ICOT Technical Memorandum: TM-0504

TM-0504

Towards Design Plan Generation for Routine Design Using Knowledge Compilation - Focusing on Constraint Representation and Its Application Mechanism for Mechanical Design

by
Y. Nagai

May, 1988

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

Towards Design Plan Generation for Routine Design Using Knowledge Compilation

- Focusing on Constraint Representation and Its Application Mechanism for Mechanical Design

(Extended Abstract)

Yasuo NAGAI

Institute for New Generation Computer Technology,

4-28, Mita 1-chome, Minatoku, Tokyo, 108, Japan

E-mail: nagai%icot.uucp@eddie.mit, nagai%icot.jp@relay.cs.net

Phone: Tokyo +81-3-456-3192. Telex: 32964ICOTJ.

Keywords

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

Summary

The goal of this research is to propose design plan generation using the knowledge compilation technique. For this purpose, it is necessary to clarify the architecture of expert systems for various designs such as LSI design, mechanical design, and configuration. Especially, we regard constraint-based problem solving as a suitable new paradigm different from the rule-based and frame-based paradigms, and we need to propose primitive tasks for the constraint-based problem solving required to realize the architecture. [1] defines the routine design and describes the framework of the expert system architectures for design problems and necessary functions for solving them. Especially, the report focuses on and specifies constraint-based problem solving, consisting of constraint representation and its application mechanisms for mechanical design, in order to consider expert system architecture, including the modelling facility for mechanical routine design.

This paper considers the knowledge compilation of routine design and design plan generation using it, assuming the above framework including constraint-based problem solving.

1. Introduction

Knowledge 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 [2].

In mechanical engineering, there are many cases when design systems or tools such as DA systems and CAD systems are provided for each design object. In fact, the individuality that the design object possesses makes it difficult to abstract, arrange, and utilize the design systems or tools as the design environment, because the corresponding model and analysis method for this object often changes when the structure of the design object changes. In other words, among synthesis and analysis tasks used in DA systems or CAD systems, especially the analysis tasks for the performance and behaviour (function) prediction and evaluation are determined and provided based on the model of the design object. It is caused by the fact that a change in the structure of the design object, such as the geometrical characteristics, may result in a change in functions.

On the other hand, the models of VLSI design for the analysis of the performance and behaviour (function) and evaluation are fixed and formalized as the design methodology and utilized, and its structural changes have little effect on the function and behaviour. The hierarchical structure of the VLSI design, especially nested structure with the function, can be represented by combining the lower level functions so that the interactions among sub-structures of the design can be minimized.

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

To reduce inconvenience and inefficiency of existing design environments, it is necessary for the designers to provide an assistance environment in which the designer can apply the approach used in VLSI design to mechanical design and can construct design systems or tools easily.

In the following sections, we introduce knowledge compilation for design problems, design plan generation using knowledge compilation for routine mechanical design, a consideration of extended constraint solver, and cuurent state of our research.

There are various kinds of knowledge in design problems, for example, knowledge about the design object and knowledge about the design process. Knowledge about the design object includes knowledge about the design style to determine the structure, functional and physical components, and the analysis and evaluation methods. This knowledge is represented as the design object model and can be regarded as *deep knowledge*.

The design plan by which the design specifications can be satisfied is generated by compiling *deep knowledge* and problem-solving heuristics [21].

The synthesis and analysis tasks in the design are interpreted and executed according to this design plan generated by knowledge compilation on the predetermined system architecture.

2. Knowledge Compilation for Design Problems

2.1 Comparison of Mechanical Design with VLSI Design

In mechanical design, it is difficult to modularize the design object because the geometrical information, as 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, since the geometric features or form description depend on the functional description or fabrication information. It results from the dynamic creation of the model about the components of the design object as the device or product.

In contrast with VLSI 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 behaviour satisfies the specifications and it is necessary to consider the analysis task.

Consideration of strategies for the decomposition of the problem or specification at each level is required to discuss the design process model for routine design. These strategies are not always formalized clearly and not applied in mechanical design as in VLSI design [3]. In circuit design, the design process at each level is formalized so that the modularity, simplicity and applicability of the design task can be enhanced. It is very important for the strategies of the problem decomposition to consider the interactions between functional description and physical description for both circuit and mechanical design. In mechanical design, most interactions are caused by the execution of the constraint representation composed of the functional concept, based on the physical laws, and feature concept of the form, such as topology or geometry of the design object at functional or physical level, and it is necessary to deal with the design problem by investigating the degree of decomposition of the problem or specification. In this case, it is assumed that the structure of the design object at the functional level has already been decided.

