**THE ANALYSIS AND PARTIAL SYNTHESIS OF
TRUSS STRUCTURES VIA THEOREM PROVING**

Revised

June 1989

Sivand Lakmazaheri, Research Associate
Department of Civil Engineering
North Carolina State University, Raleigh, NC 27695

William J. Rasdorf, Associate Professor
Departments of Civil Engineering & Computer Science
North Carolina State University, Raleigh, NC 27695

Contents

1	Introduction	1
2	Formalization	3
2.1	An Incremental Approach to Truss Analysis	3
2.1.1	Nodes	4
2.1.2	Components	5
2.1.3	Structural Modeling	7
2.2	Predicate Logic	9
2.3	Axiomatizing the Behavior of Truss Structures	9
2.3.1	The Object Sets	10
2.3.2	Functions	11
2.3.3	Predicates	12
2.3.4	Axioms	13
2.4	Truss Analysis and Partial Synthesis Via Theorem Proving	15
2.4.1	Example I (Structural Analysis)	16
2.4.2	Example II (Symbolic Computation)	17
2.4.3	Example III (Partial Synthesis)	18
3	Implementation	19
3.1	Constraint Logic Programming	20
4	Summary	20
5	Implications for Structural Design Automation	21
6	Further Work	22
7	Acknowledgment	23

The Analysis and Partial Synthesis of Truss Structures via Theorem Proving

Sivand Lakmazaheri*

William J. Rasdorf†

Abstract

A use of theorem proving for the analysis and partial synthesis of truss structures is presented. The behavior of a truss structure is modeled as the set union of the behavior of its constituent components where the behavior of each component is modeled by a set of constraints. This component/constraint model is formally represented by a set of axioms using predicate logic. The axioms are then used to **analyze** and partially **synthesize** truss structures via theorem proving.

Constraint Logic Programming is identified as a suitable implementation vehicle for the analysis and partial synthesis of truss structures via theorem proving. Several important implications of the formulation for structural design automation, for database integrity, and for parallel processing are discussed.

1 Introduction

In the quest for automating the engineering design process one seeks to represent and mechanically process design knowledge. In general, the design automation process involves (1) identifying the relevant design knowledge, (2) providing a formalism (or a set of formalisms) for representing and processing the knowledge, and (3) implementing the formalism(s) in a computing environment. The first two issues are concerned with formalization of basic design principles and the last issue is concerned with the implementation of a computer program based on the principles. The *formalization* and *implementation* are two orthogonal issues and need to be addressed separately [1]. The former is discussed herein.

In recent years a growing interest in developing scientific design theory and methodology has emerged [2] [3]. In a report to the National Science Foundation [4], leading design experts suggested that an understanding of the design process through a coherent body of fundamental design theory and methodology is needed. In the same report, it is suggested that formulation of basic principles, systematic methods, and rational concept evaluation techniques might eventually lead to the development of a design theory.

*Research Associate, Department of Civil Engineering, North Carolina State University, Raleigh, NC 27695

†Associate Professor, Departments of Civil Engineering and Computer Science, North Carolina State University, Raleigh, NC 27695

Systematic and scientific progress in many disciplines require the invention and use of appropriate mathematical apparatus with which concepts can be expressed and unified [21]. 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 [5]. As a branch of mathematics, logic is concerned with the language for defining mathematical objects (structures) and the laws for reasoning about them [6].

Mathematical logic is **foundational** to the knowledge formalization and reasoning automation in the field of Artificial Intelligence [7] [8] [9]. An intelligent agent capable of reasoning in a particular domain must have adequate knowledge about objects and their relations in the domain of interest, 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. A system that possesses these characteristics is called a *formal system* [10].

In the interdisciplinary field of engineering design, formal systems have played a central role in representing and automating the processing of design knowledge. Production systems have been extensively used as a class of formal systems to develop knowledge-based expert systems for engineering applications. Two such systems in the domain of structural design are HI-RISE [11] and ALL-RISE [12]. Shape grammars are another class of formal systems developed for reasoning about geometric objects [13]. A use of shape grammars in the design of retaining walls is reported by Fenves and Baker [14].

A widely used formal language for knowledge representation is predicate logic. Kim and Suh [15] discussed the use of predicate logic for stating design axioms. Davis [16] used predicate logic to provide a general framework for qualitative reasoning about solid objects. Hayes [17] has formulated the physical behavior of liquids using predicate logic.

Predicate logic provides a formal means for externalizing and representing design knowledge as axioms. Reasoning about design knowledge can then be automated via theorem proving. Furthermore, *logic programming* is a declarative programming methodology based on predicate logic and it provides a suitable implementation vehicle for automated reasoning in design.

Predicate logic can be used to represent truss structures and their physical behavior, and theorem proving can be used to reason about the physical behavior of truss structures. By physical behavior of truss structures we mean the response (member displacements and member forces) of the structure to its environment (applied loads and boundary conditions). By reasoning about the behavior of truss structures we mean that given an axiomatic definition of the behavior of truss structures it is possible to use a deductive inference strategy to prove some relevant theorem based on the axioms. We define the theorems of interest as those that result in determining

- the response of a truss structure to its environment (structural analysis),
- the relationships between the attributes of structural components (symbolic compu-

tation), and

- the complete truss structure from a partially defined structure (structural synthesis).

In this paper we formalize and represent the physical behavior of truss structures using a set of axioms, and we demonstrate the use of theorem proving for structural analysis and synthesis of partially defined structures (partial synthesis) using the axioms.

2 Formalization

Formalization of the behavior of truss structures is accomplished by viewing truss structures as a collection of components, by describing the behavior of each component using constraints, and by modeling the behavior of the whole structure from the behavior of its constituent components using a set of axioms.

The axioms that model the behavior of truss structures need to be defined such that both the analysis and partial synthesis of truss structures can be accommodated. This is accomplished by formulating a set of axioms that support incremental definition of truss structures. That is, the components that form the structure are defined in increments, and in each increment the behavior of the partial structure is analyzed.

In this section a constraint-based incremental approach for truss analysis and partial synthesis, a brief overview of predicate logic and its constructs, and an axiomatization of the behavior of truss structures using predicate logic are discussed.

2.1 An Incremental Approach to Truss Analysis

The displacement formulation for the analysis of truss structures involves (1) idealizing truss structures as a set of nodes and a set of truss elements such that the elements are interconnected at the nodes, (2) defining the equilibrium equations for the truss elements, (3) assembling these equations into a system of equations called the structure's equilibrium equation, and (4) solving the structure's equilibrium equation for the unknown (nodal displacements). Note that in the displacement formulation the structure's equilibrium equation models the behavior of the whole structure and the behavior of the structural components are not explicitly represented in the model. This makes it difficult, if not impossible, to automate reasoning about the behavior of truss structures and their components.

We postulate that automated reasoning about the physical behavior of structural systems in general, and truss structures in particular, is facilitated if

- the behavior of structural components is represented explicitly, and
- the behavior of structural systems is defined as the set union of the behavior of their constituent components.

In this section we describe an incremental approach to the analysis of truss structures which satisfies these requirements.

In the incremental approach, truss structures are defined using *nodes* and *components*. The nodes are used to model connectivities between the structural components. Each component has a physical behavior which is modeled using a set of constraints. The behavior of a truss structure is then modeled as the set union of the behavior of its components. The nodes, the components, and the modeling process of truss structures are discussed below.

2.1.1 Nodes

Nodes are spatial entities that describe the geometry and spatial position of structural components. A node is defined by a position vector ($\vec{P}_n$), a displacement vector ($\vec{D}_n$), and an ordered set of force vectors (Ω_n) called the node *force equilibrium history* or just *node history*. The position vector defines the node location in a two dimensional space, the displacement vector defines the node displacement, and the node's force equilibrium history models the node's force equilibrium as structural components are connected to the node.

As new structural components are connected to a partially defined truss structure the force equilibrium equations for the nodes of the structure change. For example, consider a node of a partially defined structure which connects two truss components. The force equilibrium equation for this node is

$$\vec{F}_1 + \vec{F}_2 = 0$$

where $\vec{F}_1$ and $\vec{F}_2$ are the truss components force vectors. When a new component is added to the node the force equilibrium equation for the node becomes

$$\vec{F}_1 + \vec{F}_2 + \vec{F}_3 = 0$$

where $\vec{F}_3$ is the force vector of the added truss component. To be able to define the node equilibrium equations incrementally, as structural components are added to a partially synthesized structure the node's force equilibrium histories are defined. Elements of Ω_n are force vectors and they are defined recursively. The $(i)^{th}$ element of the set is the sum of the $(i-1)^{th}$ element and the force vector of the i^{th} component. That is,

$$\vec{T}_{n_i} = \vec{F}_i + \vec{T}_{n_{i-1}} \quad \text{for } 1 \leq i < m$$

where $\vec{T}_{n_i}$ and $\vec{T}_{n_{i-1}}$ are the $(i)^{th}$ and $(i-1)^{th}$ elements of Ω_n , $\vec{F}_i$ is the component's force vector, and $(m+1)$ is the number of elements in the set. Intuitively, $\vec{T}_{n_i}$ and $\vec{T}_{n_{i-1}}$ can be viewed as sum of the forces at the node before and after connecting a component to the node.

Consider the node n shown in Figure 1a. Initially no component is connected to this node. We refer to this situation as the initial state (S_0). In the initial state the node history contains one element which is denoted by $\vec{T}_{n_0}$ and it represents the initial force vector at the node n . Now, suppose a component is connected to the node in the next state (S_1) as shown in Figure 1b. The node equilibrium equation becomes

$$\vec{T}_{n_1} = \vec{F}_1 + \vec{T}_{n_0}$$

where $\vec{F}_1$ is the component's force vector and $\vec{T}_{n_1}$ is the total node force in S_1 . Thus, the node history for n is

$$\Omega_n = \langle \vec{T}_{n_1}, \vec{T}_{n_0} \rangle$$

In the next state (S_2) another component is connected to n as shown in Figure 1c. The node equilibrium equation in S_2 is written as

$$\vec{T}_{n_2} = \vec{F}_2 + \vec{T}_{n_1}$$

where $\vec{F}_2$ is the new component's force vector. And $\vec{T}_{n_2}$ is the total node force in S_2 . And the node history becomes

$$\Omega_n = \langle \vec{T}_{n_2}, \vec{T}_{n_1}, \vec{T}_{n_0} \rangle$$

An interesting property of this formulation is that the node equilibrium equation in each state can be viewed as a behavioral characteristic of the component that is connected to the node at that state rather than as a characteristic of the node itself. In this way the need for explicit modeling of the behavior of the node is eliminated and the behavior of the structure can be modeled based on the behavior of its components only.

2.1.2 Components

Three types of components are defined for modeling truss structures. these are *truss components*, *load components*, *support components*. A truss component is a one dimensional geometrical entities (object) whose geometry is defined using two nodes. These nodes are placed at the ends of the component. Load and support components are non-dimensional spatial entities whose spatial position is identified by a node.

Truss Components: A truss component is a one dimensional element whose length and inclination angle are defined using two nodes. The behavior of each truss component is modeled by

- A system of equations that describe the truss component's equilibrium equation,

$$\begin{pmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{pmatrix} \begin{Bmatrix} \vec{D}_1 \\ \vec{D}_2 \end{Bmatrix} + \begin{Bmatrix} \vec{F}_1 \\ \vec{F}_2 \end{Bmatrix} = \begin{Bmatrix} \vec{0} \\ \vec{0} \end{Bmatrix}$$

where K_{ij} , $i, j = 1, 2$ represent the submatrices of the 2D truss element stiffness matrix. $\vec{D}_1, \vec{D}_2$ are the component displacement vectors, and $\vec{F}_1, \vec{F}_2$ are the force vectors at the two end of the truss element.

- A system of equations that ensures the displacement compatibility at the two end nodes of the component.

$$\begin{Bmatrix} \vec{D}_1 \\ \vec{D}_2 \end{Bmatrix} = \begin{Bmatrix} \vec{D}_a \\ \vec{D}_b \end{Bmatrix}$$

where $\vec{D}_a$ and $\vec{D}_b$ are the displacement vectors of the two end nodes associated with the component. These nodes are denoted by a and b .

- A system of equations that ensures the force equilibrium at the two end nodes of the component.

$$\begin{Bmatrix} \vec{T}_{a_i} \\ \vec{T}_{b_i} \end{Bmatrix} = \begin{Bmatrix} \vec{F}_1 \\ \vec{F}_2 \end{Bmatrix} + \begin{Bmatrix} \vec{T}_{a_{i-1}} \\ \vec{T}_{b_{i-1}} \end{Bmatrix}$$

where $\vec{T}_{a_i}$ and $\vec{T}_{a_{i-1}}$ are two successive elements of the force equilibrium history for the node a . And $\vec{T}_{b_i}$ and $\vec{T}_{b_{i-1}}$ are two successive elements of the force equilibrium history for the node b .

The above set of equations is called the *truss component's behavior model* and is denoted by $\mathcal{C}_t$.

Load Components: Load components are defined using three vectors, a force vector, a displacement vector, and a position vector. A load component is spatially identified by a node and its behavior is described using the following equations.

- A system of equations that ensures the displacement compatibility at the node associated with the component.

$$\vec{D}_l = \vec{D}_a$$

where $\vec{D}_l$ is the load component displacement vector and $\vec{D}_a$ is the node displacement vector. This equation is called the *load component's behavior model* and it is denoted by $\mathcal{C}_l$.

- A system of equations that ensures the force equilibrium at the node associated with the component.

$$\vec{T}_{a_i} = \vec{F}_l + \vec{T}_{a_{i-1}}$$

where $\vec{T}_{a_i}$ and $\vec{T}_{a_{i-1}}$ are two elements of the node history, and $\vec{F}_l$ is the force vector.

Support Components: Support components are defined using three vectors, a force vector, a displacement vector, and a position vector. A support component is spatially identified by a node and its behavior is described using the following equations.

- A system of equations that ensures the displacement compatibility at the node associated with the component.

$$\vec{D}_s = \vec{D}_a$$

where $\vec{D}_s$ is the support component displacement vector and $\vec{D}_a$ is the node displacement vector. This equation is called the *support component's behavior model* and it is denoted by $\mathcal{C}_s$.

- A system of equations that ensures the force equilibrium at the node associated with the component.

$$\vec{T}_{a_i} = \vec{F}_s + \vec{T}_{a_{i-1}}$$

where $\vec{T}_{a_i}$ and $\vec{T}_{a_{i-1}}$ are two elements of the node history, and $\vec{F}_s$ is the force vector.

2.1.3 Structural Modeling

The geometry and topology of truss structures are defined using nodes and components. The nodes form a space for constructing the structural components. The space formed by the nodes is called the *structure space*, it is represented by a finite set whose elements are the nodes, and it is denoted by U .

Nodes are either active or inactive. The nodes that are used to defined structural components are called *active nodes*. The set of active nodes is denoted by U_A and it is a subset of U (i.e., $U_A \subseteq U$). The set of *inactive nodes* is denoted by U_I and it is defined as the set difference of U and U_A (i.e., $U_I = U \setminus U_A$).

The behavior of a set of components (a structure) is the set union of the behavior of its constituent components. The set of constraints that models the behavior of the truss structure is called the *structure's behavior model* and it is denoted by C .

The analysis of truss structures lends itself to satisfying the set of constraints C such that the first and last elements of the force history for each node vanish. This is demonstrated by means of an example.

Consider the following structure space

$$U = \{ \langle 0, 0 \rangle, \langle 10, 0 \rangle, \langle 0, 10 \rangle, \langle 10, 10 \rangle \}$$

For readability we identify the elements of U by integer numbers rather than by their coordinates. Thus, U can be written as

$$U = \{1, 2, 3, 4\}$$

where 1 is an identifier for the node $\langle 0, 0 \rangle$, 2 is an identifier for the node $\langle 10, 0 \rangle$, 3 is an identifier for the node $\langle 0, 10 \rangle$, and 4 is an identifier for the node $\langle 10, 10 \rangle$ (see Figure 2a).

In the initial state (S_0) where no components are connected to any nodes the active and inactive node sets are

$$U_A = \{\}$$

$$U_I = \{1, 2, 3, 4\}$$

the node histories are

$$\Omega_i = \langle \vec{T}_{i_0} \rangle \quad \text{for } i = 1, 2, 3, 4$$

and the structure's behavior model (C) is empty.

In state S_1 a load component (L) is connected to node 4 (see Figure 2b). As a result, the active and inactive node sets, the node histories, and the structure's behavior model become

$$U_A = \{4\}$$

$$U_I = \{1, 2, 3\}$$

$$\Omega_i = \langle \vec{T}_{i_0} \rangle \quad \text{for } i = 1, 2, 3$$

$$\Omega_4 = \langle \vec{T}_{4_1}, \vec{T}_{4_0} \rangle$$

$$C = \{C_l\}$$

where C_l represents the load component's behavior model.

In state S_2 a truss component (M_1) is connected to nodes 1 and 3 (see Figure 2c). Hence,

$$\begin{aligned} U_A &= \{1, 3, 4\} \\ U_I &= \{2\} \\ \Omega_i &= \langle \vec{T}_{i1}, \vec{T}_{i0} \rangle \quad \text{for } i = 1, 3, 4 \\ \Omega_2 &= \langle \vec{T}_{20} \rangle \\ C &= \{C_{t1}, C_l\} \end{aligned}$$

where C_{t1} represents the truss component's behavior model.

In state S_3 a truss component (M_2) is connected to nodes 3 and 4 (see Figure 2d). Hence,

$$\begin{aligned} U_A &= \{1, 3, 4\} \\ U_I &= \{2\} \\ \Omega_1 &= \langle \vec{T}_{11}, \vec{T}_{10} \rangle \\ \Omega_2 &= \langle \vec{T}_{20} \rangle \\ \Omega_i &= \langle \vec{T}_{i2}, \vec{T}_{i1}, \vec{T}_{i0} \rangle \quad \text{for } i = 3, 4 \\ C &= \{C_{t2}, C_{t1}, C_l\} \end{aligned}$$

where C_{t2} represents the truss component's behavior model.

In state S_4 a truss component (M_3) is connected to nodes 1 and 4 (see Figure 2e). Hence,

$$\begin{aligned} U_A &= \{1, 3, 4\} \\ U_I &= \{2\} \\ \Omega_i &= \langle \vec{T}_{i2}, \vec{T}_{i1}, \vec{T}_{i0} \rangle \quad \text{for } i = 1, 3 \\ \Omega_2 &= \langle \vec{T}_{20} \rangle \\ \Omega_4 &= \langle \vec{T}_{43}, \vec{T}_{42}, \vec{T}_{41}, \vec{T}_{40} \rangle \\ C &= \{C_{t3}, C_{t2}, C_{t1}, C_l\} \end{aligned}$$

where C_{t3} represents the truss component's behavior model.

In state S_5 a support component (S_1) is connected to nodes 1 (see Figure 2f). Hence,

$$\begin{aligned} U_A &= \{1, 3, 4\} \\ U_I &= \{2\} \\ \Omega_i &= \langle \vec{T}_{i3}, \vec{T}_{i2}, \vec{T}_{i1}, \vec{T}_{i0} \rangle \quad \text{for } i = 1, 4 \\ \Omega_2 &= \langle \vec{T}_{20} \rangle \\ \Omega_3 &= \langle \vec{T}_{32}, \vec{T}_{31}, \vec{T}_{30} \rangle \\ C &= \{C_{s1}, C_{t3}, C_{t2}, C_{t1}, C_l\} \end{aligned}$$

where C_{s_1} represents the support component's behavior model.

And, in state S_6 a support component (S_2) is connected to node 3 (see Figure 2g). Hence,

$$\begin{aligned} U_A &= \{1, 3, 4\} \\ U_I &= \{2\} \\ \Omega_i &= \langle \vec{T}_{i3}, \vec{T}_{i2}, \vec{T}_{i1}, \vec{T}_{i0} \rangle \quad \text{for } i = 1, 3, 4 \\ \Omega_2 &= \langle \vec{T}_{20} \rangle \\ C &= \{C_{s_2}, C_{s_1}, C_{t_3}, C_{t_2}, C_{t_1}, C_t\} \end{aligned}$$

where C_{s_2} represents the support component's behavior model.

Suppose we are content with this truss structure and we would like to determine its behavior. The constraint set C is a model of the behavior of the truss structure subject to the following constraints.

$$\Omega_i = \langle 0, \vec{T}_{i2}, \vec{T}_{i1}, 0 \rangle \quad \text{for } i = 1, 3, 4$$

This constraint arises from the fact that sum of the forces at every node of the structure must vanish at the initial and final states. That is, the nodes are in equilibrium in the initial and final states. Therefore to analyze the structure we need to assign zero to the first and the last elements of the node histories in U_A and satisfy the constraint set C .

The incremental approach provides an abstract representation for truss structures by (1) eliminating the need for assembling the structure's equilibrium equation, (2) viewing structural components as objects, each having its own behavior, and (3) representing the behavior of each object explicitly.

2.2 Predicate Logic

Predicate logic is a formal language for representing objects and their relationships in a domain of interest. The expressions of this language are defined using predicates, functions, and logical operators.

A predicate is a relationship between a set of terms. A predicate with n term (argument) is called an n -place or n -ary predicate. Terms represent abstract objects. In predicate logic a term can be defined using other terms. The mapping between one set of terms and another term is defined using a function. For example, in $z = f^*(a, b)$ the function f^* maps a and b into z where a , b , and z are terms. In the expressions of predicate logic the predicates are separated by logical operators. The following logical operators are used in our formulation: negation ($\neg$), conjunction ($\wedge$), disjunction ($\vee$), and implication ($\leftarrow$). For a complete discussion of predicate logic see [18] or [19].

2.3 Axiomatizing the Behavior of Truss Structures

An *axiom* is an expression in the language of predicate logic for describing a relationship between a set of objects. Axioms are defined using functions and predicates. The terms

associated with the functions and predicates are called objects. The objects are grouped into sets. In this section the set of objects, the functions and predicates, and the axioms for describing the behavior of truss structures are described.

Herein, the following abbreviations and notations are used.

- Constant symbols are shown in lower case letters.
- Variable symbols are shown in upper case letters.
- $[C_1, C_2, \dots, C_n]$ represents a set with n elements.
- $[nil]$ represents an empty set.
- $\langle X_1, X_2, \dots, X_n \rangle$ represents an ordered set with n elements.
- A function or a predicate terms is left unspecified when its value is irrelevant to the discussion.
- $X \cup Y$ represents the set union of X and Y .
- $X \setminus Y$ represents the set difference of X and Y .
- $X = Y$ means objects X and Y are identical.
- $X \neq Y$ means objects X and Y are not identical.
- $X \in S$ means X is a member of set S .
- $X \notin S$ means X is not a member of set S .

2.3.1 The Object Sets

Any object that appears as an argument of a function or a predicate belongs to one of the following sets.

- $\mathcal{R}$: The set of real numbers. Every real number belongs to this set.
- $\mathcal{F}$: The set of two dimensional force vectors. Every two dimensional force vector belongs to this set.
- $\mathcal{D}$: The set of two dimensional displacement vectors. Every two dimensional displacement vector belongs to this set.
- $\mathcal{P}$: The set of two dimensional position vectors. Every two dimensional position vector belongs to this set.
- $\mathcal{N}$: The set of nodes. Every node in the structure space is a member of this set.
- $\mathcal{H}$: The set of node histories. Every node history is a member of this set.

- CV : The set of component-vector sets. All the vector sets associated with the structural components belong to this set. Intuitively a component-vector set models a node (joint) of a structural component.
- CP : The set of component-property sets. The set of material and geometrical properties of structural components belong to this set. This models the material and geometrical properties of truss components.
- TC : The set of truss components. All the truss components belong to this set. Truss components are defined using one component-property set and two component-vector sets.
- LC : The set of load components. All the load components belong to this set. Load components are defined using one component-vector set.
- SC : The set of support components. All the load components belong to this set. Support components are defined using one component-vector set.

2.3.2 Functions

Functions define mappings between a set of objects and an object. Functions are used to define objects in terms of other objects. The following functions are used in our formulation.

- p^* maps two real numbers to a position vector. That is,

$$z = p^*(a, b) \quad \text{where } a, b \in \mathcal{R} \quad \text{and } z \in \mathcal{P}$$

where a and b represent x and y coordinates of the position vector, respectively.

- f^* maps two real numbers to a force vector. That is,

$$z = f^*(a, b) \quad \text{where } a, b \in \mathcal{R} \quad \text{and } z \in \mathcal{F}$$

where a and b represent x and y coordinates of the force vector, respectively.

- d^* maps two real numbers to a displacement vector. That is,

$$z = d^*(a, b) \quad \text{where } a, b \in \mathcal{R} \quad \text{and } z \in \mathcal{D}$$

where a and b represent x and y coordinates of the displacement vector, respectively.

- n^* maps a position vector, a displacement vector, and a node history to a node. That is,

$$z = n^*(a, b, c) \quad \text{where } a \in \mathcal{P}, \quad b \in \mathcal{D}, \quad c \in \mathcal{H} \quad \text{and } z \in \mathcal{N}$$

where a represents the node position, b represents the node displacement, and c represents the node history.

- m^* maps two force vectors, a position vector, and a displacement vector to a component-vector. That is,

$$z = m^*(a, b, c, \langle d, e \rangle) \quad \text{where } a, d, e \in \mathcal{F}, \quad b \in \mathcal{P}, \quad c \in \mathcal{D} \quad \text{and } z \in \mathcal{CV}$$

Intuitively, the component-vector corresponds to the vectors associated with a node (joint) of structural components. Where, a is the joint force vector, b is the joint position vector, c is the joint displacement vector, and d and e are the last two elements of the node history associated with the node.

- g^* maps three real numbers to a component-property. That is,

$$z = g^*(a, b, c) \quad \text{where } a, b, c \in \mathcal{R}, \quad \text{and } z \in \mathcal{CP}$$

where a represents modulus of elasticity, b represents cross-sectional area, and c represents length of a structural component.

- tc^* maps one component-property and two component-vectors to a truss component. That is,

$$z = tc^*(a, b, c) \quad \text{where } a \in \mathcal{CP}, \quad b, c \in \mathcal{TC} \quad \text{and } z \in \mathcal{CP}$$

A truss component is composed of one component-property and two component-vectors objects.

- sc^* maps a component-vector to a support component. That is,

$$z = sc^*(a) \quad \text{where } a \in \mathcal{CV}, \quad \text{and } z \in \mathcal{SC}$$

A support component is composed of one component-vector object.

- lc^* maps a component-vector to a load component. That is,

$$z = lc^*(a) \quad \text{where } a \in \mathcal{CV}, \quad \text{and } z \in \mathcal{LC}$$

A load component is composed of one component-vector object.

2.3.3 Predicates

The following predicates are used for defining the axioms that model the physical behavior of truss structures.

- $structure([X_1, X_2, \dots, X_n])$: The set of objects $[X_1, X_2, \dots, X_n]$ that form a truss structure.
- $behave(X, S_0, S_1)$: X is the transition from the state S_0 to the state S_1 and its related constraints are satisfiable in state S_0 .
- $component(X)$: X is a structural component.

- *state_transition*(X, S_0, S_1): State S_1 is the result of including X into state S_0 .
- *node*(X): X is a node.
- *point*(X): X is a coordinate of a node in the structure space.
- *initial_state*(S): S is the initial state.
- *final_state*(S): S is the final state.
- *satisfied*(X): The set of constraints associated with X are satisfied.

In addition to the above predicates the predicates $=$ and $\neq$ are used in infix notation. That is, $A = B$ is the infix notation for $=(A, B)$ which means A and B are the same. And, $A \neq B$ is the infix notation for $\neq(A, B)$ which means A and B are not identical.

2.3.4 Axioms

The physical behavior of truss structures are modeled using eight axioms.

Axiom 1 *A structure is defined by an initial state, a final state, and several intermediate states where in each intermediate state a well-behaved structural component is defined.*

$$\begin{aligned} \text{structure}([X_1, X_2, \dots, X_n]) \leftarrow \\ \text{initial_state}(S_0) \wedge \text{behave}(X_1, S_0, S_1) \wedge \text{behave}(X_2, S_1, S_2) \wedge \dots \wedge \\ \text{behave}(X_n, S_{n-1}, S_n) \wedge \text{final_state}(S_n) \end{aligned}$$

Axiom 2 *If X is a structural component and the constraints associated with X are satisfied, and a transition from state S_0 to state S_1 occurs as a result of defining X , then X is well-behaved.*

$$\begin{aligned} \text{behave}(X, S_0, S_1) \leftarrow \\ \text{component}(X) \wedge \text{satisfied}(X) \wedge \text{state_transition}(X, S_0, S_1) \end{aligned}$$

Axiom 3 *X is a structural component if it is a truss component, if it is a load component, or if it is a support component.*

$$\begin{aligned} \text{component}(X) \leftarrow \\ (X = \text{tc}^*(m^*(A,), m^*(B,)) \wedge (A \neq B) \wedge \\ (Y = n^*(A,)) \wedge (Z = n^*(B,)) \wedge \text{node}(Y) \wedge \text{node}(Z)) \vee \\ (X = \text{sc}^*(m^*(A,)) \vee X = \text{lc}^*(m^*(A,)) \wedge Y = n^*(A,) \wedge \text{node}(Y)) \end{aligned}$$

X is a truss component if there are two component-vectors that map into X via tc^* , if the position vectors associated with the two component-vectors (i.e., A and B) are distinct, and if there are two nodes Y and Z such that their position vectors are A and B , respectively. X is a support component or it is a load component if there is a component-vector that maps into X , if the position vector associated with the component vector (i.e., A) is also the position vector associated with a node (i.e., Y).

Axiom 4 X is a node if its position vector is defined using the predicate "point".

$$\text{node}(X) \leftarrow (X = n^*(, p^*(Y, Z),) \wedge \text{point}(Y) \wedge \text{point}(Z))$$

Axiom 5 The initial state is defined by the empty set.

$$\text{initial_state}(S) \leftarrow S = [\text{nil}]$$

Axiom 6 The final state is defined as a set of nodes where the first and last elements of their histories are zero.

$$\begin{aligned} \text{final_state}([N_1, N_2, \dots, N_n]) \leftarrow \\ N_1 = n^*(, < 0, \dots, 0 >) \wedge N_1 = n^*(, < 0, \dots, 0 >) \wedge \dots \wedge \\ N_n = n^*(, < 0, \dots, 0 >) \end{aligned}$$

Axiom 7 X is a transition from the state S_0 to the state S_1 . The active node set is used for representing states. In each state either a node is added to the set or the force equilibrium history of an existing node is modified. The node(s) associated with X (the structural component) is added to the set when it is not a member of the set S_0 . The force equilibrium history of the node(s) associated with X is modified when the node(s) is a member of the set S_0 .

$$\begin{aligned} \text{state_transition}(X, S_0, S_1) \leftarrow \\ ((X = tc^*(, m^*(, P_1, D_1, T_1), m^*(, P_2, D_2, T_2))) \wedge \\ (N_1 = n^*(P_1, D_1, H_1)) \wedge (N_1 \in S_0 \vee (N_1 \notin S_0 \wedge H_1 = [\text{nil}])) \wedge \\ (N_2 = n^*(P_2, D_2, H_2)) \wedge (N_2 \in S_0 \vee (N_2 \notin S_0 \wedge H_2 = [\text{nil}])) \wedge \\ (N'_1 = n^*(P_1, D_1, < T_1 \cup H_1 >)) \wedge (N'_2 = n^*(P_2, D_2, < T_2 \cup H_2 >)) \wedge \\ (S_1 = (S_0 \setminus (N_1 \cup N_2)) \cup (N'_1 \cup N'_2)) \vee \\ (((X = sc^*(m^*(, P, D, T))) \vee (X = lc^*(m^*(, P, D, T)))) \wedge \\ (N = n^*(P, D, T)) \wedge (N \in S_0 \vee (N \notin S_0 \wedge H = [\text{nil}])) \wedge \\ (N' = n^*(P, D, < T \cup H >)) \wedge (S_1 = (S_0 \setminus N) \cup N')) \end{aligned}$$

If X is a truss component, and its corresponding nodes N_1 and N_2 belong to the state S_0 , or if they don't belong to the state S_0 and their histories (i.e., H_1 and H_2) are empty, then define two new nodes N'_1 and N'_2 similar to N_1 and N_2 such that the history of N'_1 is the union of H_1 and T_1 , and the history of N'_2 is the union of H_2 and T_2 where T_1 and T_2 represent the force equilibrium vectors associated with X , then form S_1 by replacing N_1 and N_2 with N'_1 and N'_2 in S_0 , respectively. Or, if X is a support or a load component with the node N and if N either belongs to S_0 or its history (i.e., H) is empty, then define N' similar to N such that its history is the union of H and T where T is the force equilibrium vector associated with X , then form S_1 by replacing N with N' in S_0 .

Axiom 8 The constraints associated with X are satisfied.

$$\begin{aligned} \text{satisfied}(X) \leftarrow \\ ((X = tc^*(, , ,)) \wedge C = C_t) \vee (X = sc^*(,) \wedge C = C_s) \vee \\ (X = lc^*(,) \wedge C = C_l) \wedge C \end{aligned}$$

2.4 Truss Analysis and Partial Synthesis Via Theorem Proving

In this section we illustrate the use of theorem proving for the analysis and partial synthesis of truss structures. Theorem proving is a special kind of deductive reasoning. In theorem proving one attempts to formally deduce the truth or falsity of an expression called a *theorem* based on a given set of expressions called *axioms*. *Resolution* is a well-known theorem proving strategy. In resolution a theorem is proved in a sequence of steps called deduction steps. In each deduction step two expressions are resolved into one by a method called *unification*. The theorem is proved true if no expression remains in the last deduction step. However, the truth of the theorem cannot be proved if one or more expressions remain unresolved. In unification two expressions are made syntactically identical by finding an appropriate substitution for their variable terms. For example, consider the two expressions $p(X, f(Y))$ and $p(g(Y), f(a))$ where X and Y are variable symbols. These two expressions can be resolved by the substitution $\{Y/a, X/g(a)\}$. This is to say that for $Y = a$ and $X = g(a)$ the two expressions become syntactically identical. For detailed discussion of unification and theorem proving see [20] [21].

Unification can be considered as a special case of constraint satisfaction [22]. Theorem proving with unification can then be generalized to theorem proving with constraint satisfaction [22]. Theorem proving with constraint satisfaction provides a suitable strategy for reasoning about constraint-based problems. In theorem proving with constraint satisfaction a theorem is proved if it is a logical consequence of the axioms and if the constraints generated in the deduction steps are satisfiable. For example, consider the two axioms

$$\begin{aligned}q(X, Y) &\leftarrow (X + Y > 5) \\s(X, Y) &\leftarrow (X * Y = 10) \wedge q(X, Y)\end{aligned}$$

and the theorem $s(5, A)$. To prove that $s(5, A)$ is true is to prove that $q(X, Y)$ is true and the constraint set $\{X = 5, Y = A, X * Y = 10\}$ is satisfiable. To prove the truth of $q(X, Y)$ it is sufficient to prove that the constraint $\{X + Y > 5\}$ is satisfiable. Thus, to prove $s(5, A)$ is to prove the satisfiability of the constraint set

$$\{X = 5, Y = A, X * Y = 10, X + Y > 5\}$$

This set can be proven to be satisfiable with proper substitutions and arithmetic operations. When a set of constraints has a unique solution, then that solution can be obtained as a byproduct of the process of proving the constraints' satisfiability. For example, the unique solution to the above constraint set (i.e., $X = 5, Y = A = 2$) can be determined by the prover. In general, the satisfiability of a linear constraint set can be proved incrementally. That is, to prove that a set of constraint is satisfiable in the i^{th} deduction step it is sufficient to show that the constraints generated in the i^{th} step are satisfiable with respect to the set of constraints proved to be satisfiable in the $(i - 1)^{\text{th}}$ deduction step. In this way it is only necessary to prove the satisfiability of a small number of constraints in each deduction step, rather than proving the satisfiability of the whole set of constraints in each step. In the remainder of this section we demonstrate the applicability of theorem proving with constraint satisfaction to the analysis and partial synthesis of truss structures by means of three examples. In each example the problem definition is formulated using a theorem.

Let Δ represent the set of axioms described in the previous section. A theorem of Δ is a logical expression that can be deduced mechanically from Δ . In general a theorem may involve several unknown variables (unknown terms). These variables are resolved via constraint satisfaction in the deduction process.

In this context the theorem proving process has two interpretations: an *analysis interpretation* and a *synthesis interpretation*.

- The **analysis interpretation** is the process in which the theorem proving strategy yields the unknown forces and displacements associated with the known structural components.
- The **synthesis interpretation** is the process in which the theorem proving strategy generates new structural components such that the constraints associated with the whole structure are satisfiable.

In the following examples the variables are shown with the symbol '?'. Terms whose value are of no interest to the reader are left unspecified.

2.4.1 Example I (Structural Analysis)

The analysis of structure manifests itself in specifying a set of partially defined structural components as the argument of the predicate *structure*. The displacements and forces associated with the components are considered to be unknown. Consider the four-component truss structure shown in Figure 3. The theorem for analyzing this structure is composed of the partial definition of the attributes associated with the structural components and the predicate *structure* with the set of components as its argument.

The theorem for analyzing this structure is

$$\begin{aligned} &(Lsupport = sc^*(m^*(?, p^*(0, 0), d^*(0, 0)),) \wedge \\ &(Rsupport = sc^*(m^*(f^*(0, ?), p^*(120, 0), d^*(?, 0)),) \wedge \\ &(Load = lc^*(m^*(f^*(1, 0), p^*(120, 0), ?),) \wedge \\ &(Member = tc^*(g^*(10000, 1, ?), m^*(?, p^*(0, 0), ?), m^*(?, p^*(120, 0), ?),) \wedge \\ &structure([Lsupport, Rsupport, Load, Member]) \end{aligned}$$

The left support is denoted by *Lsupport*; its force vector is unknown, its position vector is $p^*(0, 0)$, and its displacement vector is $d^*(0, 0)$. The right support is denoted by *Rsupport*; its force vector is $f^*(0, ?)$ which means that the x component of the vector is zero and its y component is unknown, its position vector is $p^*(120, 0)$, and its displacement vector is $d^*(?, 0)$ which means that the x component of the vector is unknown and its y component is zero. The load component is denoted by *Load*; its force vector is $f^*(1, 0)$, its position vector is $p^*(120, 0)$, and its displacement vector is unknown. The truss component is denoted by *Member*; its modulus of elasticity is 10000, its cross-sectional area is 1, and its length is not specified, its force and displacement vectors are unknown, and its end positions are defined using the position vectors $p^*(0, 0)$ and $p^*(120, 0)$.

The above theorem can be logically deduced from Δ and as a result of this deduction the unknown terms become known. That is,

$$\begin{aligned} Lsupport &= sc^*(m^*(f^*(-1, 0), , ,)) \\ Rsupport &= sc^*(m^*(f^*(0, 0), , d^*(.012, 0),)) \\ Load &= lc^*(m^*(f^*(, , d^*(.012, 0),)) \\ Member &= tc^*(g^*(, , 120), m^*(f^*(-1, 0), , d^*(0, 0),), m^*(f^*(1, 0), , d^*(.012, 0),)) \end{aligned}$$

where the left support force vector is $f^*(-1, 0)$, the right support force and displacement vectors are $f^*(0, 0)$ and $d^*(0.012, 0)$, the load displacement vector is $d^*(0.012, 0)$, and the force and displacement vectors at the first and second joints of the truss component are $f^*(-1, 0)$, $d^*(0, 0)$, and $f^*(1, 0)$, $d^*(0.012, 0)$, respectively.

2.4.2 Example II (Symbolic Computation)

In this example the relationship between the applied load and the displacement under the load for the truss structure shown in Figure 3 is determined via theorem proving. This is accomplished by defining the magnitude of the load and the magnitude of the displacement associated with the load as symbols L and D , respectively. in the definition of the load component. That is

$$\begin{aligned} (Lsupport &= sc^*(m^*(?, p^*(0, 0), d^*(0, 0),))) \wedge \\ (Rsupport &= sc^*(m^*(f^*(0, ?), p^*(120, 0), d^*(?, 0),))) \wedge \\ (Load &= lc^*(m^*(f^*(L, 0), p^*(120, 0), d^*(D, ?),))) \wedge \\ (Member &= tc^*(g^*(10000, 1, ?), m^*(?, p^*(0, 0), ?), m^*(?, p^*(120, 0), ?),)) \wedge \\ structure &([Lsupport, Rsupport, Load, Member]) \end{aligned}$$

This theorem can be logically deduced from Δ . As a result we obtain

$$\begin{aligned} Lsupport &= sc^*(m^*(f^*(-L, 0), , ,)) \\ Rsupport &= sc^*(m^*(f^*(0, 0), , d^*(D, 0),)) \\ Load &= lc^*(m^*(f^*(, , d^*(D, 0),)) \\ Member &= tc^*(g^*(, , 120), m^*(f^*(-L, 0), , d^*(0, 0),), m^*(f^*(L, 0), , d^*(D, 0),)) \\ L &= 83.33 * D \end{aligned}$$

where the left support force vector is $f^*(-L, 0)$. The right support force vector is $f^*(0, 0)$, and its displacement vector is $d^*(D, 0)$. The load displacement vector is $d^*(D, 0)$. The force and displacement vectors at the first and second nodes of the truss component are $f^*(-L, 0)$, $d^*(0, 0)$, and $f^*(L, 0)$, $d^*(D, 0)$, respectively. And, the relationship between the load and displacement is $L = 83.33 * D$. Satisfiability of the constraints associated with the above theorem is proved if and only if the relationship $L = 83.33 * D$ holds true since the constraints are expressed in terms of the L and D symbols. Therefore, the relationship $L = 83.33 * D$ is derived from the set of constraints in the process of satisfying the constraints.

2.4.3 Example III (Partial Synthesis)

In this example we show how a partial structure can be defined as a theorem and how the theorem can be proved. Let us assume that the structure space consists of four nodes, and the partial structure is composed of a load component and a support component as shown in Figure 4. A theorem for synthesizing a four-component truss structure given the partial structure is

$$\begin{aligned} & (Support = sc^*(m^*(f^*(0, ?), p^*(120, 0), d^*(?, 0)),)) \wedge \\ & (Load = lc^*(m^*(f^*(1, 0), p^*(120, 0), ?),)) \wedge \\ & structure([Support, Load, X, Y]) \end{aligned}$$

The above theorem contains two partially defined components (*Support* and *Load*) and two completely undefined components (*X* and *Y*). This theorem can be logically deduced from Δ . As a result, several solutions are obtained. A portion of the search space for a solution is shown in Figure 5. The search space is depicted by a tree whose leaf nodes represent the possible solutions. Four such solutions are shown in the figure and they are defined below.

Solution (1) In this solution *X* and *Y* are synthesized as two support components. The spatial position of *X* and *Y* is defined by the position vectors $p^*(0, 0)$ and $p^*(120, 0)$, respectively.

$$\begin{aligned} Support &= sc^*(m^*(f^*(0, 0), p^*(120, 0), d^*(0, 0)),) \\ Load &= lc^*(m^*(f^*(1, 0), p^*(120, 0), d^*(0, 0)),) \\ X &= sc^*(m^*(f^*(0, 0), p^*(0, 0), d^*(0, 0)),) \\ Y &= sc^*(m^*(f^*(-1, 0), p^*(120, 0), d^*(0, 0)),) \end{aligned}$$

Solution (2) In this solution *X* is synthesized as a support component and *Y* is synthesized as a truss component. The spatial position of *X* is defined by the position vector $p^*(0, 0)$, and that of *Y* is defined by position vectors $p^*(0, 0)$ and $p^*(120, 0)$.

$$\begin{aligned} Support &= sc^*(m^*(f^*(0, 0), p^*(120, 0), d^*(0.012, 0)),) \\ Load &= lc^*(m^*(f^*(1, 0), p^*(120, 0), d^*(0.012, 0)),) \\ X &= sc^*(m^*(f^*(-1, 0), p^*(0, 0), d^*(0, 0)),) \\ Y &= tc^*(g^*(10000, 1, 120), m^*(f^*(-1, 0), p^*(0, 0), d^*(0, 0)), \\ &\quad m^*(f^*(1, 0), p^*(120, 0), d^*(0.012, 0))) \end{aligned}$$

Solution (3) In this solution *X* and *Y* are synthesized as two support components. The spatial position of *X* and *Y* are defined by the position vectors $p^*(0, 120)$ and $p^*(120, 0)$, respectively.

$$Support = sc^*(m^*(f^*(0, 0), p^*(120, 0), d^*(0, 0)),)$$

$$\begin{aligned}
Load &= lc^*(m^*(f^*(1, 0), p^*(120, 0), d^*(0, 0)),) \\
X &= sc^*(m^*(f^*(0, 0), p^*(0, 120), d^*(0, 0)),) \\
Y &= sc^*(m^*(f^*(-1, 0), p^*(120, 0), d^*(0, 0)),)
\end{aligned}$$

Solution (4) In this solution X is synthesized as a support component and Y is synthesized as a truss component. The spatial position of X is defined by the position vector $p^*(0, 120)$, and that of Y is defined by position vectors $p^*(0, 120)$ and $p^*(120, 0)$.

$$\begin{aligned}
Support &= sc^*(m^*(f^*(0, -1), p^*(120, 0), d^*(0.0339, 0)),) \\
Load &= lc^*(m^*(f^*(1, 0), p^*(120, 0), d^*(0.0339, 0)),) \\
X &= sc^*(m^*(f^*(-1, 1), p^*(0, 120), d^*(0, 0)),) \\
Y &= tc^*(g^*(10000, 1, 169.706), m^*(f^*(-1, 1), p^*(0, 120), d^*(0, 0)), \\
&\quad m^*(f^*(1, -1), p^*(120, 0), d^*(.0339, 0)))
\end{aligned}$$

Although we have shown only four solutions in this example, theorem proving generates all the valid solutions to a truss synthesis problem. It is possible, however, to prune the solution space by introducing conditions (constraints) in the definition of the theorem. For example, by specifying the constraint that *if either X or Y is a support component, then it must not be placed where the existing Support component is located, namely at $\langle 120, 0 \rangle$* . This constraint is formally expressed as

$$\begin{aligned}
(P \neq p^*(120, 0) \leftarrow X = sc^*(m^*(, P, ,))) \wedge \\
(Q \neq p^*(120, 0) \leftarrow Y = sc^*(m^*(, Q, ,)))
\end{aligned}$$

Then, the theorem becomes

$$\begin{aligned}
&(Support = sc^*(m^*(f^*(0, ?), p^*(120, 0), d^*(?, 0)),) \wedge \\
&(Load = lc^*(m^*(f^*(1, 0), p^*(120, 0), d^*(?, 0)),) \wedge \\
&(P \neq p^*(120, 0) \leftarrow X = sc^*(m^*(, P, ,))) \wedge \\
&(Q \neq p^*(120, 0) \leftarrow Y = sc^*(m^*(, Q, ,))) \wedge \\
&structure([Support, Load, X, Y])
\end{aligned}$$

Therefore, the solution space is reduced and the solutions (1) and (3) are eliminated.

3 Implementation

Logic programming is a declarative programming methodology based on predicate logic. Logic programming differs from imperative (procedural) programming in two ways.

1. In logic programming the domain knowledge is separated from the control strategy. The knowledge is specified in a declarative logic-based language and the control strategy is based on a formal theorem proving strategy.

2. The execution of a logic program lends itself to theorem proving. That is, the input of a logic program is a theorem and the program execution involves proving the truth or falsity of the theorem.

Logic programming provides a general purpose programming approach for engineering problem solving. As Lloyd said "the range of applications together with the simplicity, elegance, and unifying effect of logic programming assures it of an important and influential future. Logic programming is about to become the fundamental unit of computation [23]."

Prolog is a logic programming language developed for efficient execution of logic programs. Prolog uses resolution with unification for theorem proving. Prolog was used by Chan and Paulson [1] [24] for the analysis of truss structures. Their formulation differs from ours in two ways.

- