf. of JSAI*, 11-43, pp.693-696 (1989).

[Nagai 91] Nagai, Y. and Ikoma, K.: Constraint Analysis and Plan Generation Based on Graph Theory (in Japanese), *Proc. of the 40th Annual Convention of IPSJ*, 1D-4, pp.218-219 (1990).

[Nagasawa 87] Nagasawa, I.: Design Expert System (in Japanese), *IPSJ*, Vol. 28, No. 2, pp.187-196 (1987).

[Serrano 90] Serrano, D. and Gossard, D.: Constraint management in MCAE, *Artificial Intelligence in Engineering: Design*, (Jero, J.S. (ed.)), pp.217-240, Elsevier (1990).

[Sussman 80] Sussman, G. J. and Steele, Jr. G. L.: CONSTRAINT - A Language for Expressing Almost-Hierarchical Descriptions, *Artificial Intelligence*, Vol. 14, pp.1-39 (1980).

[Taki 84] 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).

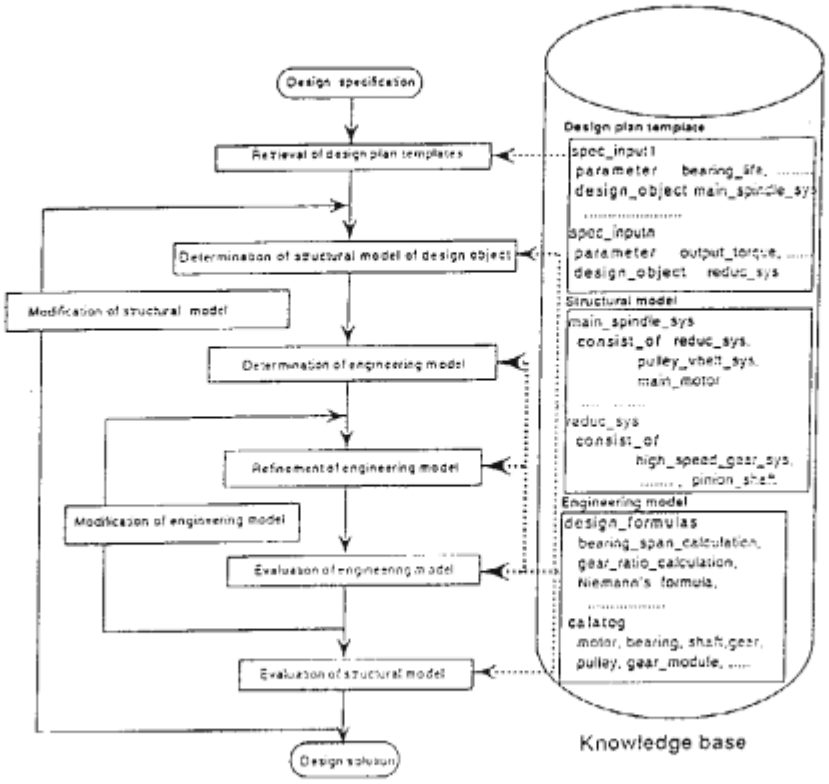


Fig. 1 Design process model of parametric design in mechanical engineering

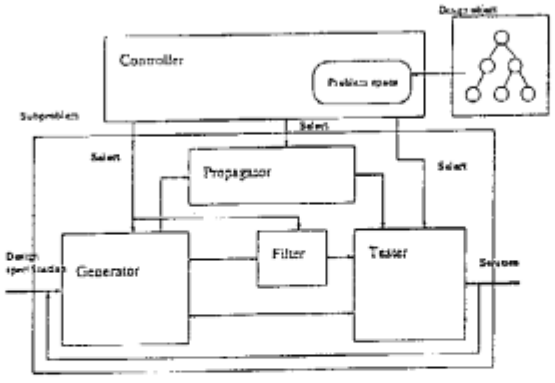


Fig. 2 Problem solving mechanism for parametric design

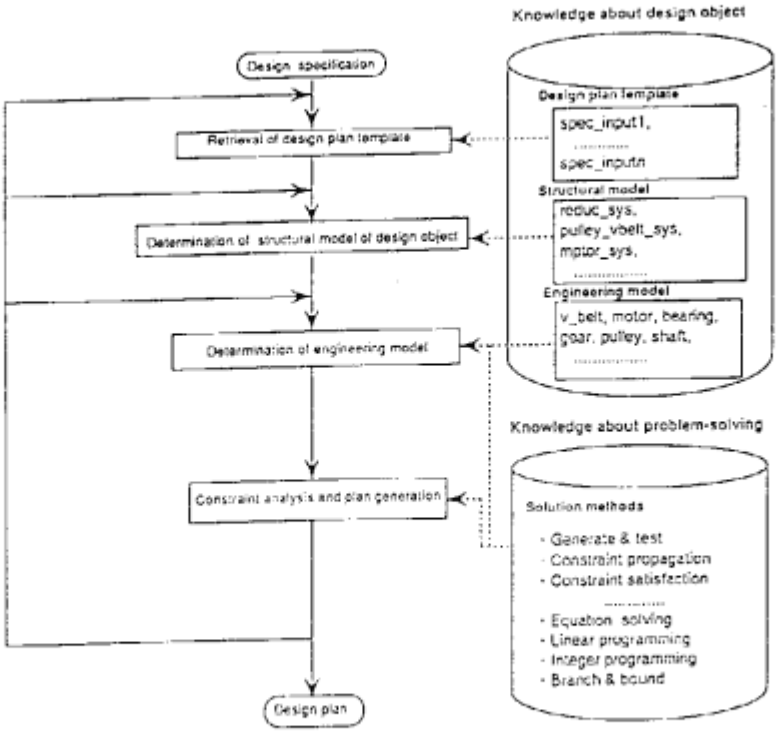


Fig. 3 General flow of a constraint-based knowledge compiler

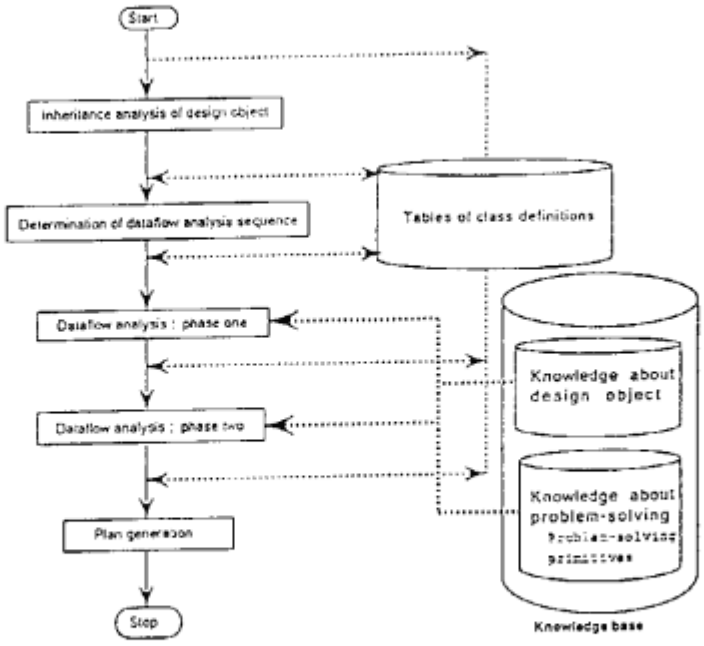


Fig. 4 General flow of constraint analysis and plan generation

```
1  procedure dataflow_analysis_main ;
2    begin
3      V ⟵ a list of components and functional blocks
             representing constraint analysis sequence
4      for V0 ∈ V, V is component
5        dataflow_analysis_and_grouping_for_component(V),
6      V1 ⟵ a list of ordered functional blocks;
7      for V2 ∈ a list V1 of ordered functional blocks do
8        dataflow_analysis_and_grouping_for_block(V2);
9      V3 ⟵ reverse(V1)
10     for V4 ∈ a reversed list V3 of V do
11       topdown_analysis(V4) ;
12   end;
```

(a) Main dataflow analysis algorithm

```
Inputs: Component V of the design object
Outputs: Returns "success" if analysis succeeds.

1  procedure dataflow_analysis_and_grouping_for_component(V);
2    begin
3      Get constraint statements Constrs from V;
4      Constr ⟵ a first element selected from Constrs;
5      while (there exists Constr)
6        Analyze a constraint statement Constr for a component according to its problem-solving type;
7        dataflow_analysis_and_grouping(Constr);
8        Constrs ⟵ Constrs \ Constr;
9        Constr ⟵ a first element selected from Constrs;
10     end;
11     return "success";
12   end;
```