In VLSI design, the silicon compilation technique is being investigated [4,5]. This technique transforms an input programming language into a circuit to be implemented in device technologies as an output by the compiler. It mainly consists of step-wise refinement, which assigns the decomposed sub-problem into the circuit using the library modules and generates the circuit, and the optimization for the resolution of the trade-off between input specification and resource.

In mechanical design, when considering the parametric design, the handling of the constraint representation is required more than the step-wise refinement and optimization utilized in the silicon compiler.

This is the significant difference between VLSI design and mechanical design. However, it is expected that the techniques and framework of silicon compilation will be applied to the framework of knowledge compilation for the mechanical design.

2.2 Constraint handling in Knowledge Compilation

2.2.1 Classification of Constraints

(a) General (domain-independent) constraints for routine design

Constraints are classified according to the following characteristics.

1) Classification according to the generation method

Constraints may be classified according to whether they are generated in statically or dynamically. Static constraints are specified in advance, and are constant and unchanging. On the other hand, dynamic constraints are imposed depending both on interactions with the user and on the system; they tend to change, with their range of applicability varying. Such constraints may be interpreted as incomplete knowledge, and, in order to manage changes in truth in the knowledge base accompanying changes in constraints, the functions of the Truth Maintenance System (TMS) [6] and Assumption-Based Truth Maintenance System (ATMS) [7] are necessary.

2) Classification according to importance

Constraints may be classified according to importance into obligatory or requisite constraints, and suggestive constraints. When such a distinction is made, not all the constraints are selected and executed on an equal basis. That is, the importance of a constraint may depend on the context, the time, or another concept. All obligatory constraints must be satisfied, and these are given explicitly. Suggestive constraints are also referred to as weak constraints, and are used as guides in choosing the optimum branch at a node in the search tree. Such constraints may be described in rule form, and are given priorities and other attributes.

3) Classification according to scope

Constraints may also be classified according to whether they apply locally or globally; this distinction is used in evaluating states in the search space. 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. For instance, when the solution space is divided and searches performed, this is equivalent to taking into account all those constraints which have been applied to states leading up to the present state, or to evaluating different parts of the solution space relative to each other.

4) Classification according to the propagating variable information

Constraints may be classified according to the propagating variable information, that is, depending on whether the constraint variables propagate, or whether the constraint propagates over the interval bound in which the variable can take on a value or values. At present, the constraint logic programming system [8,9,10], CONSTRAINT system [11], and most other constraint systems [12] handle only constraints in which values are propagated.

Constraints which propagate over interval bounds in which variables can take certain values, are described using inequalities, and the variables of the constraint are not constant; these constraints propagate over the interval bound as a label. Most design problems include sub-problems which can be solved using the method of existing operations research; it is extremely important that the architecture enable functions to operate in a unified framework based on constraint propagation in labels with interval bounds [13].

There are some combinatory possibilities of the above classified constraints.

(b) Domain-specific constraint for routine design

There are various domain-specific constraints for routine design. These constraints are related to the simplified design process composed of the conceptual design, fundamental design and detailed design. They are described in this section.

In conceptual design, the description of performance and cost from the definition of the requirement and specification description can be regarded as constraints. In fundamental design, the ways of the mapping or instantiating the functional description to the real or physical world and the environments for its realization can also be regarded as constraints. For example, when the 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 form and structure representation, and knowledge regarding the design style for the configuration of the components and relations among them can be regarded as the constraints. In this case, because the model depends on the design object, the constraints derived from the model depends on the design object.

In particular, structural constraints should be considered in routine design. Structural constraints are reflected in terms of the design style, and specifications, and requirements at each abstract level of the design, and determine the structural decomposition, partition, and design style at lower design level. In hierarchical design, it should be noted that the constraints are propagated to a lower design level. The design style constraints decide the structure of the design object and the problem decomposition at lower design level. Constraints are partitioned through the structure of the design object and decomposition of the design problem. For example, there are the implementation constraints i.e., technology-dependent constraints at the implementation level.

