ICOT Technical Memorandum: TM-0745

TM-0745

Design Plan Generation through Constraint Compilation

by
Y. Nagai & K. Ikoma

July, 1989

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

Design Plan Generation Through Constraint Compilation

Yasuo NAGAI

Information & Communication Systems Laboratory
Toshiba Corporation

70 Yanagicho, Saiwai-ku, Kawasaki-shi, Kanagawa, 210, Japan

Phone: Tokyo +81-44-548-5353.

Kenji IKOMA

Fifth Research Laboratory
Institute for New Generation Computer Technology

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

Phone: Tokyo +81-3-456-3192.

Keywords

design expert system, design process, design plan, knowledge compilation, constraint, constraint compilation, constraint-based problem solving, mechanical design

Abstract

The goal of this research is to consider constraints as a sophisticated representation of knowledge and to improve the efficiency of the constraint-based problem solving mechanism. As the first step, we propose to generate design plans by knowledge compilation. For this we need to clarify the architecture of expert systems for various types of design [27] such as circuit design [19,20,32], mechanical design [13,23,24], and configuration [21]. Therefore, we discuss the technical issues from the viewpoint of knowledge representation and problem solving. In particular, we regard constraints as a suitable knowledge representation paradigm different from rule and frame representations, and constraint-based problem solving as a suitable new problem solving paradigm. We propose primitive tasks for the constraint-based problem solving mechanism, based on the design process model. We define routine design and discuss the expert system architectures for design problems and the necessary functions for solving them. Furthermore we consider constraints as a sophisticated knowledge representation and study efficient problem solving for constraint processing.

This paper reports on the basic concepts of knowledge compilation, on knowledge compilation for design problems, and on design plan generation through this method, using constraint representation and constraint-based problem solving. *

1. Introduction

A large quantity of knowledge that depends on the specific design objects is required to develop knowledge-based systems for design problems. Therefore a knowledge representation specific to the design problem and an efficient problem solving method that uses this knowledge are keypoints for research on a tool for building design expert systems. The goals of our research are to clarify the architecture of knowledge-based systems for design problems and to realize expert system building tools [26].

* This paper describes research done at the Fifth Research Laboratory of the Institute for New Generation Computer Technology (ICOT).

Different kinds of designs require different design methods, parts, and know-how, and have given rise to different standards. The process model and the decision sequence necessary to realize design systems are not formalized explicitly, except for parts of the stylized design such as routine design. For this reason, most design systems are built and utilized as specialized systems for special purposes. These systems are provided to designers or users as individualized systems. Is is difficult for the users to maintain and extend these systems, and it is not also efficient for them to use these systems as general purpose tools. We therefore suggest the realization of general purpose building tools, enabling system builders (and designers) to build and maintain design systems. In other words, we want to avoid describing design knowledge as existing procedural programs and forcing system builders to describe design knowledge in the representation form which is dependent on the specific problem solver. This is a very important problem, since design knowledge should not be invisible to designers. The knowledge can be embedded inside the system, and it is easy to reuse design knowledge by inheriting knowledge among designers.

Consequently, we shall take the approach described in Fig.1, which shows design knowledge as separate from the problem solver, and both being represented declaratively. The environment in which the designer can build the system by combining them is provided, and the problem solver of the built system can be handled efficiently.

In most design knowledge, relations between objects may be represented using symbolic descriptions such as mathematical formulas. When we compare the declarative with the 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.

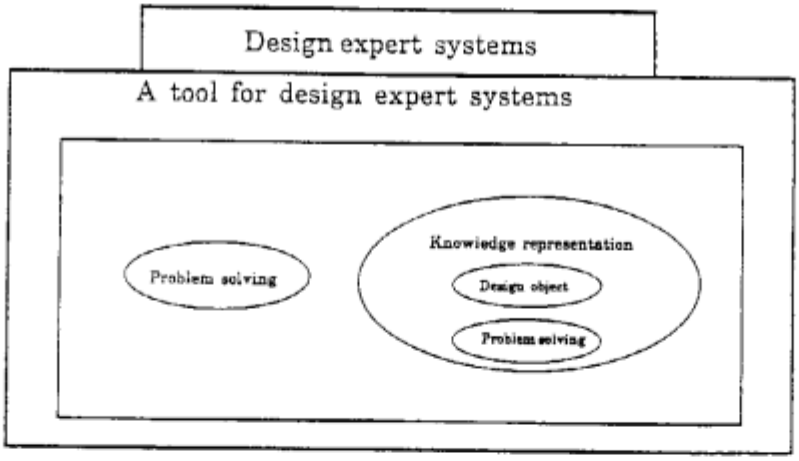

Fig. 1 Research Approach

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.

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

We will focus on this advantage of constraints. consider the realization of constraint-based problem solving for design problems, and regard knowledge compilation as the most important technical issue. Generally, knowledge compilation is a technique by which knowledge about the domain is stored in declarative form.

such as facts and theories, and this stored knowledge is applied and utilized by interpretive procedures. This technique makes existing paths of processing more efficient rather than enabling new paths of processing. In this case, we regard knowledge compilation as a technique by which knowledge linked to an efficient problem solving task (shallow knowledge) is generated using principled knowledge such as physical laws and structure of objects (deep knowledge). Therefore, more efficient procedures specific to the task domain can be generated using the knowledge compilation technique [2].

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 (semi-)automatically, using design knowledge represented declaratively.

Before discussing knowledge compilation in detail, we clarify the architecture of knowledge-based systems for design problems. Therefore, knowledge representation, problem solving and the architecture of design expert systems are discussed in Section 2.

Section 3 gives an overall flow of knowledge compilation, concentrating on the problem of knowledge compilation for design problems.

Section 4 describes design plan generation through knowledge compilation and constraint compilation. We describe the characteristics of mechanical design problems which have guided us in analyzing the problem. Then we define in detail the process of design plan generation through constraint compilation and describe the constraint compiler. Last we describe an actual constraint compiler that applies the concept of knowledge compilation to mechanical design, MECHANICOT.

We give the details of design plan generation using MECHANICOT, giving as an example the problem of gear unit design.

2. Research Issues of Design Expert System

2.1 Knowledge Representation and Problem Solving

Existing expert systems are broadly classified into systems that solve analytical problems and systems for synthetic problems. Analytic problems like diagnostic problems generally select hypotheses in a limited solution space, because a set of hypothetical solutions and a set of rules for selecting hypotheses can be predetermined. Synthetic problems, however, need efficient problem solving, since solution spaces are so large that generating candidate solutions beforehand is difficult. Design problems are typical examples of synthetic problems. The development of a design expert system requires a large amount of knowledge that depends on a design object. Representation of design knowledge and a good problem solving mechanism are important for a design expert system.

This section gives an overview of the design knowledge representation and of the problem solving mechanism in the design expert system.

2.1.1 Knowledge Representation

Knowledge representation requires two facilities: knowledge must be represented suitably for the expert systems, and designers must be able to represent their own knowledge easily. Design knowledge is knowledge about the design objects themselves and about problem solving [36]. Knowledge about design objects consists of the structures, shapes, and attributes of the object being designed. A system's knowledge about a design object is called an object model. Knowledge about problem solving, however, includes methods for analyzing object models, for evaluating and modifying solutions, for designing the object, and for choosing among candidate solutions. A design process can be regarded as a design requirement satisfaction process; operations such as selection, modification and refinement are repeatedly applied to an object model by using knowledge about problem solving. Furthermore, to enable designers to build an expert system by themselves, we require an environment in which a design expert system can be built by declaratively representing an object model and knowledge about problem solving. To realize the environment, we propose a tool that generates a design plan from separate inputs of an object model and knowledge about problem solving, and that provides an interface between design knowledge and the problem solver.

2.1.2 Problem Solving

If a design plan is given explicitly, a design problem can be solved according to the plan. There are often cases where a design plan cannot be given explicitly, but only constraint and problem-solving heuristics can

be given. An effective way of solving these cases is to employ constraint-based problem solving, regarding a design process as a constraint satisfaction process. In addition, the whole design process can be seen as a constraint satisfaction process; an object model represents constraints on the structures of the design objects, and design requirements and knowledge about problem solving also represent constraints. These constraints are given priorities and changed dynamically according to the designer's intention and preference, and to trade off between performance, due dates and cost. Therefore, a constraint solver suitable for a design problem is required.