(b) Dataflow analysis algorithm for the component

```
Inputs: Functional block V of the design object
Outputs: Returns "success" if analysis succeeds.

1  procedure dataflow_analysis_and_grouping_for_block(V);
2    begin
3      Get constraint statements Constrs from V;
4      Constr ⟵ a first element selected from Constrs;
5      while (there exists Constr)
6        Analyze a constraint statement Constr for a block according to its problem-solving type;
7        dataflow_analysis_and_grouping(Constr);
8        Constrs ⟵ Constrs \ Constr;
9        Constr ⟵ a first element selected from Constrs;
10     end;
11     V' ⟵ a set of components of functional block V;
12     for V'' ∈ V' do
13     dataflow_analysis_and_grouping_for_block(V'');
14     return "success";
15   end;
```

(c) Dataflow analysis algorithm for the functional block

```
Inputs: List V of components or functional blocks of the design object
Outputs: Generates ESP objects and returns "success" if generation succeeds.

1  procedure plan_generation(V) ;
2    begin
3     if V is empty
4      then  return "success";
5      else if V has a component or functional block CB then
6       begin
7        Translate a name of a block or functional block CB to ESP class;
8        Translate parameters of a table for CB to ESP attribute definition;
9        if V has a component or functional block with an inheritance relation CBI
10        then
11         begin
12          Translate CBI to ESP goal method;
13          plan_generation({CBI});
14         end;
15       Translate execution parts of CB to ESP subgoal methods using problem-solving primitives;
16       if V has a component or functional block with part-whole relation CBPW
17        then Translate CBPW to ESP goal method using problem-solving primitives;
18       if V has constraint statements Constrs
19        then Translate Constrs to ESP subgoal methods using problem-solving primitives;
20       V1 ⟵ a first element selected from CBPW;
21       if V1 is a component or functional block
22        begin
23         plan_generation({V1});
24         V2 ⟵ CBPW \ V1;
25         plan_generation(V2);
26        end;
27       Translate execution parts of V to ESP subgoal methods using problem-solving primitives;
28      end;
29      else
30       begin
31        V3 ⟵ a first element selected from V;
32        plan_generation({V3});
33        V4 ⟵ V \ V3
34        plan_generation(V4);
35       end;
36 end;
```

(g) Plan generation algorithm

Fig. 5 General algorithm of dataflow analysis and plan generation

```
Inputs: Component or functional block V of the design object
Outputs: Returns "success" if analysis succeeds.

1  procedure topdown_analysis(V);
2    begin
3      Get constraint statements Constrs from a table of a component or functional block V
4      Constr ⟵ an first element selected from Constrs;
5      while (there exists Constr)
6        Analyze a constraint statement Constr for a block from a table;
7        Perform dataflow analysis and grouping of Constr for a component from table;
8        Constrs ⟵ Constrs \ Constr;
9        Constr ⟵ a first element selected from Constrs;
10     end;
11     if V is a functional block
12       then
13         begin
14           V' ⟵ V ∪ components of functional block V;
15           for V'' ∈ V' do
16             Perform dataflow analysis and grouping of V '' for a block from table;
17         end;
18         return "success";
19   end;
```

(d) Topdown dataflow analysis algorithm for the whole system

```
Inputs: Constraint statement Constr
Outputs: Returns "success" if analysis succeeds.

1  procedure dataflow_analysis_and_grouping(Constr) ;
2    begin
3        Initialize an operand table of Constr;
4        Make an operand table of Constr;
5        Assign a level to an operator of Constr;
6        Perform a grouping of an operator of Constr;
7        return "success";
8    end;
```

(e) Dataflow analysis algorithm for the constraint

```
1  procedure plan_generation_main ;
2    begin
3      V ⟵ a list of components and functional blocks
             representing constraint analysis sequence
4      plan_generation(V);
5    end;
```

