A Formal Approach to Structural Design Automation Using Predicate Logic

Sivand Lakmazaheri* William J. Rasdorf †

Abstract

The common approach to engineering software development involves the identification of relevant concepts, relations between concepts, and strategies for manipulating concepts and relations in an informal manner. We believe that although the informal approach to the identification and utilization of engineering knowledge may be sufficient for a certain class of problems, it fails to systematically support the development and study of complex automated (intelligent) engineering systems. Therefore, a formal approach is needed for such developments and studies.

In this paper a formal theory that captures the knowledge associated with the physical behavior of structural systems is formulated. The physical behavior of structural systems is defined as the response (member displacements and member forces) of the structure to its environment (applied loads and boundary conditions). Predicate logic is used as the underlying formal language of the theory and the resolution theorem proving strategy is used for reasoning about the theory. The use of the formal theory for the analysis and synthesis of structural systems is illustrated and several research areas where the formal approach shows promise are discussed.

1 Introduction

Computer technology and the science of computing have and will continue to influence engineering problem-solving automation in two ways. First, they provide a variety of programming techniques, languages, and environments (herein called the computing tools) for developing engineering software systems. Second, the science of computing helps formalize engineering knowledge and helps systematize and automate engineering problem solving. That is, it provides a means to formally represent, analyze, and effectively utilize engineering knowledge for developing automated (intelligent) engineering systems.

*Research Assistant, Civil Engineering Department, North Carolina State University, Raleigh, North Carolina 27695

†Associate Professor, Civil Engineering Department, North Carolina State University, Raleigh, North Carolina 27695

The engineering research community has directed most of its resources and efforts towards advancing engineering problem solving through the use of the computing tools [Law89] without paying much attention to the systematic formalization and effective utilization of engineering knowledge. However, the development of truly automated (intelligent) systems is an extremely difficult problem, one which cannot be truly solved without addressing the core problem, namely the identification, representation, and effective utilization of knowledge for automated reasoning.

We argue that knowledge formalization and its effective utilization are important for developing and studying automated (intelligent) engineering systems. This is true because even the most sophisticated computer hardware or the most advanced programming environment cannot solve an engineering program unless it is programmed to do so. To program, one has to have a clear understanding about the problem and how it should be solved. That is, one needs to know what are the basic concepts and relations between concepts (knowledge), how they should be represented, and how they should be manipulated in order to solve the problem. Simply put, one needs to represent and study engineering knowledge before one can develop a system that utilizes the knowledge for solving an engineering problem.

The common approach to engineering software development involves the identification of relevant concepts, relations between concepts, and strategies for manipulating concepts and relations in an informal manner. We believe that although the informal approach to the identification and utilization of engineering knowledge may be sufficient for a certain class of problems, it fails to systematically support the development and study of complex automated (intelligent) engineering systems. Therefore, a formal approach is needed for such developments and studies.

1.1 A Formal Approach

Systematic and scientific progress in many disciplines require the invention and use of appropriate mathematical apparatus with which concepts can be expressed and unified [Genesereth88]. Mathematical logic is one such apparatus. Logic is a branch of mathematics that arose from a concern with the nature and the limits of rational or mathematical thought, and from a desire to systematize the modes of its expressions [Barwise85]. As a branch of mathematics, logic is concerned with the language for defining mathematical objects (structures) and the laws for reasoning about them [Barwise77].

Mathematical logic is *foundational* to knowledge formalization and reasoning automation in the field of Artificial Intelligence [Hayes77] [Moore85] [Nillson89]. An intelligent agent capable of reasoning in a particular domain must have adequate knowledge about objects and their relations in the domain, and must be able to employ a reasoning technique to interpret and manipulate its knowledge.

In general, mathematical logic provides a rigorous symbolic language (formal language) for representing knowledge, a well-defined method for assigning meaning to the expressions of the language, and a mechanical method for manipulating the expressions of the language [Frost86].

A formal language is composed of a vocabulary and a set of syntactic rules for expressing the valid statements of the language. The subset of the valid statements of a formal language that captures the knowledge of interest in a particular domain is called a *formal theory*. The statements of the formal theory are called *axioms*.

Reasoning in a domain can be modeled based on reasoning about the formal theory associated with the domain. Reasoning about a formal theory lends itself to formulating a valid expression using the underlying formal language of the theory and proving the truth or falsity of the expression. Such an expression is called a *theorem* and the task of proving its truth value is known as *theorem proving*.

In this paper a formal theory that captures the knowledge associated with the physical behavior of structural systems is formulated. The physical behavior of structural systems is defined as the response (member displacements and member forces) of the structure to its environment (applied loads and boundary conditions). Predicate logic is used as the underlying formal language of the theory and the resolution theorem proving strategy is used for reasoning about the theory.

1.2 Organization

In Section 2 predicate logic and the resolution strategy are discussed. Section 3 provides a conceptual view of the modeling process for describing the behavior of structural systems. In Section 4 the formal theory is given. Section 5 illustrates the use of the theory for reasoning about the behavior of several truss structures. Section 6 provides a summary of the work. Finally, Section 7 provides a discussion of the practical implications of the formal approach.

2 Background

In this section predicate logic and the resolution theorem proving strategy are briefly described.

2.1 Predicate Logic

Predicate logic is an expressive formal language for representing knowledge. The vocabulary of the language is composed of a set of constant symbols, a set of variable symbols, a set of predicate symbols, a set of function symbols, two or more logical operators (e.g., $\neg$, $\wedge$, $\vee$, $\Rightarrow$), and the universal ($\forall$) and existential ($\exists$) quantifiers.

The valid expressions of the language are called well-formed fomulas (*wffs*) or clauses and are inductively defined as follows [Chang73].

- Constant and variable symbols are terms.