2.2.2 Constraint-based Problem Solving

The architecture and necessary problem solving mechanism of the expert system for routine design are described. Researches has been conducted on architectures consisting of primitive tasks for routine design, called design generic tasks [14,15]. These architectures provide the ability to structure knowledge for the various design descriptions and problem solving for the design to reduce the gaps between the necessary functions for the task in the design process and the functions supported by expert system building tools. However, they do not support the modeling facility and it seems that they are insufficient for this generic task approach to handle the constraint representation. Therefore, the architecture including the constraint representation and its application mechanism, and the modeling facility, are investigated. This constraint representation is proposed as a new paradigm for knowledge representation and the application mechanism as a new paradigm for the architecture of routine design expert systems.

The application mechanisms for constraint represenatation in routine design expert systems are defined as constraint-based problem solving, and are described in detail. Furthermore, constraint-based problem solving focuses on the failure recovery handling mechanism. Failure recovery handling consists of constraint satisfaction problem handling and the redesign problem of partial design. Modeling corresponds to the formalization of various descriptions for design knowledge and the method of handling design knowledge by trial and error. The knowledge derived based on this modeling can also be regarded as deep knowledge. The following constraints are generated, derived, modified, or deleted from modeling during the design process.

The constraint-based problem solving mechanism is described according to the above constraint classification through matching the design process model for a routine design to the design system or tool.

The functions required for constraint-based problem solving are listed below.

1) A function for constraint propagation and its control

In the process of satisfying constraints, and when values are assigned to variables of the constraint, the values of other constraint variables may be determined by the former variable; this is the mechanism of constraint propagation. Such a mechanism must take into account both cases considering local constraint propagation and cases where the problem cannot be solved by local constraint propagation only. A typical example of the former is the propagation method using data-flow analysis introduced in the CONSTRAINT system. An example of the latter case is the variable elimination method of simultaneous equations. In particular, when using propagation methods based on data-flow analysis, the trade-off between constraints, such as occur by regarding the TMS as a constraint satisfaction problem (CSP) [16], may result when the propagation is not always sufficient. Clearly, a strategy for controlling constraint propagation is needed. Interactions between constraints and the least commitment of constraints are also indispensable for realizing the constraint propagation. For instance, in practical design problems, if we consider design by step wise refinement, interaction between constraints applying to sub-problems which are solved separately are extremely important. One approach to the problem of constraint interactions is minimizing interactions between sub-problems, such as that adopted in the MOLGEN system [17,18]. This is referred to as the principle of least commitment; by delaying constraint evaluations as far as possible, refinements according to the design plan are executed, and evaluations are performed when necessary.

2) Constraint relaxation and selection

Relaxation and selection are applied to weak constraints. Relaxation of a constraint is equivalent to searching for alternatives to the specified constraint. That is, at the failure stage, when a constraint has not been satisfied, alternative constraints, at the same or a lower level, are searched for. Selection involves the choice of a constraint when there are two or more competing constraints, and is regarded as constraint interpretation. In this way, it is thought that constraint relaxation and selection can be formulated as a planning problem [19].

3) Preservation and management of dependency relations

In processes where the values of constrained variables are propagated through the execution of constraint propagation mechanisms, when contradictions in variable values arise, the preservation and management of dependency relations among constraints, variables, and constant values are deemed important to resolve such contradictions and to explain the propagating values [20].

4) Monitoring mechanism for constraint evaluation

A monitoring mechanism for constraint evaluation should not be omitted from any problem-solving mechanism which relies on constraint representations. It manages constraint checks and ensures consistency, and is to some extent realizable using demons or attached procedures.

2.2.3 Role of this mechanism in the design process

This section considers constraint-based problem solving relative to the design process. The fundamental task at each design 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.

Mechanical design that mainly belongs to parametric design can be regarded as the generate & test + failure recovery (+ optimization + analysis & evaluation) paradigm.

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 configurational or architectural knowledge about the design object, is indexed by the requirement or specification of the design, and can be regarded as a constraint. The refinement, optimization, and analysis and evaluation tasks are selected and executed according to this constraint. The decomposition of the requirements or specifications of the design are executed by applying this design style constraint.

It is assumed that problem decomposition can transform or map the sub-problem to the component or assembly. In this case, there are two methods for problem decomposition: one is problem decomposition into sub-problems with interactions, and the other is problem decomposition into independent sub-problems. It is important for the former to consider the relations among the compositions at the same level, and for the latter to consider the relations among the components and sub-components.

Refinement transforms the divided specification into structural representation composed of the components and relations among them. These relations among components can be regarded as a constraint. Constraints on the component attributes are particularly important. For example, the propagation mechanism of constraints of the component attributes, in decomposition into interacting sub problems is different from that in decomposition into independent sub-problems. The former mechanism propagates the interactions among sub-problems as the constraints, and the latter propagates the constraints upward or downward according to the hierarchical representation of the design object when there are no interactions among independent sub-problems.