(f) Main plan generation algorithm

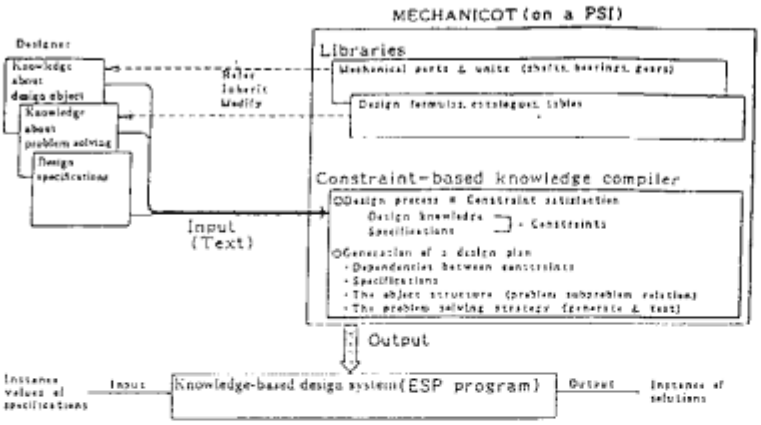
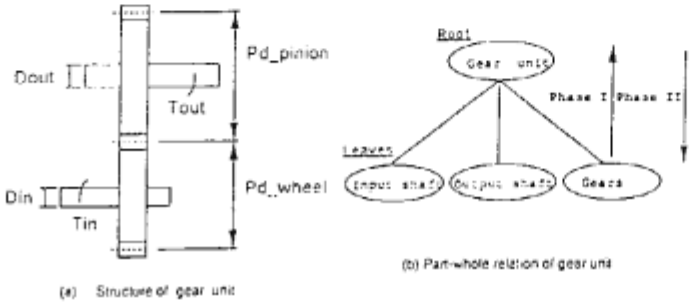


Fig. 6 Architecture of MECHANICOT



(a) Structure of gear unit

(b) Part-whole relation of gear unit

Input

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

Output

Rg, Pd_wheel, Pd_pinion, Din, Dout, Rout

(c) Input parameters and output parameters

Fig. 7 Schematic description of a gear unit

Gear design

1. For each gear (Pinion and wheel)

$m = Z/Pd$ (1)

$Pd \geq Ds \div 7m$ (2)

$Pd \leq 2000/\pi \cdot Rpm$ (3)

m: gear module [m]

selected from the standard values set of

{2.0 2.5 4.0 5.0 6.0}

Z: number of gear teeth

Pd: gear pitch diameter [m]

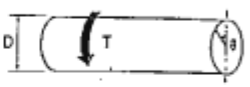
Ds: shaft diameter [m]

Rpm: number of shaft revolutions [rpm]

2. For a pair of gears

$Rg = \frac{Pd\ wheel}{Pd\ pinion}$ (4)

Shaft diameter



$D = \sqrt[4]{\frac{32T}{\pi G} \times \frac{180 \times 10^3}{\pi \theta}}$ (5)

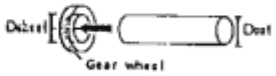
D: shaft diameter [m]

T: twisting moment [kg·m]

θ: torsion angle [rad]

G: shearing strength [kg/m²]

Structural constraints



Dwheel ≥ Dout (6)