- $f(t_1, t_2, \cdots, t_n)$ is a term if t_1, t_2, $\cdots$, t_n are terms and f is a function symbol.
- $p(t_1, t_2, \cdots, t_n)$ is an atomic formula if t_1, t_2, $\cdots$, t_n are terms and p is a predicate symbol.
- An atomic formula is a *wff*.
- $\neg A$ is a *wff* if A is a *wff*.
- $A \wedge B$, $A \vee B$, and $A \Rightarrow B$ are *wffs* if A and B are *wffs*.
- $\forall_x A$ and $\exists_x A$ are *wffs* if A is a *wff* and x is a variable in A.

To extend the expressive power of predicate logic to be of greater use in an engineering domain, the above definition can be extended as follows [Jaffar86a].

- t is an arithmetic term if and only if t is a real number.
- If t_1 and t_2 are arithmetic terms then $t_1 + t_2$, $t_1 - t_2$, $t_1 * t_2$, and t_1/t_2 are arithmetic terms.
- An arithmetic term is a term.

This extension, as suggested by Jaffar and his co-workers [Jaffar87a] permits the representation of arithmetic constraints in the framework of predicate logic.

2.2 Resolution

The resolution strategy is a mechanical procedure for determining the truth value of a given theorem based on the set of axioms of a theory. The resolution proof procedure is iterative in nature. In each iteration two *wffs* (clauses) are resolved into one. The procedure yields the empty clause ($\{\}$) when the theorem is a logical consequence of the thoery. To resolve two clauses, a generalized modus ponens rule and a substitution procedure are used [Loveland78]. The generalized modus ponens rule is "*from* $(\neg A \vee C)$ *and* $(A \vee B)$ *deduce* $(B \vee C)$." The substitution procedure involves finding a substitution for the variables such that two unit clauses are made syntactically identical. For example, consider the two unit clauses $A(2, f(x))$ and $A(x, y)$; a substitution, θ, that makes the two clauses identical is $\theta = \{x = 2, y = f(2)\}$. That is, for $x = 2$ and $y = f(2)$ the two clauses become syntactically identical. The procedure for finding a substitution is called *unification*.

Unifying two terms (e.g., A and B) involves satisfying the constraint A = B where both terms are non-arithemtic. However, unification cannot handle arithmetic constraints. For example the constraint $A * B = C + D$ where A, B, C, and D are arithmetic terms cannot be handled using unification. Therefore, to handle arithmetic expressions unification needs to be generalized to a constraint satisfaction strategy. This generalization was addressed by Jaffar and Lassez [Jaffar87a] and an efficient algorithm for handling linear constraints within the framework of the resolution strategy was developed [Jaffar86a]. This generalization extends the unification strategy by including arithemetic constraints in a substitution. That is, a substitution can take the form $\theta = \{x = 2, y = f(2), x + z = 2\}$ where both unifiable terms and satisfiable constraints are included.

3 Structural Modeling: A Conceptual Description

Structures are modeled using nodes and components, the physical behavior of components is represented using constraints, and the behavior of the structure as a whole is defined using the behavior of its constituent components. The process of structural analysis is modeled as a state transformation process. The nature of nodes, states and state transformations, structural components, and structures is discussed below and a conceptual representation for modeling structures is presented. The conceptual representation presented in this section is then formalized in the next section.

Nodes: Nodes are spatial entities with three attributes: a position vector, a displacement vector, and a set of force vectors (force history). The position vector defines the spatial position of the node. The displacement vector defines the displacement of the node due to the applied loads. The force history defines the sum of the forces applied to the node. For example, the force history of the node common to three structural components is $\{F_0, F_1, F_2, F_3\}$. where F_0 is the initial force applied to the node, F_1 is the sum of F_0 and the force applied to the node due to the first structural component, F_2 is the sum of F_1 and the force applied to the node due to the second structural component, and F_3 is the sum of F_2 and the force applied to the node due to the third structural component. Let N, N_1, N_2, $\cdots$ denote nodes and p, d, and h be functions that return the position vector, the displacement vector, and the force history of a node, respectively.

States and State Transformations: A state is a set of distinct nodes (two nodes are distinct if their spatial location is different). One state can be transformed into a new state by adding one or more force vectors to the force history of one or more nodes. For example, consider the state (S_i) shown in Figure 1(a) where the force history associated with each node is shown diagramatically by arrows. Figure 1(b) shows a new state (S_{i+1}) which is a transformation of S_i. Note that S_{i+1} is reached by adding a force vector to two of the nodes which occurs when a structural component is added to the structure (see below). The addition of a structural component to the structure results in incorporating the end forces of the component in the force history of the nodes that are associated with the component.

Initial and Final States: The force equilibrium condition at the joints of the structure needs to be maintained in two states: the initial state and the final state. In the initial state, when the structure is composed of no components, the sum of the forces at the structural joints is zero. This corresponds to defining the state nodes such that the force history of every node contains exactly one force vector with a zero magnitude. In the final state, when all the structural components have been defined, the sum of the forces at the structural joints must also be zero. This corresponds to assigning zero to the last force vector of the force history of each node in the state.

Structural Components: A structural component is defined using a set of variables. The behavior of the component is defined using a set of relations (constraints) between its variables. Let N denote the set of nodes associated with a component, let C be the set of constraints that describe the behavior of the component, and let V be the set of variables in C. A component is denoted by the ordered set $< N, V >$ and the set of constraints describing the behavior of the component is denoted by $\lambda V.C$. That is, C is the set of

expressions whose variables are the elements of the set V.

In general, the behavior of most types of structural components, from 2D truss elements to 3D solid elements, can be modeled using their nodes and their equilibrium equations (constraints) which can be written as $Kd = f$ where K is the stiffness matrix, d is displacement vector, and f is the force vector of the element.

Structures: A set of interconnected structural components forms a structure. The behavior of a structure is defined using the behavior of its constituent components. Since the behavior of each component is modeled using constraints, the behavior of the whole structure can be defined as the set of its component constraints. A structure is *well-behaved* if its component constraints are satisfied and the displacement compatibility and the force equilibrium conditions are maintained at its nodes.