The structural information derived from the object model is constraint expressed explicitly. Design knowledge such as methods for analyzing object models and design requirements such as cost and performance are also regarded as constraints. However, many of the existing tools that support the construction of expert systems do not make it easy to express the constraint concept explicitly; the person constructing the system must use a tool-specific language to realize mechanisms for applying constraint representations which depend on the design object. We therefore represent design knowledge declaratively by introducing the concept of constraint explicitly, and realize a suitable problem solving mechanism. For this, the classification of constraints in design problems is important.

When considering design problems there are of course various domain-specific constraints. These constraints are related to each of the stages of design process composed of the conceptual design, fundamental design and detailed design. They are described in this section.

In conceptual design, the constraints are the description of performance and cost from the list of the requirements and the specification. In fundamental design, the constraints are the ways of mapping or instantiating the functional description to the real or physical world. For example, when models are selected and performance is analyzed and evaluated according to them, the constraints are derived from these models.

In detailed design, the design object is refined according to the selected model. Then the constraints are the form and structure representations together with knowledge about the design style for component configuration and about relations between components. In this case, because the model depends on the design object, the constraints derived from the model depend on the design object [22].

Next, we will classify constraints for this design problem according to the following characteristics and discuss them [25].

1) Static and Dynamic Constraints

Many constraint systems consider constraints to be static entities. Static constraints are specified in advance, and are constant and unchanging. In design problems, however, not all constraints are given in the initial stages of a design process; many are added or deleted during the design process and are dynamically changed in design problems. They are imposed depending both on interactions with the user and on the system; they tend to change, with their range of applicability varying.

2) Obligatory (Hard) and Suggestive (Soft) Constraints

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

3) Local and Global Constraints

One way of solving a design problem is to divide it into subproblems. Thus, it is necessary to distinguish whether the applicable scope of a constraint closes locally within a subproblem or is globally related to other subproblems. In addition, interactions between local constraints within a subproblem and interactions between local and global constraints must be considered. Local constraints are used to conduct searches when a state changes within a given model, object or process and the scope over which the constraint is valid is limited to within the model or object or process. Global constraints are used when a state is to be evaluated using not only local constraints, but all related constraints, without imposing any limit.

4) Finite and Infinite Domains

Some constraints in design problems are represented as inequalities. Therefore, not only do constraints propagate values, they also propagate over interval bounds in which variables that can take certain values must be considered. Variables of the constraint are not constant; these constraints propagate over the interval bound as a label [8].

Because one constraint may possess multiple characteristics, there are some combinatorial possibilities of the above classified constraints. Therefore, the framework that can handle constraints from the uniform viewpoint is expected. In the following, the constraint-based problem solving mechanism that considers several characteristics among them is discussed.

2.2 Design Process and Constraint-Based Problem Solving

As shown in Fig.2, we will divide a formalization of knowledge-based system for design problems into three levels: knowledge level, architecture level, and program level.

At the knowledge level, after a type of design problem is determined, the corresponding solution space, design theory, and design specification are formalized.

At the architecture level, the (constraint-based) problem solving model is determined according to the design process. This problem solving model can be realized by applying various solution methods such as problem decomposition, constraint propagation, failure recovery, (hierarchical) generate & test, least commitment, and (linear) approximation. The details of the problem solving model are described in Section 2.2.2.2.

At the program level, knowledge-based systems are realized using expert shells composed of knowledge representations such as rule and frame descriptions and programing languages such as logic programming and constraint logic programming languages, according to the fixed architecture of these systems.

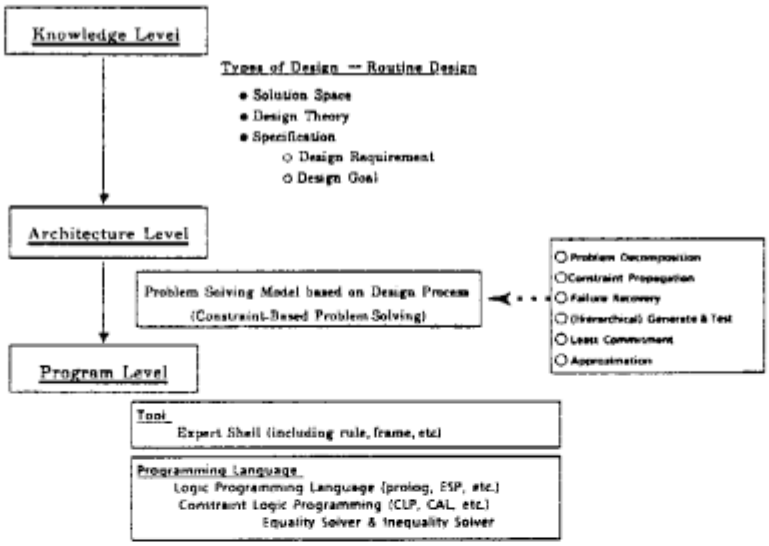

Fig.2 Formalization of Knowledge-Based Systems

Consideration of a modeling of the design process model is required to realize a problem solving mechanism for design problems using a constraint satisfaction process. Furthermore, a more efficient problem solving mechanism can be realized using tools and programming languages at the program level.

This section considers the design process, the target architecture of the design expert system including constraint-based problem solving mechanism relative to the design process, and tools and programming languages to realize the problem solving mechanism.

2.2.1 Design Process

At the knowledge level, we need to formalize a knowledge-based system for design problems according to the design process model.

Fig.3 shows a model of the design process for routine mechanical design. This design problem consists of editing pre-existing designs, and is based on this model. 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 catalogues or non-standard parts from the design. Most of the strategies for selecting standard or non-standard parts for an implementation of components are described in the specification or requirements. They generally trade off the performance and cost.

The fundamental tasks at each level makes the iterative design composed of the problem decomposition and refinement proceed according to the design plan. If a design fails, redesign is executed, and the problem is decomposed and refined again. It backtracks the previous design decisions in the tasks at the higher level or executes local modification at the same level, and executes the iterative design.

Planning decomposes and refines the problem or specification according to the design plan. The design style determined from the design plan, in other words, the architectural knowledge about the design object, is indexed by the requirement or specification of the design, and can be regarded as constraints. The refinement, optimization, analysis, and evaluation tasks are selected and executed according to the design style. The decomposition of the requirements or specifications of the design are executed by applying the design style. It is assumed that problem decomposition can transform or map the problem to the subproblem or component. There are two methods for problem decomposition: in one case the subproblems are interdependent, in the other they are independent of one another. In the former we consider the relations between the components at the same level, and in the latter we consider the relations between the components and subcomponents.

Refinement transforms the divided specification into structural representation composed of the components and relations between them. These relations between components can be regarded as constraints. Constraints on the component attributes are particularly important. For example, the mechanism for the propagation mechanism of constraints in decomposing into interacting subproblems is different from that in decomposing into independent subproblems. The former mechanism propagates the interactions among subproblems as the constraints, and the latter propagates the constraints upward or downward according to the hierarchical representation of the design object.

Optimization modifies the structural representation locally, so that the functions expressed in the specification do not change.

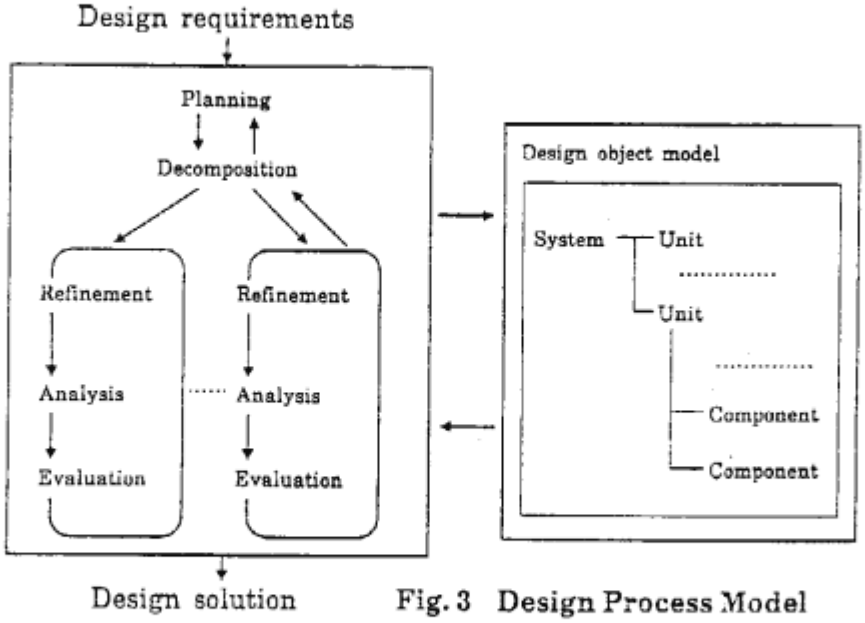

Fig. 3 Design Process Model

2.2.2 Target Architecture of Design Expert System

2.2.2.1 Problem Solving Models for Constraint Reasoning

The architecture of design expert systems which use the constraint-based problem solving mechanism is composed of the following primitives: the generator, propagator, tester, and the failure recovery module, as shown in Fig.4. This problem solving mechanism is extended based on a generate & test method.