Dwheel: gear hole diameter

Dout: shaft diameter

Fig. 8 Constraints for gear unit design

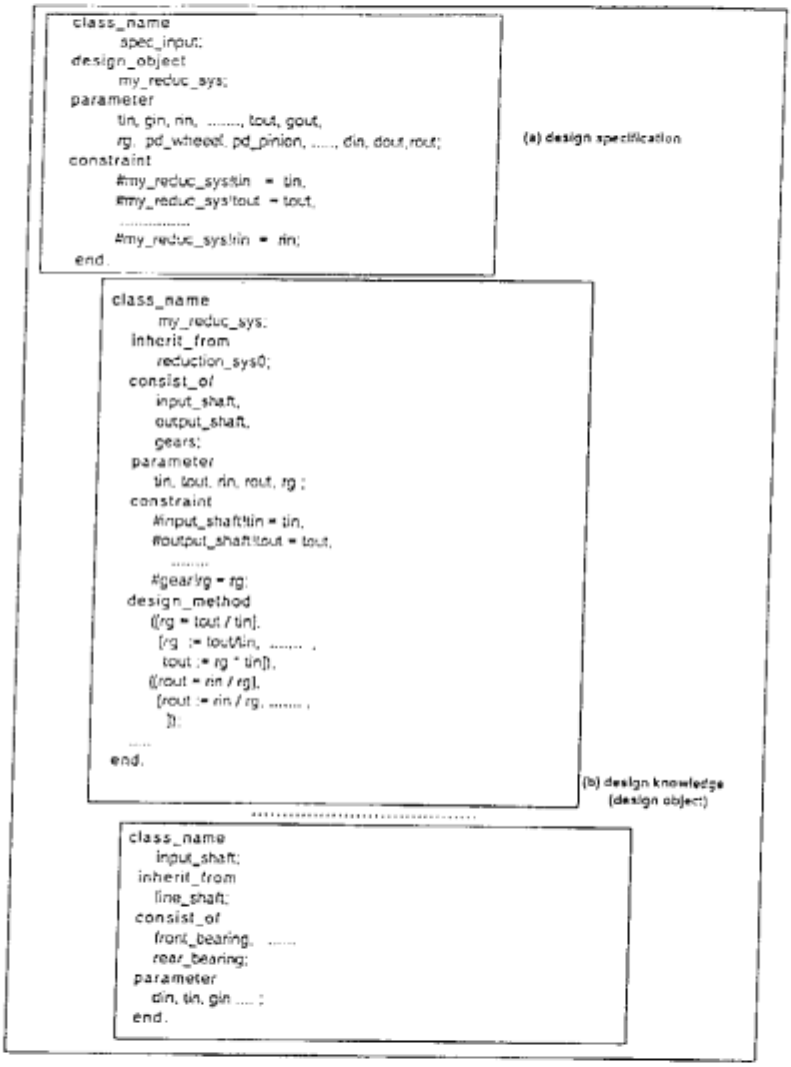


Fig. 9 Input design specification and design knowledge

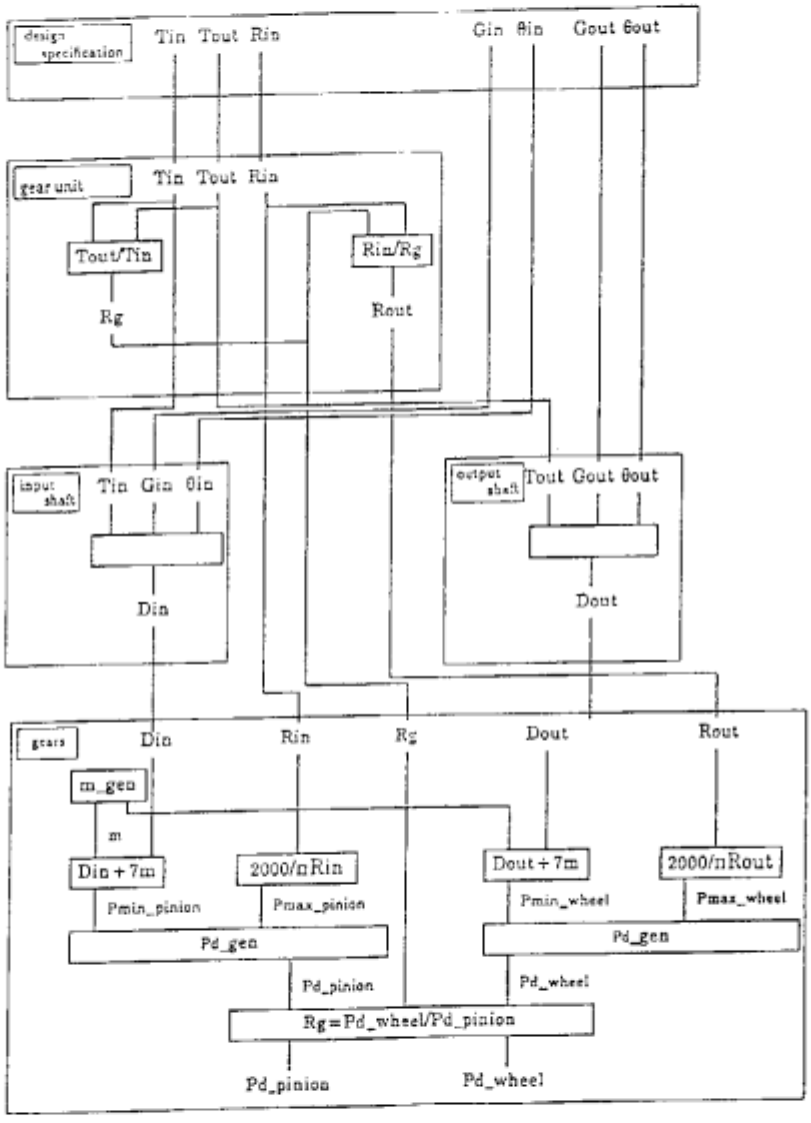


Fig. 10 Constraint network description of gear unit

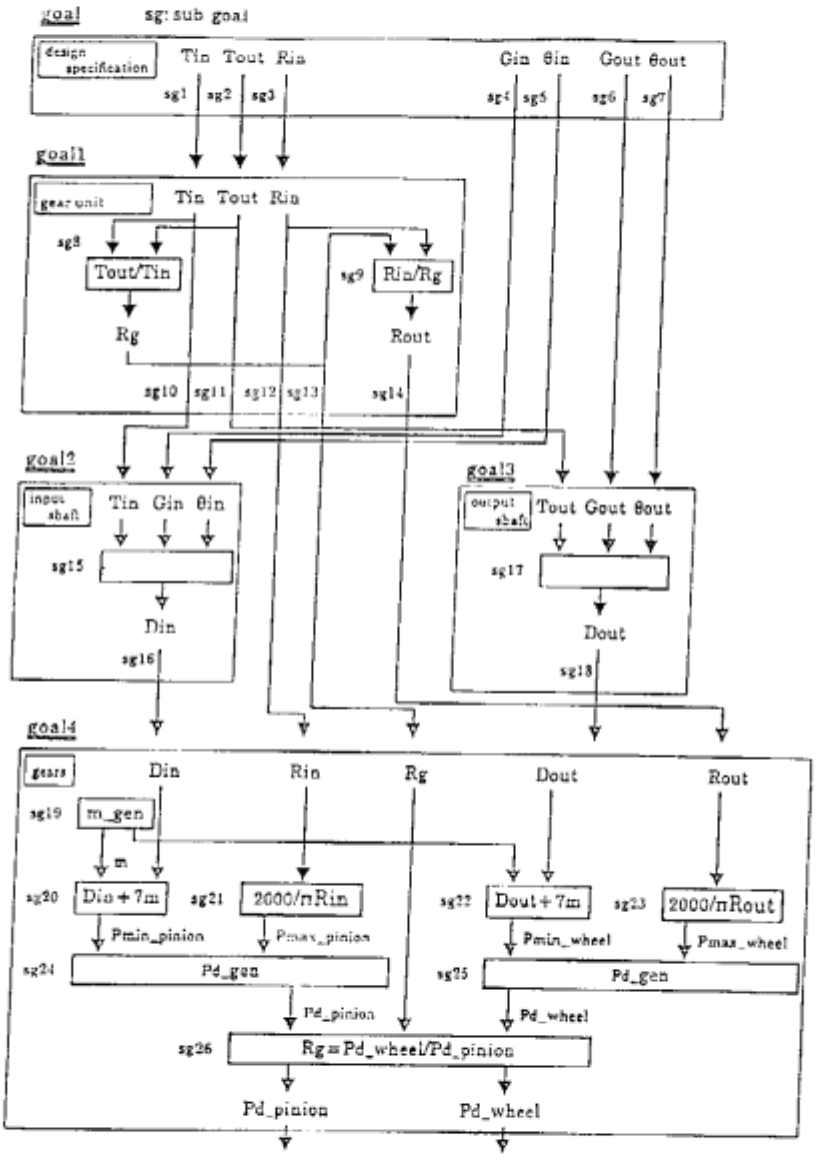


Fig. 11 Generated design plan

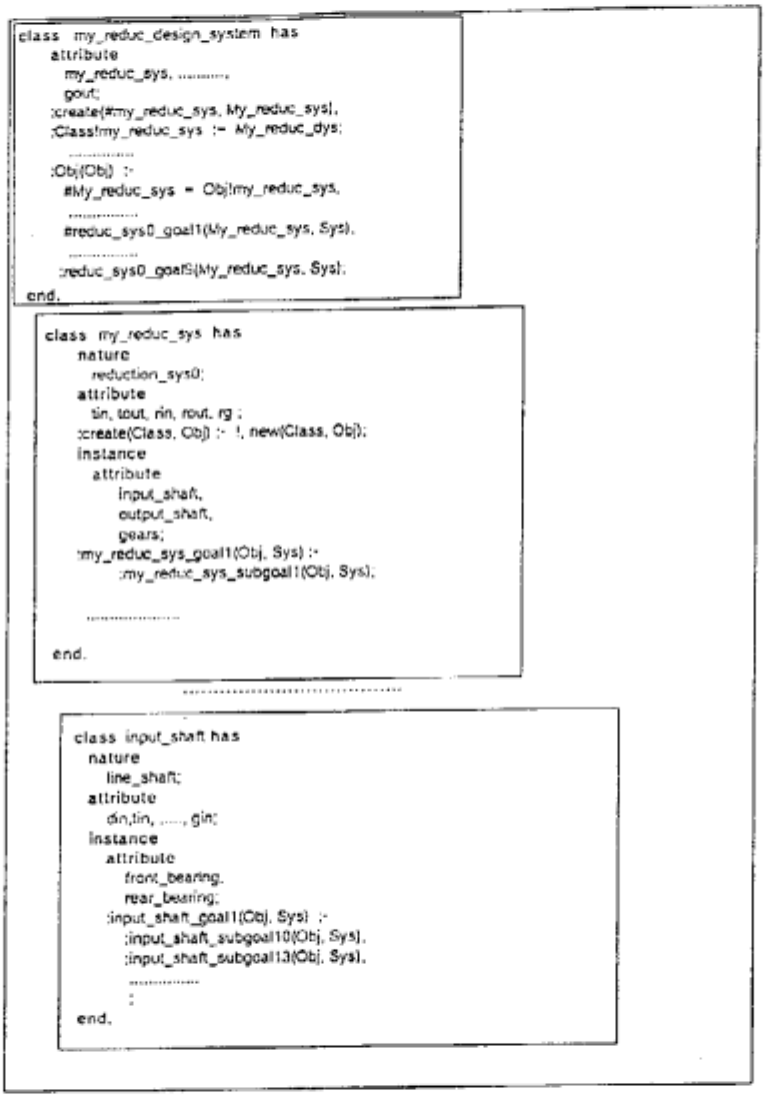


Fig. 12 Design plan written in ESP code

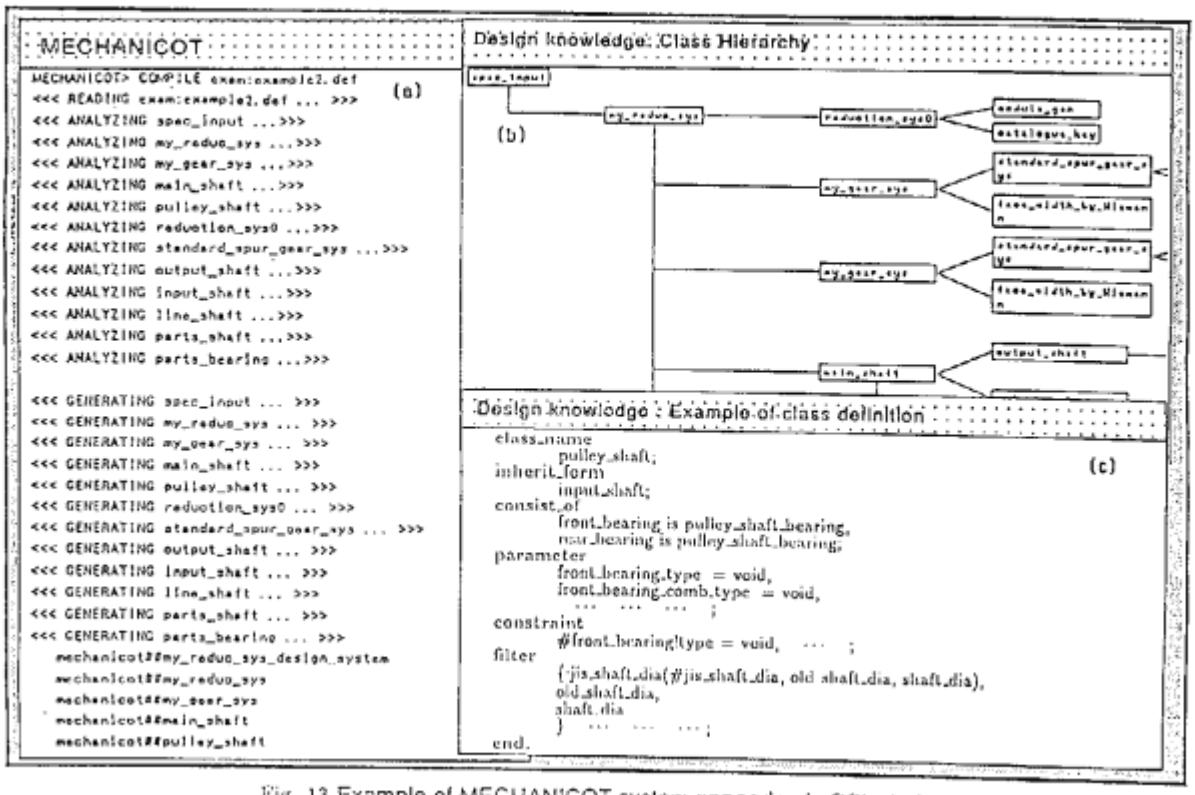