The process of defining a structure and its behavior can be modeled using states and state transformations. The unloaded nodes of the structure constitute the initial state. To move from one state to the next, a structural component whose constraints are satisfied is added to the structure. As a result, the force history of the nodes associated with the component is modified; a state transformation has occurred. Once all the structural components are added to the structure, and the last force vector of the force history of every node is set to zero in order to maintain force equilibrium at the nodes, the final state is reached. For example, given the initial state S_0 and three structural components, the state transformation process can be written as

$$initialstate(S_0) \wedge transform(S_0, X_1, S_1) \wedge transform(S_1, X_2, S_2) \wedge$$
$$transform(S_2, X_3, S_3) \wedge finalstate(S_3)$$

That is, S_0 is the initial state, the structural component X_1 transforms S_0 to a new state (S_1), the structural component X_2 transforms S_1 to a new state S_2, the structural component X_3 transforms S_2 to a new state S_3, and S_3 is the final state. Note that a component can cause a state transformation if its constraints are satisfied. Thus, when the final state is reached all the component constraints are satisfied and a solution to the problem is obtained.

4 The Formal Theory

As it is being presented in this paper, the formal theory for describing the behavior of structural systems is composed of a formal language and a set of axioms. The formal language of the theory has a vocabulary and a set of rules for constructing the valid expressions of the language. These rules were given in Section 2.1. The vocabulary of the language and the axioms of the theory and their intuitive interpretation are given below. The formal theory is represented as Δ_s.

Vocabulary

Constants: Lowercase letters denote constants.

Variables: Uppercase letters denote variables.

Functions:

$last(N)$:	Returns the last element of the set X.
$p(N)$:	Returns the position vector of the node N.
$d(N)$:	Returns the displacement vector of the node N.
$h(N)$:	Returns the force history of the node N.

Predicates:

$structure(\{X_1, X_2, \cdots, X_n\})$:	$X_1, X_2, \cdots, X_n$ are components of a structure.
$initialstate(S)$:	S is an initial state.
$transform(S_{i-1}, X_i, S_i)$:	X_i transforms S_{i-1} to S_i.
$object(X_i)$:	X_i is a structural component.
$nodes(\{N_1, N_2, \cdots, N_n\})$:	$N_1, N_2, \cdots, N_n$ are a set of nodes.
$position(P)$:	P is a candidate spatial location for a node.
$finalstate(S)$:	S is a final state.
$next(S_{i-1}, X_i, S_i)$:	S_i is the next state to S_{i-1}.
$constrain(V)$:	The constraints associated with the set of variables V are satisfiable.

Quantifiers: $\{\forall, \exists\}$

Logical Operators: $\{\neg, \vee, \wedge, \Rightarrow\}$

Axioms

(1) $initialstate(S_0) \wedge transform(S_0, X_1, S_1) \wedge transform(S_1, X_2, S_2) \wedge \cdots \wedge transform(S_{n-1}, X_n, S_n) \wedge finalstate(S_n) \Rightarrow structure(\{X_1, X_2, \cdots, X_n\})$

If S_0 is the initial state and X_1 transforms S_0 to S_1 and X_2 transforms S_1 to S_2 and ... and X_n transforms S_{n-1} to S_n and S_n is the final state then the structure which is composed of X_1, X_2, $\cdots$, X_n is well-behaved.

(2) $object(X_i) \wedge next(S_{i-1}, X_i, S_i) \Rightarrow transform(S_{i-1}, X_i, S_i)$

If X_i is an object and X_i generates S_i from S_{i-1} then X_i transforms S_{i-1} to S_i.

(3) $nodes(N) \wedge constrain(V) \Rightarrow object(< N, V >)$

If N is a set of nodes and V is a set of variables and the constraints associated with V are satisfiable then the set $< N, V >$ defines an object.

(4) $\lambda V.C \Rightarrow constrain(V)$

If the arithmetic expressions $\lambda V.C$ are satisfiable then the constraints associated with the set of variables V are satisfiable.

(5) $position(p(N_1)) \wedge position(p(N_2)) \wedge \cdots \wedge position(p(N_n)) \Rightarrow nodes(\{N_1, N_2, \cdots, N_n\})$

If $P(N_1), P(N_2), \cdots, P(N_n)$ are position vectors then the node set $\{N_1, N_2, \cdots, N_n\}$ exists.

(6) $h(N_1) = [0] \wedge h(N_2) = [0] \wedge \cdots \wedge h(N_n) = [0] \Rightarrow initialstate(\{N_1, N_2, \cdots, N_n\})$

If the force history of each node N_1,N_2,$\cdots$, and N_n contains only one element with a magnitude of zero then the set of nodes $\{N_1, N_2, \cdots, N_n\}$ constitutes the initial state.

(7) $last(h(N_1)) = 0 \wedge last(h(N_2)) = 0 \wedge \cdots \wedge last(h(N_n)) = 0 \Rightarrow$
$finalstate(\{N_1, N_2, \cdots, N_n\})$

If the last element of the force history of each node N_1,N_2,$\cdots$, and N_n has a magnitude of zero then the set of nodes $\{N_1, N_2, \cdots, N_n\}$ constitutes the final state.

(8) $position(a)$

(9) $position(b)$

$\vdots$

$a, b, \cdots$ are the permissible spatial positions for the nodes.

A more detailed discussion on the formulation of the theory for 2D truss structures and an implementation of the theory for 2D truss structures are presented in [Lak90] and [Lak89], respectively.

5 Reasoning about the Theory

The conventional approach for automating structural analysis involves developing a model that has predefined input and output parameters. For example, the automation of the displacement formulation of the finite element method for structural analysis involves developing a model that, given a structure, takes the nodal forces (f) and the element properties (K) as its input and gives the nodal displacements (d) and member forces as its output. In this model, it is further assumed that the type (truss, beam, etc.) of all the elements in the structure is known.