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

2.3 Knowledge Compilation for Design Problems

Knowledge compilation techniques are being investigated in many problem areas such as diagnostic and machine learning problems. Knowledge compilation for design problems, especially mechanical design, is defined in this sections. This knowledge compilation for design problems is a technique that transforms the input design specifications to the design plan, assuming that the structure of the design object has been determined.

The input of a knowledge compiler is design specification, including the functional description and constraints such as performance and resource limitation, in the form of parameter description.

The knowledge compiler determines the structure of the design object, and the analysis and evaluation method, 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, and its structural knowledge, and public knowledge such as the catalogues and parts standards.

The relationship among the components for the implementation of the design object, the problem decomposition and refinement method, the relationships among the constraints and parameters, and relationships among the components or parts and attributes are analyzed and determined.

The design plan, where 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, and analysis and evaluation method of the design object, are generated using the above analysis result in the knowledge compiler. It is generated on predetermined architecture such as the generate & test + failure recovery paradigm and can be considered as the program for the design that is, the design system. In this case, the optimization of the generated design plan, considering the problem decomposition composed of the independent sub-problem with no interaction and the sub-problem with interactions, must be executed.

The interpretation of the design plan corresponds to the execution of the design system and is executed by the design plan interpreter.

The environment of this design plan generation using the knowledge compilation technique and its interpretation may be considered as support for a CAD system construction environment customizable by the designer.

3. Design Plan Generation Using Knowledge Compilation for Routine Design in Mechanical Engineering

3.1 Constraint Representation in Mechanical Design

In the problem of V-belt drive design, various constraint representations are utilized and applied. They consist of most of the knowledge about the design object, composed of knowledge about the design style or configuration, the functional unit, block, or component, and the physical components.

Therefore, there are various application mechanisms for each constraint representation, the problem solving mechanism for each constraint representation. However, existing constraint handling systems facilitate only general-purpose problem solving mechanisms based on the same criteria about constraint representation and are insufficient for these facilities to deal with routine design problems. Here, the problem-solving mechanisms for static or dynamic constraints, local or global constraints, and obligatory or suggestive constraints described above are needed in order to deal with routine design problems for the machine, parametric design.

In mechanical design, the function concepts based on the physical laws, and feature concept of the form, such as topology or geometry, of the design object at the functional or physical level can be described as mathematical formulas, and they can be considered as constraints.

Apart from the above constraints, consideration of the handling of the equality and inequality as constraint representation, in particular, is required in parametric design. There are two methods of determining the parameters by the application of this constraint representation. One is when the parameters on the constraints are given and the values that parameters can take on are restricted explicitly. This is considered as the assignment of values to parameters by checking the values using constraints. The potential values of constraint parameters on constraints can be considered as a set composed of discrete and pre-enumerated values.

The other method is when the constraint parameters are given, but the values that the parameter can take on are not specified explicitly. This assignment of values to parameters is executed by considering constraints as formulas for calculation.

Furthermore, handling of the inequality depends on the characteristics of the problem. For instance, the inequality may be interpreted and handled as an equality, especially in substitution formulas.

3.2 Assumed Architecture and Problem Solving Mechanism

This problem for routine mechanical design belongs to modification design or edit design, and is a typical example of mechanical design.

Its tasks consist of the decision on the artifact structures and the structural parameters, without the optimization and transformation of the artifact structures.

The artifact structure in routine design is determined by combining the components or is determined according to predefined design styles of the artifact. In this case, the artifact structures are determined by retrieving the appropriate design style from the knowledge base.

The components are implemented using the standard parts by looking up them in catalogues or using non-standard parts from the design. Most of the selection strategies of standard or non-standard parts for component implementation are described in the specifications or requirements. They mostly depend on the trade-off of the performance and cost.

The necessary architecture for this design can be formalized as the generate & test + failure recovery (+ optimization + analysis & evaluation) paradigm.

The problem-solving primitives are the generator, propagator, tester, and failure recoverer.

The generator assigns values to parameters or assigns functional components to the components for implementation. This parameter can be classified in one of two ways according to the type of the value; one takes the continuous value, and the other takes the discrete value. The former assigns parameters of the attributes by local modification based on the predetermined component. The latter assigns parameters by retrieving the standard parts for implementing components from the catalogue, a table look-up method.