The generator assigns values to parameters or maps functional components to the physical components. The generator can take either a continuous or a discrete value. The former assigns parameters of the attributes by local modification based on the predetermined components. The latter assigns parameters by retrieving the standard parts for implementing components from the catalogue, a table look-up method.

The propagator assigns or selects values to parameters by activeiy evaluating constraints and propagating constraints.

The tester checks the constraints and can be considered as the passive handling of constraints. In general, the inequality description can be handled by the tester, but in some contexts, constraints can also be considered as the equality.

The failure recover module modifies the attributes of the components locally using the advice mechanism and plans problem decomposition. The advice mechanism repairs partial or local designs using heuristics about the attributes of the components. It uses the above generator and propagator as primitives. In failure recovery handling, both the obligatory and the suggestive constraints must be handled. Suggestive constraints are selected or relaxed by a planning algorithm that tries to satisfy as many constraints as possible.

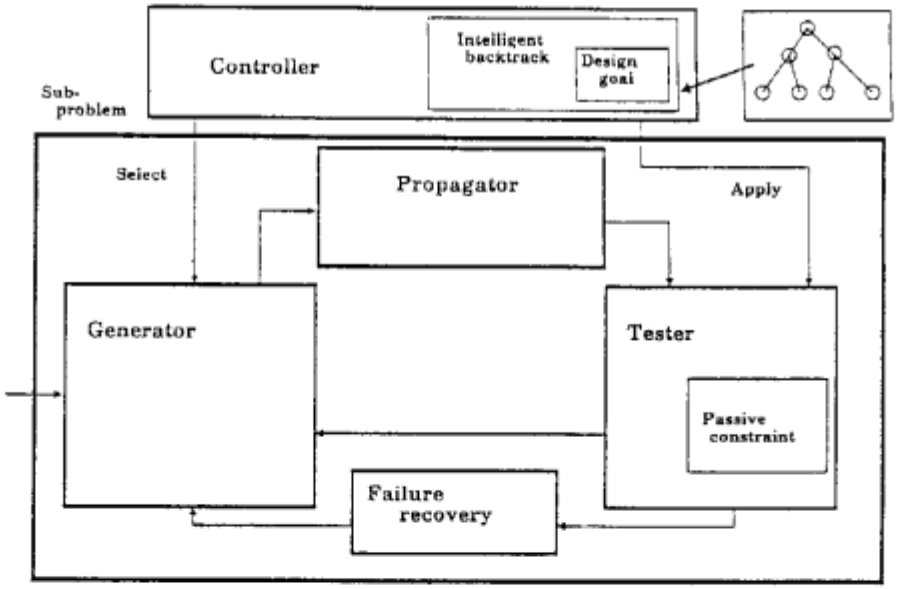

Fig. 4 Problem Solving Model for Constraint Reasoning

2.2.2.2 Architecture of Design Expert System

We describe the architecture of the expert system for routine design. Research has been conducted on architectures consisting of primitive tasks for routine design, called generic tasks for design [5,6,23,28]. 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, they do not provide a modeling facility and it seems that they are insufficient to handle the constraint representation for this generic task. Therefore we investigate the architecture including the constraint representation and its problem solving mechanism. Constraint representation is proposed as a new paradigm for knowledge

representation, and the problem solving mechanism as a new paradigm for the architecture of routine design expert systems.

We now define and describe in detail constraint-based problem solving in routine design expert systems.

Constraint-based problem solving corresponds to an efficient execution of the design plan obtained by compiling design specification and various related types of design knowledge. The initial constraints are the design specification and knowledge about the design object; the compilation process is based on the fundamental tasks of the design process, such as planning, problem decomposition, and refinement. The constraint-based problem solving mechanism is described according to the above constraint classification by matching the design process model for a routine design to the design system or tool.

When we solve the problem, we can deal with this problem by using various problem solving strategies and inference methods. Even though we solve the same problem, there are several solution methods and the difficulty of the problem may depend on each solution method.

Therefore, it is necessary to realize the flexible and efficient constraint-based problem solving mechanism to select and apply the following suitable solution methods and inference methods.

1) Constraint Propagation and Its Control

When, in the process of satisfying constraints, a value is assigned to one variable, the values of other variables may be determined by the former variable; this is a mechanism of constraint propagation.

Both data flow analysis and simultaneous equations are typical examples for a solution method of constraint propagation [4,33]. Data flow analysis takes care of local constraint propagation, simultaneous equations used when the problem cannot be solved by local constraint propagation only. In particular, when using data flow analysis, a trade-off between constraints may result if the constraints cannot all be satisfied. Clearly, we need a strategy for controlling constraint propagation.

Furthermore, in hierarchical design, interactions between constraints must also be taken into account when realizing constraint propagation. In particular, structural constraints should be considered in mechanical design. Structural constraints are reflected in the design style, specifications, and requirements at each abstract level of the design, and determine the structural decomposition, partition, and design style at lower design levels. In hierarchical design, the constraints are propagated from higher to lower design levels. The design style constraints determine the structure of the design object and the problem decomposition at lower design levels. Constraints are partitioned through the structure of the design object and decomposition of the design problem.

2) Failure Recovery

The failure recovery retries the design decision when the active assignment of values to variables by means of generators and propagators fails, or the tester returns a failure. This failure recovery executes an efficient generate & test process using procedural heuristic knowledge, and executes a constraint relaxation.

Constraint relaxation is applied to weak constraints. It is equivalent to searching for alternatives to the specified constraint. That is, at the failure stage, when a constraint has not been satisfied, we search for alternative constraints at the same or a lower level. Selection involves the choice of a constraint when there are two or more competing constraints, and is regarded as constraint interpretation. In this way, constraint relaxation can be formulated as a planning problem [11].

3) Least Commitment

Another approach to the problem of constraint interactions is minimizing interactions between subproblems. This is referred to as the principle of least commitment; by delaying constraint evaluations as long as possible, refinements according to the design plan are executed, and evaluations are performed when necessary [31].

4) (Hierarchical) Generate & Test

Generate & test is a strategy for general problem solving. In this strategy, a generator is used to generate values for variables and a tester is used to test these values. This strategy may make the problem solving mechanism inefficient, depending on the kind of problem being solved. The characteristics of the problem therefore should be considered carefully. In design problems the problem must be divided into structured

subproblems with hierarchical levels; we can then apply generate & test again in each subproblem to give an efficient problem solving system.

5) Problem Decomposition (Divide & Conquer)

If we consider design by step-wise refinement, interaction between constraints among separate subproblems are extremely important. Considering practical design problems, we need to decompose the problem into subproblems such that the interactions among the subproblems can be minimized.

6) Preservation and Management of Dependencies among Constraints

In processes where the values of constrained variables are propagated through the execution of constraint propagation mechanisms, the preservation and management of dependency relations among constraints, variables, and constant values are deemed important for resolving such contradictions which may arise in variable values and to explain the values generated [15]. A mechanism for monitoring constraint evaluation should be included in any problem-solving mechanism that relies on constraint representations. It manages constraint checks and ensures consistency, and is to some extent realizable using demons or attached procedures.

7) Adoption of Solution Methods for CSP

A constraint satisfaction problem (CSP) is a problem that assigns values to all variables such that all relations on variables (all constraints) can be satisfied when a set of finite variables is given where each domain defines the set of finite values that variables can take. A CSP can be represented in terms of constraints [9,10]. Design problems can be represented in terms of the constraint network, but there are alternative methods for generating values to variables in constraints. Therefore well-structured design problems require strategies for efficient problem solving, such as control of the search process, in addition to the other techniques for solving CSPs.