The formal theory Δ_s presented in Section 4 can also be viewed as a model for describing the behavior of structural systems. This model, however, makes no prior assumptions about its input and output parameters. The model represents a relationship between the parameters in such a way that, in principle, any parameter can be considered either as an input or as an output. Member and nodal forces, member and nodal displacements, material and section properties and type of the structural elements constitute the input and output parameters in Δ_s. Δ_s also makes no prior assumptions about the type of the structural elements. While in the conventional model one is forced to specify the type of each structural element, in Δ_s the selection of the element type can be left to the model. In this sense, Δ_s can be viewed as a synthesis mechanism capable of generating a portion of, or even the whole structure.

Reasoning about Δ_s lends itself to formulating a theorem and proving its truth value using the resolution strategy. Of particular interest are those theorems that model structural systems. These theorems, when proved true, render the output parameters associated with the modeled structure. These theorems are expressed using the predicate *structure*. Given a set of structural component denoted by X_1, X_2, $\cdots$, X_n the theorem is simply written as $structure(\{X_1, X_2, \cdots, X_n\})$. The theorem states: "is the structure composed

of X_1, X_2, $\cdots$, X_n well-behaved?" The truth value of this theorem can be determined using the resolution strategy. If the theorem proves to be true, then the unknown forces and displacements associated with the components and the type of components (if not specified) are determined.

Using the following three examples the utility of the formal theory presented herein for the analysis, synthesis, and symbolic calculation of the relationship between two parameters of interest is demonstrated. Although the following examples involve 2D truss structures, the theory is applicable to other types of structures as well. The theory can handle both statically determinant and statically indeterminate structural systems.

5.1 Example I

In this example Δ_s is utilized for analyzing a truss structure. Here, the nodal forces and the type, properties, and configuration of the components are input parameters and the nodal displacements and member forces are output parameters. The next example illustrates the case where the type and configuration of some of the components are output parameters. Consider the truss structure shown in Figure 2. In this structure there are seven truss components, two support components, and one load component, each with a unique numerical identifier as shown in the figure. The theorem for analyzing this structure is $\mathit{structure}(\{X_1, X_2, \cdots, X_{10}\})$ where X_i denotes the i^{th} structural component. As a result of proving this theorem the unknown member force and nodal displacement vectors are evaluated. The evaluated force and displacement vectors for components 1 and 8 are given below.

$$X_1 :: \begin{cases} F = (F_i, F_j) & \text{where } F_i = (0.25, 0.5) \quad \text{and } F_j = (-0.25, -0.5) \\ D = (D_i, D_j) & \text{where } D_i = (0, 0) \qquad\quad \text{and } D_j = (0.003, -0.00989) \end{cases}$$

where F_i and D_i are the force and the displacement vectors at one end and F_j and D_j are the force and the displacement vector at the other end of component 1.

$$X_8 :: \begin{cases} F = (-0.25, 0.5) \\ D = (0, 0) \end{cases}$$

where F is the force vector and D is the displacement vector associated with component 8.

5.2 Example II

In this example Δ_s is used to synthesize a truss structure. Here, the nodal forces and the type and properties of two support components are input parameters, and the nodal displacements and the type and configuration of three components are output parameters.

Consider the force and support components shown in Figure 3(a). It is possible to synthesize a well-behaved truss structure that supports the load using Δ_s. For example, the truss structure shown in Figure 3(b) can be generated as a candidate solution using the theory. The structure of Figure 3(b) is just one solution, other valid solutions can also be deduced using Δ_s.

5.3 Example III

In this example Δ_s is used to perform a symbolic computation. Here, the nodal displacements and the cross sectional area of a member are output parameters and everything else is input. Consider the truss structure shown in Figure 4. Δ_s is used to determine the symbolic relationship between the cross-sectional area of the member (denoted by A) and the lateral displacement at the right support (denoted by D). The theorem that renders the required relationship involves defining all the structural components while leaving the cross-sectional area of A as a variable (C). As a result, the resolution strategy determines the displacement (D) of the right support in terms of C. This relationship is

$$-16.63 + 6.84 \times D = 208.33 \times C \times (1.41 - 0.70 \times D)$$

A similar relation for the other structural components can be deduced from the theory.

6 Summary

A structural system is viewed as a collection of truss, load, and support components. The behavior of each component is expressed using a set of constraints and the behavior of the structure, as a whole, is defined in terms of the behavior of its constituent components. A formal theory (Δ_s) that captures the relationship between a structure and its constituent components and describes the behavior of the structure using its component constraints is developed. The theory is formally expressed using predicate logic. Reasoning about the theory, which provides a means to analyze and synthesize structural systems, is accomplished using the resolution theorem proving strategy.

7 Discussion

Through the use of the formal approach one is able to systematically identify, formally represent, analyze, communicate, and effectively utilize engineering knowledge, thus facilitating the development of intelligent systems for engineering problem solving. This approach exhibits the following properties.

- It unifies (integrates) the representation of arithmetic and non-arithmetic constraints. Constraints are represented as expressions of a formal language and are processed using a theorem-proving strategy.
- The constraint representation and processing are considered as two separate tasks. The human problem solver is mainly concerned with the representation of the problem, and constraint processing task is done automatically, with little or no human interaction.
- Engineering knowledge can be declaratively and formally represented. Declarative representation facilitates modification of knowledge. Also, formal representation makes knowledge amenable to mechanical processing.

- The formal representation of knowledge enables the knowledge to be analyzed. The adequacy of knowledge for the intended use can be formally determined.
- Rules for reasoning are formally expressed, and their adequacy for the intended use can be examined.
- It provides a means for the formal communication of engineering knowledge between man and machine.
- It provides a framework for developing constraint-based systems for engineering design.
- It provides a framework for maintaining the integrity of engineering databases.
- It facilitates the parallel processing of engineering problems.