The propagator assigns the values to the parameters by using the active evaluation of the constraint and propagation of 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, it can also be considered as the equality and handled by the tester.

The failure recoverer modifies the attributes of the components locally using the advice mechanism and plans problem decomposition. The advice mechanism can be considered as the repair of partial or local design using the heuristics about the attributes of the components. It uses the above generator and propagator as primitives. The obligatory and suggestive constraints must be handled in failure recovery handling. The advice mechanism using the selection and evaluation of the constraint is executed to the obligatory constraint. For suggestive constraints, planning, such as a compromising algorithm, is required in order to relax and select this constraint. This is a mechanism that satisfies as many constraints as possible, too.

The constraint-based problem solving in the generate & test + failure recovery architecture corresponds to the execution of the representation generated by compiling various items of design knowledge composed of the design specification and knowledge about the design object based on the fundamental tasks of the design process, such as planning, problem decomposition, and refinement. This representation is derived by compiling the constraint representation such as the various formulas about the features of the functional or physical environment by assuming this structural description and can be considered as the constraint network, when the structure of the design object, configuration of the components, is fixed.

3.3 Design Plan Generation using Knowledge Compilation in Mechanical Engineering

The power transmission mechanism for a lathe can be realized in different styles. A lathe is a machine that removes metal from a workpiece by gripping it securely in a holding device and rotates it under power against a suitable cutting tool. This machine consists of a functional sub-system or unit such as the shaft (spindle) system, or power transmission unit. Furthermore, each system component or unit is implemented by configuring or combining the basic machine elements, composed of a spur, belt, or other type of gear, the shaft, and the bearing according to the design style. For example, the design style for the power transmission between two parallel axes shows the reducer with two shaft units, whose components are the two shafts, bearings, and reducer. The power transmission or drive mechanism of this reducer can be realized using a design style such as the gear-drive, belt-drive, or special-drive. Furthermore, there are four general types of belt in the belt-drive, each with its own special characteristics, limitations, and special-purpose variations for different applications: flat belts, V belt, film belts, and timing belts. In shaft unit design, it is important to consider the bearing types and bearing configuration, for example, the bearing mount type; the type and configuration can be also regarded as a kind of design style.

In this section, an example of this power transmission unit, using a V-belt drive unit, is explained [22].

It results from the analysis of the design process of machine tool, especially, the part of the machine unit in the lathe. The purpose of the formalization of this problem is to analyze how human design proceeds and to consider the system architecture for intelligent design assistance and automated design.

This problem is formulated by the following specification parameters, design parameters, and performance parameters [23].

The specification parameters describe the definition of domain-specific problems. They are: drive speed, load speed, load power, design horse power to be transmitted, center distance, desired life time, and pulley diameter.

The design parameters that the designer must determine are pulley sizes, belt types, belt length, and number of belts.

The performance parameters used to evaluate the quality factors about the result of the design are load speed, life, belt velocity, lateral force on the shaft, and cost.

The V-belt drive 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 such as the components and relationship among them 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 the standard parts and data, the 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.

In this case, we assume that the relationships among the design goals and subgoals correspond to the hierarchical relationships of the design object; the relationships among 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.

Design plan generation using knowledge compilation is executed based on this assumption. Furthermore, when the default strategy of the problem decomposition, namely the composition decomposition, is fixed according to the assumption, it is desirable for the efficient execution of the problem-solving mechanism to determine the ordering of the execution of the decomposed components beforehand. In future, it is desirable that the design plan, whose design goals and methods were compiled during knowledge compilation, will be interpreted and executed dynamically according to a design context or model determined by a design plan interpreter, including a goal-driven scheduler.

3.4 Suitability of Generated Design Plan Description for Constraint Logic Programming

A language scheme called constraint logic programming (CLP) has been proposed [8,9,10]. 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 inequations 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 admit the advantage that this program is performed declaratively by evaluation and assignment of values to variables more flexibly than the PROLOG program is performed intuitively. Convential 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 constitutes the constraint network description, including the design knowledge, the problem-solving heuristics and problem-solving primitives of the predetermined architecture. The mechanism of the interpretation and execution of this design plan mostly depends on the mechanism of the constraint propagation. Constraint propagation and its control is most important function for constraint-based problem solving. Constraint propagation consists of local propagation and non-local propagation. CLP languages can handle this propagation mechanism easily, because the local propagation and non-local propagation can be handled by the constraint solver, which can be viewed as the generalization of unification. Furthermore, the CLP's three features, logical, functional, and operational features are preserved as conventional logic languages. Thus, the CLP languages scheme is suitable for the above constraint-based problem solving. In first version of the knowledge compiler environment, the design plan description is translated into this language. In this case, the problem solving primitives for constraint-based problem solving other than that described in Section 3.2 will be provided and extended on this language, because the CLP languages are considered as the programming languages.