2.2.2.3 Relation to Programming Languages (Program Level)

A language scheme called constraint logic programming (CLP) has been proposed [12,16,30]. This scheme defines a class of languages designed to deal with constraints using a logic programming approach. It handles mathematical formulas composed of linear equations and inequalities as algebraic constraints. The interpreter of most CLP languages consists of three modules: inference engine, constraint solver, and preprocessor or interface modules. The constraint solver solves constraints which cannot be handled by the engine. In other words, it determines the solvability of that set and, if solvable, computes the solutions, given a set of constraints. To obtain the solutions, it needs the solution methods for a set of constraints, that is the solution methods for simultaneous equations

CLP languages allow more flexible evaluation and assignment of values to variables than PROLOG does. Conventional programming languages must determine the ordering of the evaluation and assignment in the form of the procedural statement and must bind all arguments to values. Therefore, CLP languages provide more flexible and expressive power for describing constraints than do conventional programming languages.

The design plan generated using the knowledge compilation technique is the constraint network description; it includes the design knowledge, the problem-solving heuristics and problem-solving primitives of the predetermined architecture. The mechanism for the interpretation and execution of this design plan mostly depends on the mechanism of constraint propagation. Constraint propagation and its control are very important for constraint-based problem solving; both local and non-local propagations must be taken care of. CLP languages can handle this propagation mechanism easily because both the local and the non-local propagations can be handled by the constraint solver, which can be viewed as a generalization form of unification. Furthermore the CLP's logical, functional, and operational features are available as in conventional logic languages. Thus the CLP language scheme is suitable for the above constraint-based problem solving. In the second version of the knowledge compiler environment, the design plan description is translated into this language. Problem solving primitives for constraint-based problems other than the ones described here will be provided as extensions of this language, since the CLP language is the programming language.

3. Knowledge Compilation

Knowledge compilation techniques are being investigated in many problem areas such as diagnostic and machine learning problems.

This compilation is a technique by which knowledge in declarative form, such as facts and theories, about the domain is stored and this stored knowledge is applied and utilized by interpretive procedures. This technique makes existing paths of processing more efficient rather than enabling new paths of processing. Therefore, more efficient procedures specific to the task domain can be generated using the knowledge compilation technique.

In this case, knowledge compilation means to transform knowledge representation (problem specifications) at an abstract level to one at a more concrete level; from knowledge level to architecture level, from knowledge level to program level, and from architecture level to program level. The purpose of this method is to improve the efficiency of the utilization of various kinds of knowledge and the problem solving mechanism.

In the following sections we introduce knowledge and constraint compilation for design problems, design plan generation for routine mechanical design, and the current state of our research.

3.1 Knowledge Compilation for Design Problems

In this section we define knowledge compilation for design problems.

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 this object often changes 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 circuit design to mechanical design and can construct design systems or tools easily.

By dividing design knowledge into knowledge about 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 the constraint satisfaction problem. Knowledge transformation is a very important technique to handle these independent kinds of knowledge uniformly, to generate design plans according to primitives of constraint-based problem solving, and to improve the efficiency of the problem solving. We consider two methods for transformation: one 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 design object is fixed, and an efficient problem solving mechanism can be realized.

The second method is to translate knowledge of object models and knowledge about problem solving into intermediate descriptions and to interpret these intermediate descriptions. This is an interpretation approach, which corresponds to an interpretation of a design plan. Furthermore, it is possible to interpret a design plan generated during problem solving efficiently by reusing knowledge derived beforehand using knowledge transformation techniques (including the design plan and its intermediate description). At first, we focus on a design problem where the structure of the design object and knowledge about problem solving are fixed. Therefore we adopt knowledge compilation as our knowledge transformation technique.

This knowledge compilation for design problems is a technique that transforms the input design specifications into the design plan, assuming that the structure of the design object has been determined. Fig.5 shows an overview of knowledge compilation. The input of the knowledge compiler is a design specification which includes the functional description and constraints such as performance and resource limitation in the form of a parametric description.

The knowledge compiler determines the structure of the design object and the analysis and evaluation methods, based on the instances of the configuration and mechanism of the design object stored in the knowledge base. The structure of the design object may be given by the designer through the user interface. The knowledge base stores knowledge about the design object model such as constraints for the analysis and evaluation of the model, its structural knowledge, and public knowledge such as the catalogues and parts standards. The compiler analyzes the relationship among the components of the design object, studies

the problem decomposition and refinement method, determines the relationships among the constraints and parameters and among the components or parts and attributes. The design plan by which the design specifications can be satisfied is generated by compiling design knowledge and problem-solving heuristics [3] along the architecture described in Section 2.2.2. The generated design plan can be considered the program for the design.

The above analysis is used to generate the design plan; the functional or physical components are determined for implementation of the design object by retrieving knowledge of the catalogues and parts standards, assuming that the structure has been determined. The compiler also analyzes and evaluates the design object at this time and then optimizes the design plan, considering the problem decomposition taking care of both the independent subproblems and the subproblems with interactions.

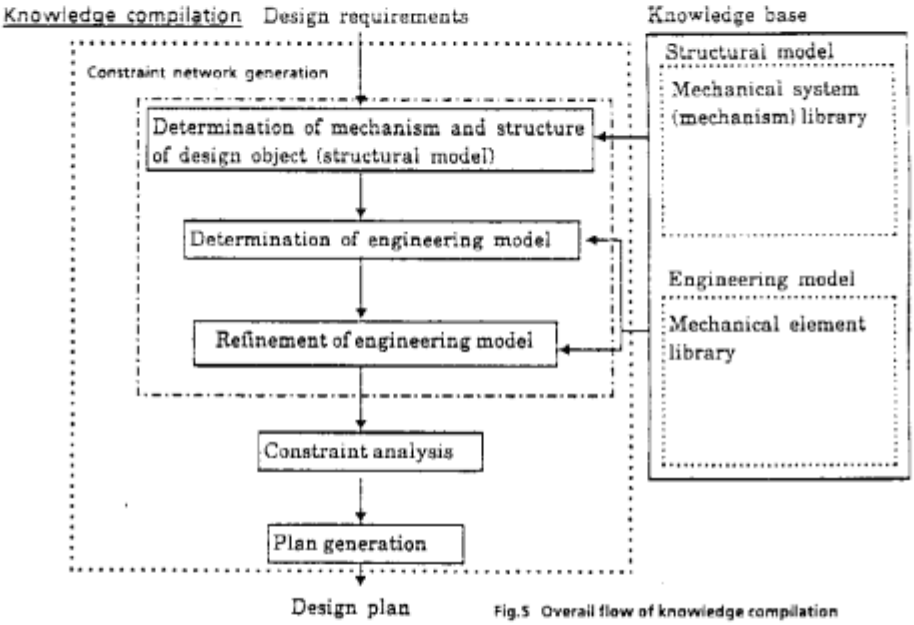

Fig.5 Overall flow of knowledge compilation

Next, we will adopt a knowledge interpretation approach to handle a design problem where the structure of the design object and knowledge about problem solving are changing during the problem solving.

The interpretation of the design plan is executed by the design plan interpreter which is based on the problem solving model shown in Fig.4; the synthesis and analysis tasks in the design are executed according to this design plan which is generated by knowledge compilation, using the interpreter based on the predetermined system architecture.

In future, a design system building environment customizable by the designer will require this design plan generation facility using both knowledge compilation and knowledge interpretation.

3.2 Mechanical Design Problem

In mechanical design it is difficult to modularize the design object because the geometrical information, the representation in three dimensions, and manufacturing and assembly information are closely linked with the design object. This is why the behavior of the design object changes as the geometric features change: the geometric features depend on the functional description or fabrication information. This results from the dynamic creation of the model from the components of the design object.

In contrast with circuit design, feature description at the functional level has little effect on feature description at the physical level and it is difficult to abstract the components of the design object from their behavior or function. Therefore, given the specifications, it is difficult to determine whether the behavior satisfies the specifications, so the analysis itself requires careful consideration.

Strategies for the decomposition of the problem are not always formalized clearly and not applied in mechanical design as in circuit design [22]. In circuit design, the design process at each level is formalized

so that the design tasks can be kept modular and simple. It is very important to consider the interactions between functional description and physical description for both circuit and mechanical design. In mechanical design, most interactions are between the function, based on the physical laws, and the form, such as topology or geometry of the design object at functional or physical level; the design problem must be dealt with by investigating the degree of decomposition of the problem or specification. In this case, we assume that the structure of the design object at the functional level has already been decided.