No other approach to automating engineering problem solving provides all the advantageous mentioned above. The role of the formal approach in the constraint-based design, database integrity maintenance, and parallel processing for structural design is discussed below.

7.1 Constraint-Based Design

In general, an engineering design problem can be represented using constraints and can be solved by constraint processing. That is, the problem representation involves identifying and expressing a set of constraints that define the relationships between a set of parameters of interest. A problem solution is obtained by taking a subset of the parameters as input and deriving the output parameters from the constraints and input parameters. This approach to automation is called the constraint-based approach and has been applied to several engineering problems [Sussman80,Serrano86,Ervin87,Chan86].

In the formal approach a problem is modeled using objects (parameters) and relations (constraints). The objects and relations are then formally represented using a language of logic and are mechanically manipulated using theorem proving. The formal model of the problem can then be viewed as a system of constraints, and theorem proving can be used as a means for deducing the output parameters from the constraint system and input parameters. Thus, the formal approach can be used as a basis for developing constraint-based systems for engineering problem solving.

The use of the formal approach for developing constraint-based systems has two main advantages. First, it supports the integration of arithmetic and non-arithmetic constraints by providing a formal representation language and a mechanical processing strategy for both constraint types. Second, the formal approach supports the incremental processing of constraints. In particular, arithmetic constraints can be processed incrementally, and in each increment their satisfiability can be established parametrically. This is advantageous in problems whose solutions have to be obtained by searching through a space of possible solutions. In such problems the constraint system can be partially processed once, and then candidate solutions can be selected and tested using the partially processed constraints. Thus, the constraint processing time is substantially reduced.

7.2 Database Integrity Maintenance

A (relational) database provides a problem-independent data structure for storing, retrieving, and manipulating data. A relational database is composed of a set of relations. Each database relation can be viewed as a two-dimensional table where rows (tuples) represent objects and columns (attributes) represent properties of objects. The use and advantages of databases for managing engineering data have been addressed by several researchers [Eastman82,Fenves85,Rasdorf82,Rasdorf86,Rasdorf87].

One of the research questions in the area of engineering databases is how to maintain the integrity of a design database. The integrity of a database can be examined and maintained by defining, checking, and enforcing constraints between data items in the database.

There is a close relationship between the relational data model and predicate logic. A database relation can be represented by a set of facts (axioms with no right-hand side clause). Thus, the totality of a relational database can be captured by a set of such facts and implemented using a logic programming language such as Prolog and CLP(R). In this sense, a logic program with just facts models a relational database. Furthermore, integrity constraint can be expressed as axioms of logic and implemented as rules in the logic programming language. Such axioms (rules) can be used to monitor and enforce the validity of the database via theorem proving.

The set of axioms developed for the analysis and synthesis of structures can be viewed as the integrity constraints associated with a structural design database. These constraints can be used for monitoring and enforcing the integrity of the database. When the database is populated by one or more application programs, the constraints can be used to monitor the integrity of the database data (i.e., to check the correctness of the data generated by the application programs and to verify the correct transmission of data from the application programs to the database). The constraints can also be used to generate data. When the database contains incomplete knowledge about a structure, then the constraints can be used to generate the unknown information. For example, given a database that contains the geometry, topology, loading and boundary conditions, and material properties associated with the components of a structure, it is possible to generate the forces and displacements associated with the structural components using the constraints.

Thus, the formal theory Δ_s can be used for analyzing and synthesizing as well as for maintaining the integrity of structural design databases. Of course, to fully explore the database application of the formal theory, an appropriate computing environment needs to be developed. We postulate that such a development is mainly an implementation issue and does not pose much theoretical difficulties.

7.3 Parallel Computation

The formal theory for analyzing and synthesizing structures can be processed in parallel. That is, structures can be analyzed and synthesized in parallel. This capability of the model arises from the fact that the behavior of a structure is explicitly represented in terms of the behavior of its constituent components.

A structure can be partitioned in an arbitrary fashion to several substructures. The satisfiability of the constraints associated with each substructure can be established independently (in parallel); then the substructures can be assembled by imposing the displacement continuity and force equilibrium conditions at their common nodes. Consider a truss structure having a set of components (X). Let us partition X into two substructures, namely X_1 and X_2, such that they have only one node (N) in common. Let Σ_1 and Σ_2 represent the constraints associated with X_1 and X_2, respectively. The satisfiability of the constraint sets Σ_1 and Σ_2 can be established independently, thus in parallel. Then, the two substructures can be assembled quite easily by unifying the displacement of N in the two partitions. That is, if D_1 represents the displacement of N in Σ_1 and D_2 represents the displacement of N in Σ_2, then we unify the two displacements by defining the constraint $D_1 = D_2$ and by proving its satisfiability with respect to the union of the constraint sets Σ_1 and Σ_2.

Furthermore, let H_1 and H_2 denote the force history of N in the two partitions, respectively. H_1 is the set of force vectors representing the sum of forces at N in the substructure X_1. Thus, it can be written as $H_1 = \{h_{11}, h_{12}, \cdots, h_{1n}\}$, where h_{11} is the initial force vector at N, and h_{1n} is the sum of the force vectors at N in X_1. And $H_2 = \{h_{21}, h_{22}, \cdots, h_{2m}\}$, where h_{21} is the initial force vector at N, and h_{2m} is the sum of the force vectors at N in X_2. To assemble the two partitions we need to ensure that the force equilibrium is maintained at N. This can be done quite easily by unifying the two force vectors h_{1n} and h_{12}. That is, the initial force at N in X_2 is the same as the final force at N in X_1. Thus, the node force history after assembling the two partitions becomes $H_{21} = \{h_{11}, h_{12}, \cdots, h_{1n-1}, h_{21}, h_{22}, \cdots, h_{2m}\}$. This scheme for assembling two substructures can be generalized to handle substructures with more than one common node.

Using the above scheme a structure can be partitioned into an arbitrary number of substructures, the substructures can be partially analyzed in parallel, and they can be assembled (two at a time) to form the complete structure. We postulate that for large structures the processing time for partially analyzing a substructure is much greater than the processing time for assembling two substructures. Because, the assembly of two substructures requires satisfying a small number of constraints, while the partial analysis of a substructure requires processing many constraints. Thus, the total time for structural analysis can be reduced significantly via parallel processing. For example, the structure shown in Figure 4 can be partitioned into three substructures, as shown in Figure 5. Let us suppose that the processing of a substructure requires P units of CPU time, and the assembly of two substructures requires Q units of CPU time. Then, the sequential processing of the structure requires $M + M + M + N + N$ or $3M + 2N$ units of CPU time. Since $M >> N$, then the total processing time can be approximated to $3M$. On the other hand, the required time for processing the structure in parallel is $M + 2N$ or approximately M. Therefore, the total processing time for the structure is reduced by a factor of 3, if the structure is processed in parallel.

8 Acknowledgment

This work was sponsored by the National Science Foundation under grant MSM-8451465, a Presidential Young Investigator Award. The support of NSF is gratefully acknowledged.

References

[Barwise77] Barwise, J. (1977) *Handbook of Mathematical Logic*, North-Holland Publishing Company.

[Barwise85] Barwise, J., and Feferman, S. (1985) *Persectives in Mathematical Logic, Model-Theoretic Logics*, Springer-Verlag.

[Chan86] Chan, W.T. (1986) *Logic Programming For Managing Constraint-Based Engineering Design*, PhD Thesis, Stanford University, CA.

[Chang73] Chang, C.L., and Lee, R.C. (1973) *Symbolic Logic and Mechanical Theorem Proving*, Academic Press.

[Eastman82] Eastman, C.M., and Lafue, G.M.E. (1982) "Semantic Integrity Transactions in Design Databases," in *File Structures and Databases for CAD*, Encarnacao, J., and Krause, F.L. (Editors), IFIP, Pages 45-54.

[Ervin87] Ervin, S.M., and Gross, M.D. (1987) "RoadLab- A Constraint Based Laboratory for Road Design," *Artificial Intelligence in Engineering*, Volume 2, Number 4, Pages 224-234.

[Fenves85] Fenves, S.J., and Rasdorf, W.J. (1985) "Treatment of Engineering Design Constraints in Relational Databases," *Engineering with Computers*, Volume 1, Number 1, Pages 27-37.

[Frost86] Frost, R. (1986) *Introduction to Knowledge Base Systems*, Macmillan Publishing Company, New York.

[Genesereth88] Genesereth, M.R., and Nilsson, N.J. (1988) *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann.

[Hayes77] Hayes, P.J. (1977) "In Defense of Logic," Proceedings of the Fifth International Joint Conference on AI, Pages 559-565.

[Jaffar86a] Jaffar, J., and Michaylov, S. (1986) "Methodology and Implementation of a CLP System," Proceedings of the Fourth International Conference on Logic Programming, Pages 196-218.

[Jaffar87a] Jaffar, J., and Lassez, J-L. (1987) "From Unification to Constraints," Proceedings of the Sixth Conference on Logic Programming, Pages 1-18.

[Lak89] Lakmazaheri, S., and Rasdorf, W.J. (1989) "Constraint Logic Programming for the Analysis and Partial Synthesis of Truss Structures," *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, Volume 3, Number 3, Pages 157-173.

[Lak90] Lakmazaheri, S., and Rasdorf, W.J. (1990) "The Analysis and Partial Synthesis of Truss Structures via Theorem Proving," *Engineering with Computers*, Volume 6, Number 1, Pages 31-45.

[Law89] Law, K.H., Rasdorf, W.J., Karamouz, M., and Abudayyeh, O.Y. (1989) "The Role of Computing in Civil Engineering Education," Proceedings of the Sixth Conference on Computing in Civil Engineering, Pages 442-450.

[Lloyd87] Lloyd, J.W., (1987) *Foundations of Logic Programming*, Second extended Edition, Springer-Verlag.

[Loveland78] Loveland, D. (1978) *Automated Theorem Proving: A Logical Basis*, North-Holland.

[Moore85] Moore, R.C. (1985) "The Role of Logic in Knowledge Representation and Commonsense Reasoning," in *Readings in Knowledge Representation*, Brachman, R.J., and Levesque, H.J. (Editors), Morgan Kaufmann, Pages 336-341.

[Nillson89] Nillson, N. (1989) "On Logical Foundation of Artificial Intelligence: A Response to the Reviews by S. Smoliar and J. Sowa," *Artificial Intelligence*, Volume 38, Number 1, Pages 132-133.

[Rasdorf82] Rasdorf, W.J. (1982) "Structure and Integrity of a Structural Engineering Design Database," DRC-02-14-82, Design Research Center, Carnegie Mellon University, Pittsburg, PA.

[Rasdorf86] Rasdorf, W.J., and Wang, T.E., "CDIS: An Engineering Constraint Definition and Integrity Enforcement System for Relational Databases," Proceedings of the 1986 ASME International Computers in Engineering Conference, Volume 2, Pages 273-280.

[Rasdorf87] Rasdorf, W.J., Ulberg, K.J., and Baugh, J.W. (1987) "A Structure-Based Model of Semantic Integrity for Relational Databases," *Engineering with Computers*, Volume 2, Number 1, 1987, Pages 31-39.

[Serrano86] Serrano, D., and Gossard, D.C. (1986) "Combining Mathematical Models with Geometric Models in CAE Systems," Proceedings of the 1986 ASME International Computers in Engineering Conference, Pages 277-284.

[Sussman80] Sussman, G.J., and Steele, G.L. (1980) "Constraints - A Language for Expressing Almost-Hierarchical Descriptions," *Artificial Intelligence*, Volume 14, Number 1, Pages 1-39.

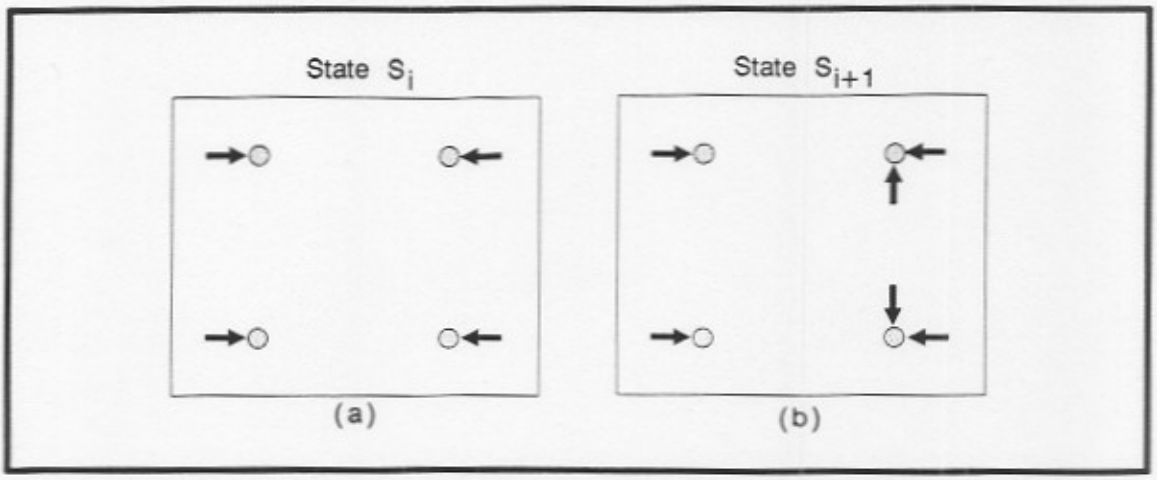

Figure 1: A State Transformation.

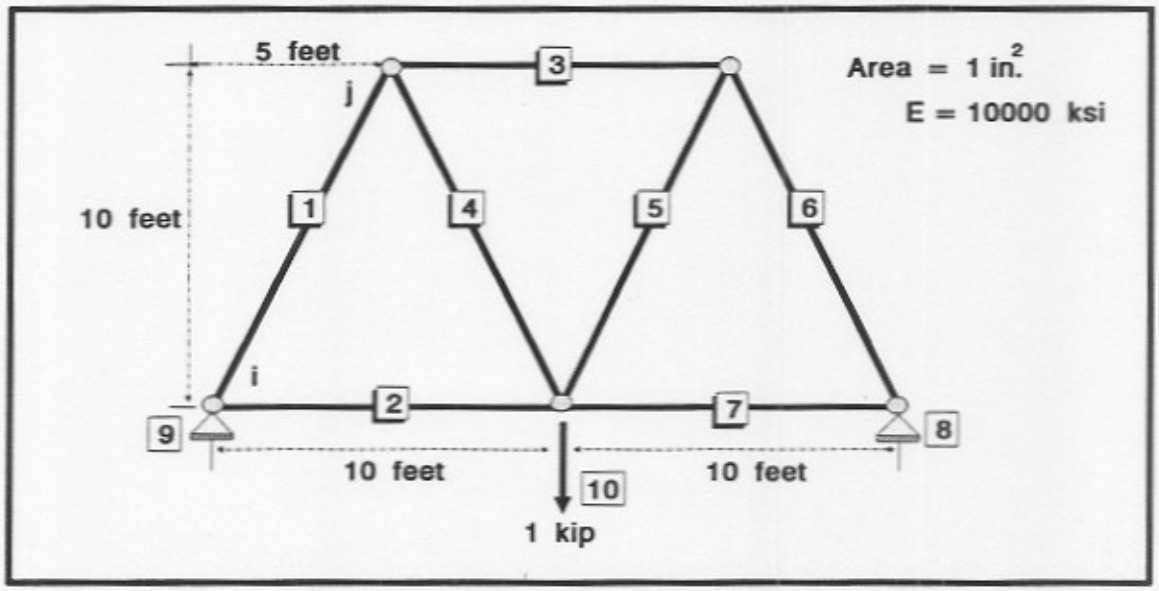

Figure 2: The Truss Structure for Example I.

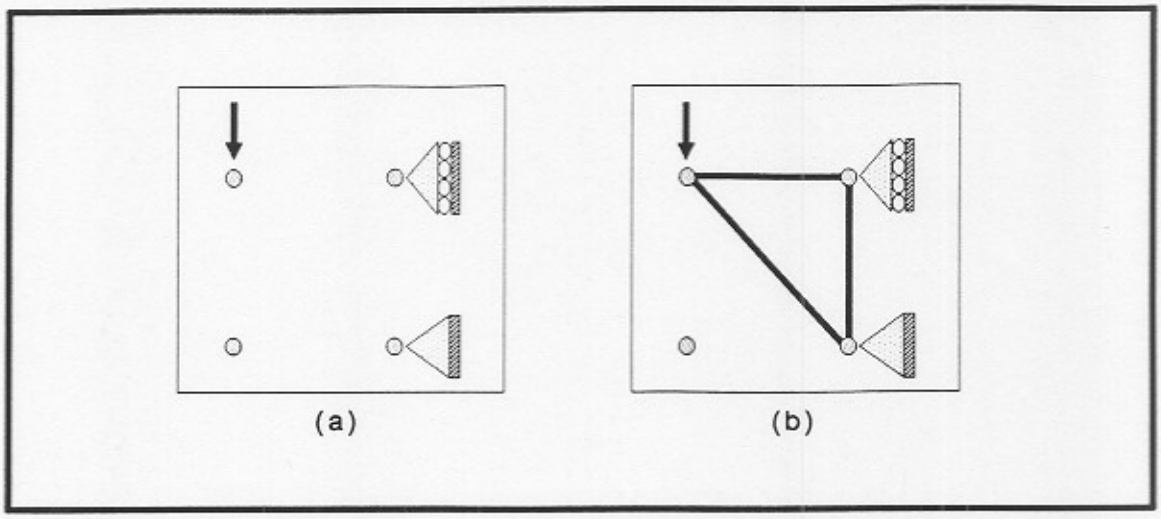

Figure 3: A Synthesized Structure for Example II.

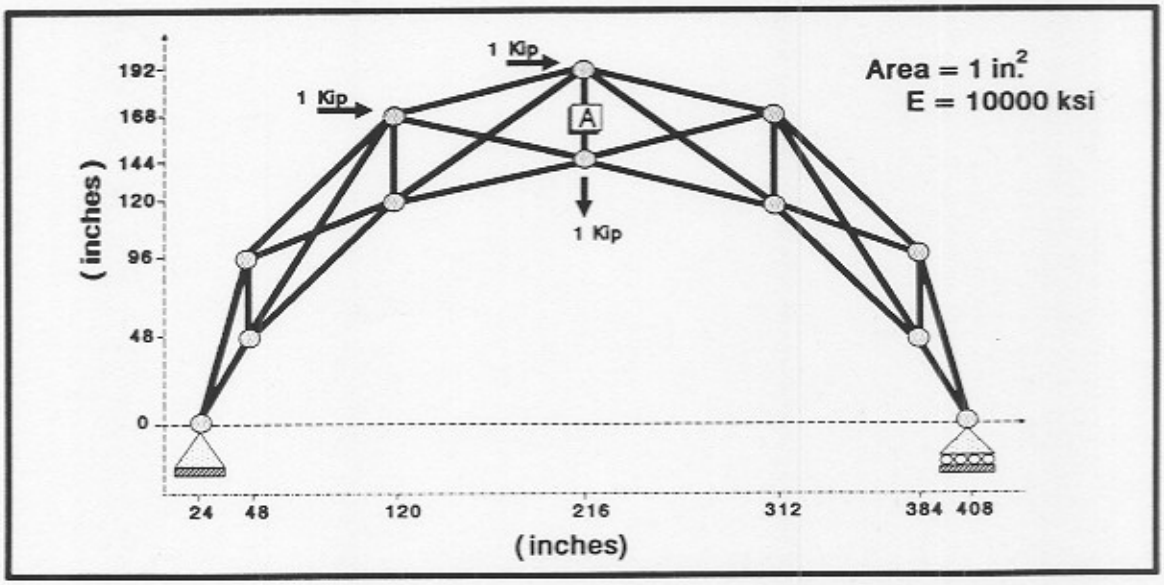

Figure 4: The Truss Structure for Example III.

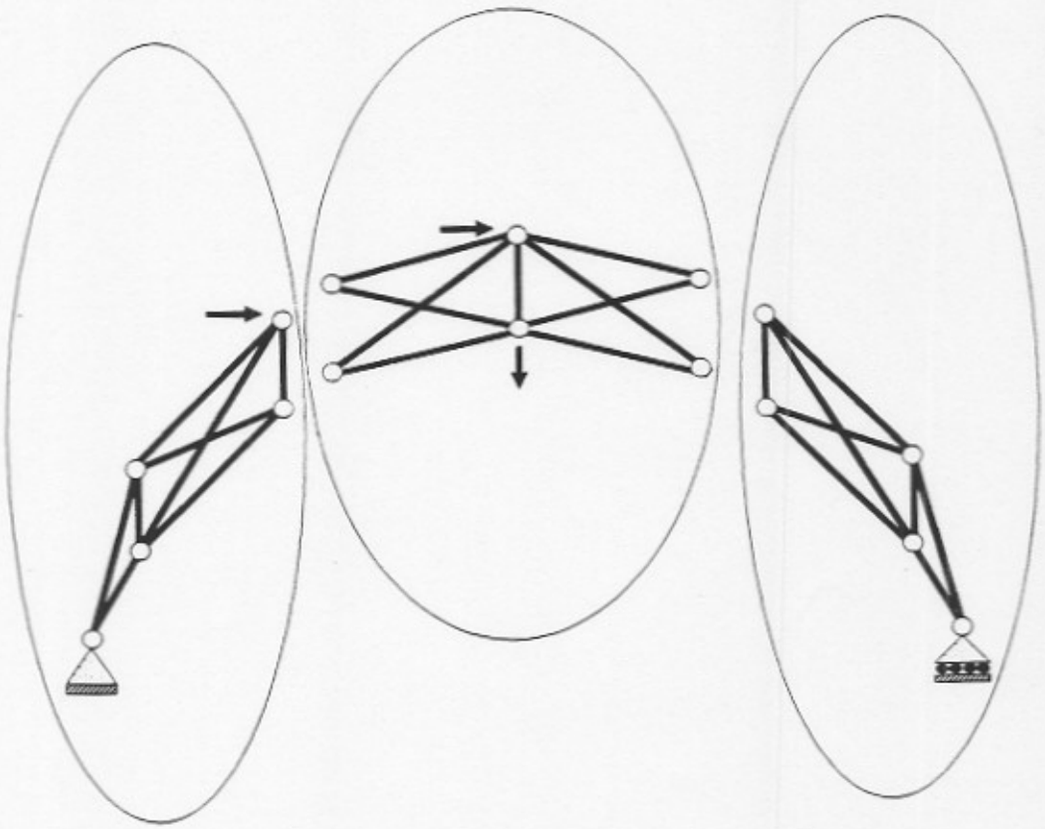

Figure 5: A Partitioned Truss Structure