Fig. 13 Example of MECHANICOT system appearing in PSI window

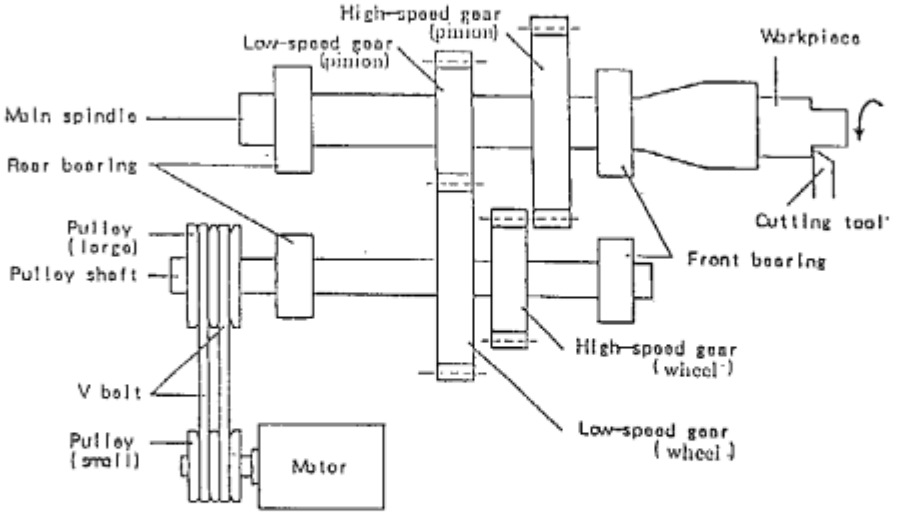


Fig. 14 Outline of design object: main spindle head of lathe

Table 1 Problem-solving primitives according to the problem-solving types

Problem-solving types assigned to a constraint	Constraint incorporated problem-solving primitives
Generator	name_gen(*Parameter*, [*Min*, *Max*, *Inc*]) name_gen(*Parameter*, [*Discrete_set*])
Tester	name_tester(*Constraint*, [*Input_para*], [*Result*])
Design_method	name_design_method(*Constraint*, [*Methods_or_Functions*])
Filter	name_filter(*Constraint*, [*Input_para*], [*Output_para*])

Note

Min: minimum value, *Max*: maximum value, *Inc*: increment,
Discrete_set: a set of discrete values, *Input_para*: input parameters,
Constraint: constraint, *Result*: result or output value,
Methods_or_Functions: a set of methods or functions,
Output_para: output parameters.

Table 2 Example of input and output parameters of a main spindle head design

Input parameters (design requirements)		Example of values
Cutting capacity	Workpiece material	S48C
	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*.

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 constraint analysis

Problem-solving type		*G*	*T*	*DM*	*F*	*C*
Design problem	Gear unit	3	2	28	2	75
	Spindle head	11	9	105	2	192

Note

G: *generator*, T: *tester*, DM: *design_method*,

F: *filter*, C: *structural constraint*.