4. Design Plan Generation Through Constraint Compilation

4.1 Constraint Compilation

Constraint compilation is a technique by which knowledge about the domain (constraints, facts, and theories) is stored in declarative form and is applied by constraint-based problem solving mechanisms. A constraint compiler makes existing paths of constraint processing more efficient rather than enabling new paths [14]. Considering a design problem, it transforms the input design specifications into the design plan, assuming that the structure of the design object has been determined.

4.1.1 Constraint Compiler

A constraint compiler can specialize knowledge by combining knowledge independent of a certain design object and a designer's heuristics which depend on a certain design object. Since the constraint compiler can generate a design plan by analyzing dependencies among constraints, design knowledge can be also represented declaratively. Inputs to the constraint compiler are the design requirements, object models, and knowledge about problem solving. Reference to results of previous designs and the designers' heuristics about searching for alternatives are also represented as knowledge about problem solving. From these inputs, the constraint compiler analyzes dependencies among constraints and parameters, generates a design plan, and provides an interface between the design knowledge and the constraint solver at the architecture level or the languages at the program level. The output from the compiler is a specialized design expert system including the designers' heuristics at the architecture or program level. Therefore, a flexible environment in which an expert system can be built by designers themselves is obtained by dividing design knowledge into object models and knowledge about problem solving, and by generating design plans using a constraint compiler.

4.1.1.1 Overview of the Constraint Compiler

The general flow of control of the constraint compiler is shown in Fig.6. The compiler contains three main procedures: lexical and syntax analysis, inheritance relations analysis (by generation of class definition tables), and constraint analysis (by both generation and update of tables and determination of a constraint analysis sequence). The input to this compiler is user definitions and libraries, and the output is a design plan. Inheritance relations between functional blocks and between components are analyzed in the inheritance analysis part. In the constraint analysis sequence determination part, relations between components are analyzed and a directed-acyclic graph (DAG) that represents part-whole relations and abstract-concrete relations is generated using this analysis result. The constraint analysis sequence is determined according to this graph. In the constraint analysis part, dependencies among constraints inside components and functional blocks are analyzed according to this determined sequence. After this the compiler analyzes dependencies between functional blocks and components, between functional blocks, and between components, and finally among the constraints of the whole system. In the design plan generation part, a design plan that enables efficient problem solving is generated according to the dependencies between constraints assuming the problem solving primitives shown in Fig.2.

4.1.1.2 Constraint Analysis and Design Plan Generation

Constraint analysis executes as follows:

- Generates a constraint network from the constraint representation.
- Regards the generated constraint network as a graph description and calculates topological information of the graph description [18,37].

- Using that topological information, it extracts parts of this graph that contain loops and groups them as blocks.
- Groups as blocks those parts that contain no loops.

Thus, design plan is generated by grouping the blocks analyzed above and reanalyzing them. Finally, problem solving mechanisms suitable for each blocks are selected, so that the problem can be solved efficiently.

It is unfortunately necessary to reduce the complexity of the problems by some methods because systems for design problems are large-scale and complex and need to handle a large quantity of knowledge. For example we need an algorithm for dividing the problem into easier subproblems using multiple hierarchical levels of system abstractions. Similarly, handling the constraint network 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 of network as a flat structure. These hierarchical levels of this structured network correspond to functional blocks, or components.

In order to deal with constraint analysis for structured networks with hierarchical levels, the analysis phase is divided into two phases. The analysis is executed according to the tree description composed of part-whole relations and is-a relations of the system.

Phase one proceeds from the bottom up; it performs data flow analysis and reduction (merging) for each component and functional block.

Phase two instead starts from the root and proceeds from the top down towards the leaves: the dependencies of the constraint network are determined by reanalyzing constraint dependencies among functional blocks, and between functional blocks and components.

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.

After dependencies among constraints inside components and functional blocks are obtained by the constraint analysis, design plan generation proceeds towards the higher level of the hierarchy of the design object using the result of the analysis. As shown in Fig.7, constraint analysis generates this design plan so that the problem solving mechanism can be executed efficiently on the assumed architecture.

In other words, the generation of the efficient problem solver based on the problem solving strategies suitable to the given problem is required to identify and assume tasks necessary to solve the problem from the analyzed dependencies. These problem solving strategies are described in Section 2.2.2.2.

4.1.1.3 Extension of Constraint Analysis

Considering practical design problems, especially parametric design, there are various types of constraints to be satisfied. When the structure of the network of constraint is complicated for complex systems, an efficient constraint-based problem solving mechanism is not realized enough using only CLP languages, especially their constraint solver. Because this constraint solver handles the constraints in both the flat and global set.

To improve the execution speed, it is therefore necessary to separate the constraint network description of the design plan into parts that can be processed by local propagation and parts that require non-local propagation, using the concept of the structuring of the network with hierarchical level. To split up the constraint network we consider the constraint network as a graph and extract the tree description and loop description, and interpret and execute both the tree description and the loop description [18]. Thus, by splitting up the constraint network, each description is dealt with using the appropriate solver for constraint handling to reduce the search space of the problem. Furthermore, when the ordering of the assignment of values to parameters in this constraint network changes, the tree description and loop description generated from the original graph description also change. The former can be considered as a data flow graph.

An effective utilization of the structure of the problem space extracted using the constraint analysis will lead to the architecture suitable to knowledge-based systems. In this case, the extension of the constraint analysis considering this separation concept will lead to the improvement of constraint-based problem solving.

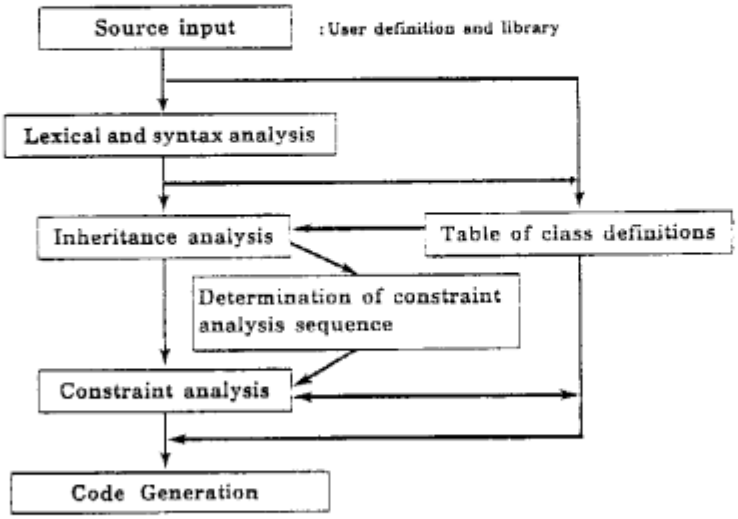

Fig. 6 Compilation Procedure

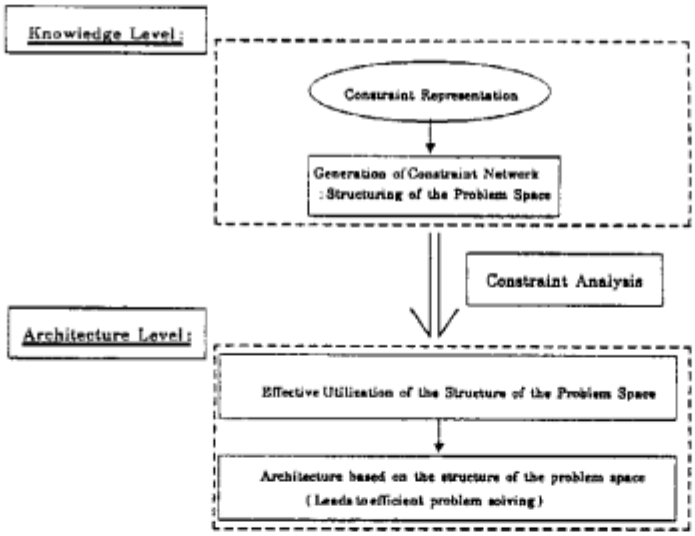

Fig.7 Role of Constraint Analysis

Furthermore, we will consider the following extensions of constraint analysis. We need to apply an incremental analysis (compilation) or interpretation function to this constraint analysis, to handle constraints that change dynamically, such as addition, deletion, and modification of constraints during a design process and to handle their preference. Furthermore, design plan generation requires scheduling of goals and subgoals according to the rough prediction of the necessary cost for problem solving during the constraint analysis considering an efficient problem solving.