4. Consideration of Extended Constraint Solver

Considering the practical design problem, especially parametric design, there are many constraints. The constraint-based problem solving handles many constraints and is a burden on the constraint solver of CLP languages. because this constraint solver handles the constraints kept in the flat and global set.

Therefore, to improve the execution of problem solving, it is necessary to separate the constraint network description of the design plan into processing parts by the local propagation method and processing parts by the non-local propagation method.

To separate the constraint network, we consider the constraint network as a graph and extract the tree description and loop description from this graph description, and interpret and execute the tree description and loop description whose structures are cyclic [24].

Thus, from the separation of the constraint network, each extracted description is dealt using the appropriate solver for constraint handling for the reduction of the search space of the problem.

In this case, when the ordering of the assignment of values to parameters in the constraint network changes, the tree description and loop description generated from the original graph description also change and can be considered as a data flow graph including cyclic descriptions. Furthermore, the result of the tree search executed according to the generated tree description also changes.

The extension of the constraint solver considering this separation concept will lead to the improvement of constraint-based problem solving.

5. Current State

The first implementation of the design plan generation environment, including the knowledge compiler, is being carried out using both the Extended Self-contained Prolog (ESP) language and CAL *(Contrante ave Logique)* language [10] on the personal sequential inference (PSI) machine. In this case, the propagation and its control mechanisms of constraint-based problem solving almost utilize the mechanism of CAL, especially the constraint solver.

After that, the extension of the constraint solver shown in the section 4. will be executed to improve the execution of the problem solving.

6. Conclusion

This paper considered a method of design plan generation and interpretation using knowledge compilation techniques. It focused on the architecture of expert systems, including a modeling facility for routine design, proposed by focusing on constraint-based problem solving, and composed of constraint representation and the application mechanism.

Machining tools, specifically, a power transmission unit for a lathe, were selected as an example.

Our future research is to provide an assistance environment in which the designers can apply the approach used in VLSI design such as silicon compilation to mechanical design, and can construct design systems or tools.

For this purpose, it is also necessary to clarify the architecture of expert systems for various routine designs, such as VLSI design, mechanical design, and configuration, by regarding constraint-based problem solving as a new paradigm different from rule-based and frame-based paradigms, and to propose primitive tasks for constraint-based problem solving required to realize the architecture of expert systems for various routine designs. This paper can be considered as the first proposal in this research.

Acknowledgements

I would like to thank Mr. Yuichi Fujii, Chief of the Fifth Research laboratory for his encouragement in my research and other members of the Fifth Research laboratory for helpful comments. I would also like to thank Prof. Isao Nagasawa, Kyuusyuu University, for useful suggestions and comments on needs of knowledge compilation for mechanical design. 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] Nagai, Y., Towards an Expert System Architecture for Routine Design - Focusing on Constraint Representation and an Application Mechanism for Mechanical Design, 3rd Int'l Conf. on CAD/CAM Robotics & Factories of the Future, 1988, (to appear)

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

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

[4] Kahrs, M., Critics as optimization operators in a silicon compiler, Knowledge Engineering in Computer-Aided Design, Gero(ed.), North-Holland, 1985

[5] Shirai, K., Nagai, Y. and Takezawa, T., Functional Level Design System For Digital Signal Processors, Procs. of the IFIP TC 10/WG. 10.5 Int'l Conf. on Very Large Scale Integration, 1985

[6] Doyle, J., A Truth Maintenance System, Artificial Intelligence vol. 12, 1979

[7] de Kleer, J., An Assumption-Based TMS, Artificial Intelligence vol. 28, 1986

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

[9] 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

[10] Sakai, K. and Aiba, A., CAL: A Theoretical Background of Constraint Logic Programming and Its Applications, ICOT-Technical Report, 1988, (to appear)

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

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

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

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

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

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

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

[18] Stefik, M., Planning and Meta-Planning (MOLGEN: Part 2), Artificial Intelligence, Vol. 16, 1981

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

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

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

[22] 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, (to appear) (in Japanese)

[23] 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

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