In future, constraint analysis will not handle the static role of constraints; it must be able to interpret roles of constraint and directions of an information flow of constraints dynamically.

4.1.2 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. 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.8 shows the architecture of the building tool. An expert system building tool, MECHANICOT, is being developed [35]. MECHANICOT is a tool for a mechanical parametric design. It analyzes dependencies between structures of a design object and parameters, produces a design plan, and builds a specialized design expert system.

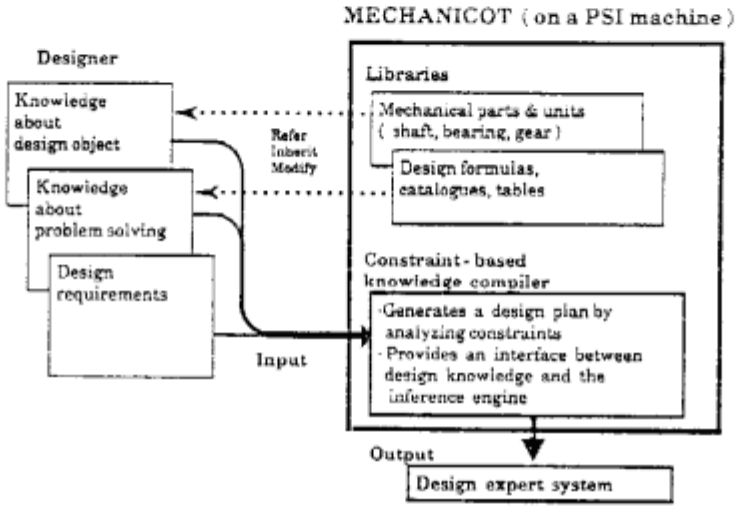

Fig. 8 Overview of MECHANICOT

4.2 Design Plan Generation of Gear Unit Design

Next, we will describe in detail the constraint analysis process, using a gear unit as an example.

4.2.1 Problem of Gear Unit Design

Fig.9 shows the main spindle head of a lathe. It consists of a main spindle to grip a workpiece and to rotate it, 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. The problem is to determine the dimensions of each part and find each part number by searching catalogues. The main spindle head of a lathe can be realized in different styles and consists of functional subsystems. Furthermore, each subsystem is implemented by configuring or combining the basic machine elements, composed of a pair of gears, the shaft, and the bearing, according to the design style [17]. For example, the design style for the power transmission unit between two parallel axes shows the reducer with two shaft units, whose components are the two shafts, bearings, and reducer. The power transmission unit of this reducer can be realized using a design style such as the gear-drive, belt-drive, or special-drive.

In this section we give as an example this gear unit. This problem is formulated in terms of the following specification parameters which describe the definition of domain-specific characterics. Fig.10 is a schematic description of a gear unit used in a reduction system of a main spindle unit. In this design, input parameters (design requirement) are the twisting moments of input and output shafts, shearing strength, tolerant torsion angle, and the number of input revolutions. The output parameters are gear ratio, pitch diameters of gears, shaft diameters, and the number of output revolutions.

The gear unit design can be considered as the problem of determining the design parameters so that the specification parameters and performance parameters are satisfied when the structure of the design object is assumed to be fixed. It is a typical example of parametric design. It is necessary to consider that there are two strategies for the realization or implementation of the physical components using standard parts and data, non-standard parts, and their combination in this problem. These strategies are determined depending on the design specifications or requirements and the parametric design is executed based on the strategies. Fig.11 shows an example of design knowledge of the gear unit.

4.2.2 Example Using MECHANICOT

Parametric design problems such as gear units can be considered routine design and can be formalized as well-structured problems. Fig.12 shows a constraint network of the problem formalized from the viewpoint of the concept of constraint. There are some alternatives in generating the values for the variables. Therefore, solving the problem can be efficient or inefficient according to the solution methods for problems, because there are several possible solution methods (strategies) for them. Although the problem can be formalized as a constraint network, the available strategies do not lead to an efficient solution. Functions to insure efficient problem solving must be provided to deal with practical problems.

MECHANICOT analyzes the relationships among components of a gear unit, generates the tree description showing part-whole and is-a relations, and determines analysis sequences according to the tree description.

Dependencies among constraints on components and functional blocks of the design object are analyzed according to the determined sequence.

Constraint analysis consists of two phases. In phase one, data flow analysis [1] is executed for the input shaft, output shaft, and the pair of gears that corresponds to leaf parts of the tree description. Next, traversing toward the root of the tree, data flow analysis proceeds in the functional block (gear unit) in the upper part of the tree description. After that, we perform data flow analysis for the gear unit and its components, including the input shaft, output shaft, and the 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 the pair of gears), and dependencies between these components, are reanalyzed; finally the dependencies in the whole system is analyzed.

Fig.13 shows an intermediate description of dependencies found by constraint analysis of a gear unit design. For example, constraints from structural relations, generator, tester, and filter are assigned to subgoals. Each constraint is interpreted as a function. When processing constraint, a subgoal is assigned to each constraint statement; a subgoal is also generated for each method. In the middle a data flow description of the design method for calculating a shaft diameter is described. In this design method, a twisting moment T, a shearing strength G, and a torsion angle theta are input. Shaft diameter D is the output.

Fig.14 shows a design plan generated according to this analyzed result. This figure consists of data flow descriptions for each gear and for a pair of gears. Subgoals are integrated into goals based on the input-output dependencies of parameters generated by a data flow analysis. Names are assigned to goals in exactly the same way as to subgoals.

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 included subgoals. We assume that the relationships among the goals and subgoals correspond to the hierarchical relationships of the design object shown in Fig.15; 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. Based on this assumption we generate a design plan using constraint compilation and can

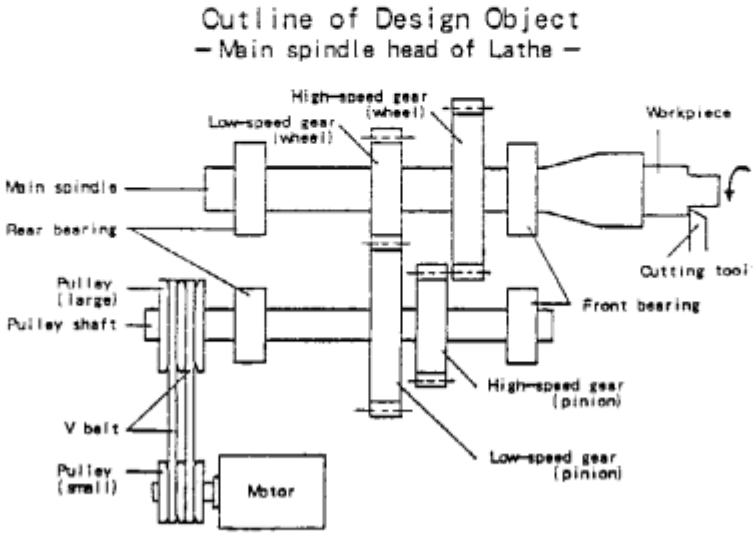

Fig.9 Outline of spindle head of lathe

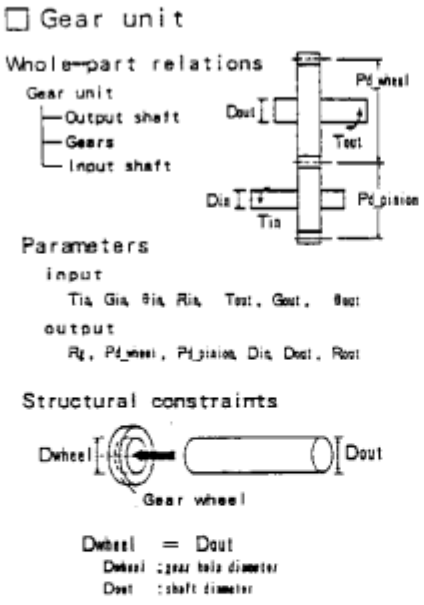

Fig.10 Schematic description of gear unit

Output shaft diameter : Dout

$$Dout = \{(32\,Tout \times 180 \times 10) / (\pi \cdot G \cdot \theta out)\}^{\frac{1}{3}}$$

G : Shearing strength

Gear ratio : Rg

$$Rg = Tout / Tin$$

Number of output revolution : Rout

$$Rout = Rin / Rout$$

Pitch diameter of output gear : Pd

Pd_gen : from min to max of discrete value of Pd

$$Pd\_gen \geq Dwheel + 7m = Pmin\_wheel$$

$$Pd\_wheel \leq 2000 / \pi \cdot Rout = Pmax\_wheel$$

Dwheel : Hole diameter of gear

$$Rg = Pd\_wheel / Pd\_pinion$$

Gear module : m

m_gen : {2.0, 2.5, 4.0, 5.0, 6.0}

Fig. 11 Design knowledge (formulas) of gear unit

finally obtain the plan written in ESP code shown in Fig.16. When the default strategy of the problem decomposition is fixed according to the assumption, it is more efficient to determine beforehand the ordering of the execution of the decomposed components.

5. Current State and Future Research

The first implementation of the design plan generation environment, including the knowledge compiler, is being carried out using the Extended Self-contained Prolog (ESP) language [7] on the personal sequential inference (PSI) machine [34].

MECHANICOT provides a design support environment where a designer can input and modify design requirements, easily design knowledge composed of a model of the design object and the design process, and where the design plan can be generated using constraints derived from that knowledge.

MECHANICOT is an automated system with no user interaction. It receives design requirements and design object representation 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 mechanism in the ESP language, such as unification and backtracking mechanisms.

So far, we only handle static and obligatory constraints. 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 constraints and dynamic constraints, such as the addition, deletion and modification of constraints during design, has not yet been investigated. Both static analysis for constraints and dynamic analysis, including constraint relaxation, are required to realize dynamic constraint handling, considering a current constraint compiler.

We have not yet realized a specific mechanism for constraint-based problem solving, and a constraint propagation is performed using the unification function in ESP. A constraint solver with a constraint propagation and relaxation mechanism is required to handle dynamic constraints.

Next, we will use the CAL *(Contrante Avec Logique)* language [30] as a language providing a constraint propagation to implement the design plan generation environment. In this case, a realization of the propagation and its control mechanisms of constraint-based problem solving utilize the inference mechanism of CAL language, especially the constraint solver.

After that, the extension of the constraint solver shown in Section 4., which adopts the constraint solver for linear inequality, will be also executed to improve the execution of the constraint-based problem solving.

In the future, we will adopt an interpretation approach as knowledge transformation technique. In this case, the design plan, whose design goals and methods were compiled during knowledge compilation, will be executed dynamically according to a design context or model determined by a design plan interpreter.

6. Conclusion

This paper has considered a method of design plan generation and interpretation using constraint compilation, a form of knowledge compilation. It has focused on the architecture of expert systems based on applying constraint-based problem solving.

We have demonstrated the technique on a mechanical component, a gear unit.

Our future research is to provide an environment in which designers can apply techniques used in existing compilers to mechanical design, and can construct design systems or tools.

For this purpose we clarify the architecture of expert systems for various routine designs such as circuit design, mechanical design, and configuration. We regard constraint-based problem solving as a new paradigm different from rule-based and frame-based paradigms, and for constraint-based problem solving we propose primitive tasks required to realize the architecture of expert systems for various routine designs.

Acknowledgments

I would like to express thanks to Mr. Satoru Terasaki, joint researcher of the Fifth Research laboratory, and other members of the Fifth Research laboratory: Mr. Takanori Yokoyama, Mr. Katsumi Inoue, and Mr. Hirokazu Taki for helpful comments. I would also like to thank Prof. Isao Nagasawa, Kyusyu

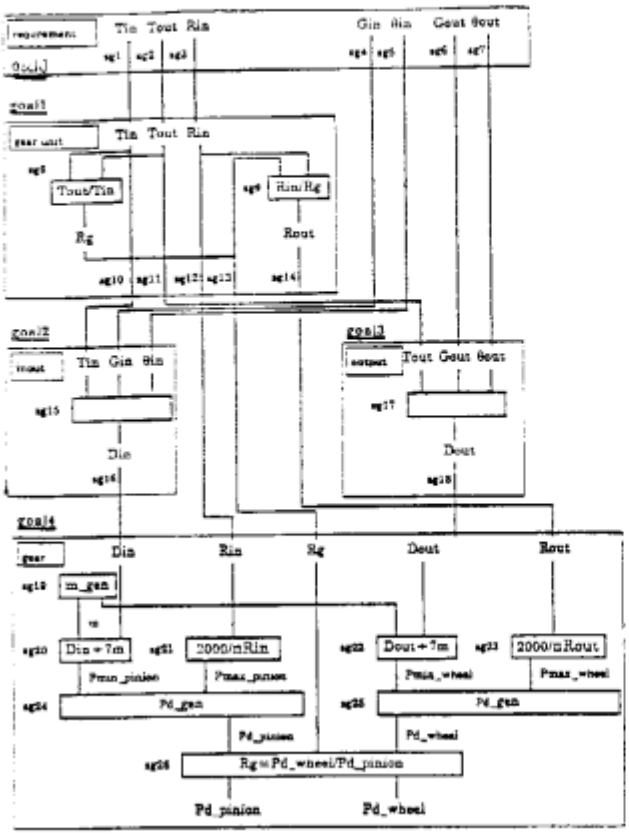

Fig. 12 Constraint Network Description

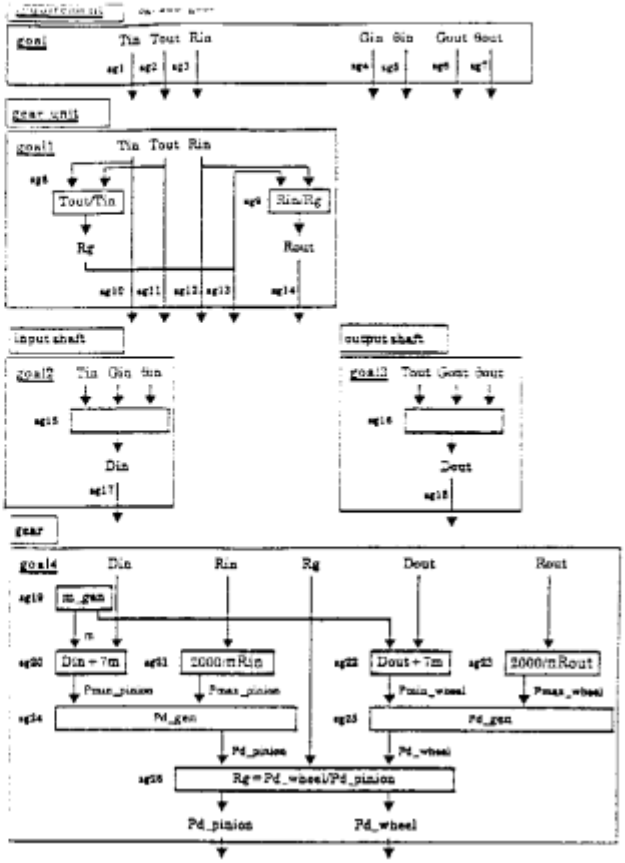

Fig. 13 Intermediate Description of Dependencies

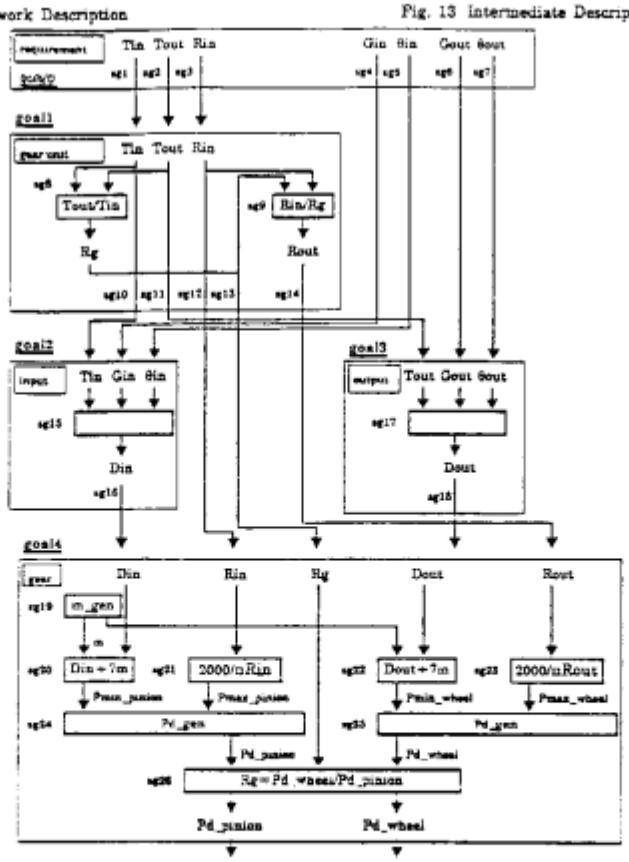

Fig. 14 Generated Design Plan

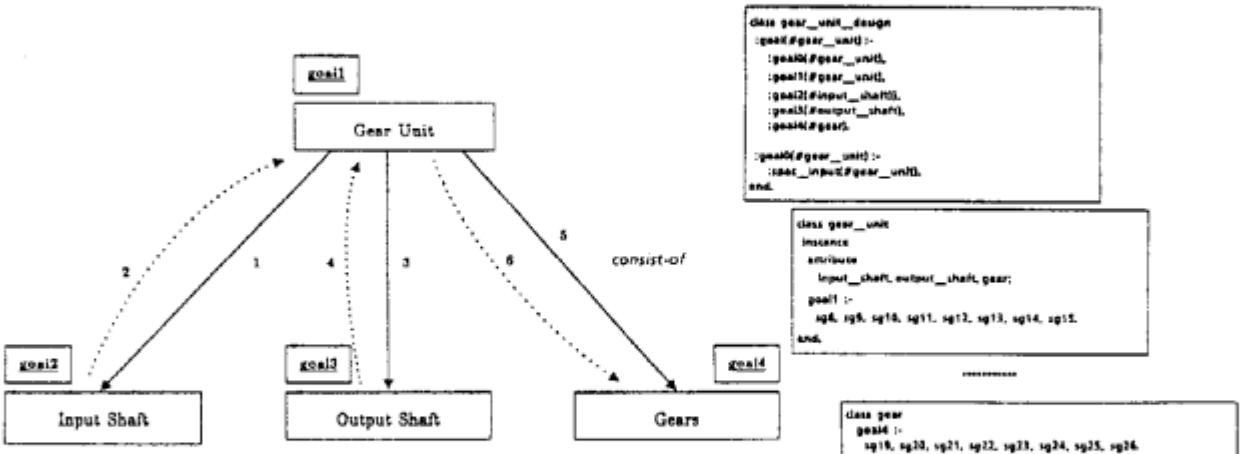

Fig.15 Relationship between goals according the consist-of relation

Fig.16 Design Plan written in the ESP code

Industrial Technology University, for useful suggestions and comments on needs of knowledge compilation for mechanical design. I would like to thank 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.

Reference

[1] Aho, A. V. and Ullman, J. D., Principles of Compiler Design, Addison-Wesley Publishing Company Inc., 1977

[2] Anderson, J. R., Knowledge Compilation: The General Learning Mechanism, Machine Learning, An Artificial Intelligence Approach, Vol. 2, R. S. Michalski, J. G. Carbonell and T. M. Mitchell (ed.), Morgan Kaufmann Publisher, Inc., 1986

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

[4] Borning, A., The Programming Language Aspects of ThingLab, a Constraint-oriented Simulation Laboratory, ACM Trans. on Programming Language and System vol. 3, 1981

[5] Brown, D.C. and Chandrasekaran, B., Knowledge and Control for a Mechanical Design Expert System, IEEE COMPUTER, 1986

[6] Chandrasekaran, B., Generic Tasks in Knowledge-based Reasoning: High-Level Building Blocks for Expert System Design, IEEE expert, 1984

[7] Chikayama, T., Unique Features of ESP, Proc. of International Conference on Fifth Generation Computer Systems, 1984

[8] Davis, E., Constraint Propagation with Interval Labels, Artificial Intelligence 32, 1987

[9] Dechter, R. and Pearl, J., The anatomy of easy problems: A constraint-satisfaction problem, Proc. of IJCAI-85, 1985

[10] Dechter, R. and Pearl, J., Network-based heuristics for constraint satisfaction problems. Artificial Intelligence, Vol. 34, 1987

[11] Descotte, Y. and Latombe, J.- C., Making Compromises among Antagonist Constraints in a Planner, Artificial Intelligence, 27, 1985

[12] Dincbas. M., Constraints, Logic Programming and Deductive Databases, France-Japan Artificial Intelligence and Computer Symposium 86, 1986

[13] Dixon, J. R., Howes, A., Cohen, P. R., and Simmons, M. K., DOMINIC I: Progress Towards Domain Independence In Design By Iterative Redesign, Proc. of ASME Computers in Engineering Conference, 1987

[14] 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), 1988

[15] Harris, D. R., A Hybrid Structured Object and Constraint Representation Language, Proc. of AAAI-86, 1986

[16] Heintze, N. C., Jaffar, J., Lassez, C., Lassez, J.-L., McAloon, K., Michaylov, S., Stuckey, P. J., and Yap, R. H.C., Constraint Logic Programming: A Reader, Fourth IEEE Symposium on Logic Programming, 1987

[17] Inoue, K., Nagai, Y., Fujii, Y., Imamura, S., and Kojima, T., Analysis of the Design Process of Machine Tools. - Example of a Machine Unit for Lathes - , ICOT-Technical Memorandum, 1988, (in Japanese)

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

[19] Kowalski, T. J. and Thomas, D. E., The VLSI Design Automation Assistant: Prototype System, Proc. of IEEE 20th Design Automation Conference, 1983

[20] McDermott, D., Circuit Design as Problem Solving, Artificial Intelligence and Pattern Recognition in Computer Aided Design, (ed. J. C. Latombe), North-Holland, 1978

[21] McDermott, J., R1: A Rule-based Configurer of Computer Systems, Artificial Intelligence, 19, 1982

[22] Medland, A. J., The Computer-based Design Process. 1., Engineering design-data processing I., Kogan Page Ltd, 1986

[23] Mittal, S., Dym, C. L. and Morjaria, M., A Knowledge-based Framework for Design, Proc. of AAAI-86

[24] Murthy, S. and Addanki, S., PROMPT: An Innovative Design Tool, Proc. of AAAI 87, 1987

[25] Nagai, Y., Terasaki, S., Yokoyama, T., and Taki, H., Expert System Architecture for Design Tasks, Proc. of Int'l Conf. on FGCS 88, ICOT, 1988

[26] 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, Vol. 4 No. 3, 1989

[27] Nagasawa, I., Design Expert System, IPSJ, Vol. 28, No. 2, (in Japanese), 1987

[28] Nicklaus, D. J., Tong, S. S., and Russo, C. J., ENGENIOUS: A knowledge-directed computer-aided design shell, Proc. of 3rd Conference on Artificial Intelligence Applications, 1987

[29] Rinderle, J. R., Implications of Function-Form-Fabrication Relations on Design Decomposition Strategies, Proc. of ASME Computers in Engineering Conference, 1986

[30] Sakai, K., Aiba, A., Sato, Y., Hawley, D., and Hasegawa, R., Constraint Logic Programming Langauge CAL, Proc. of Int'l Conf. on FGCS 88, ICOT, 1988

[31] Stefik, M., Planning with Constraints (MOLGEN: Part 1), Artificial Intelligence, Vol. 16, 1981

[32] Subrahmanyam, P. A., Synapse: An Expert System for VLSI Design, IEEE Computer, July, 1986

[33] Sussman, G. J. and Steel Jr., G. L., CONSTRAINT - A Language for Expressing Almost-Hierarchical Descriptions, Artificial Intelligence, Vol. 14, 1980

[34] 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, 1984

[35] Terasaki, S., Nagai, Y., Yokoyama, T., Inoue, K., Horiuchi, E. and Taki, H., Mechanical Design System Building Tool: MECHANICOT, JSAI, SIG-KBS, (in Japanese), October, 1988

[36] Tomiyama, T. and Hagen, P. J. W. T., Organizing of Design Knowledge in an Intelligent CAD System, in Expert Systems in Computer-Aided Design (ed. J. Gero), North-Holland, 1987

[37] Townsend, M., Discrete Mathematics: Applied Combinatorics and Graph Theory, The Benjamin Cummings Publishing Company, Inc., 1987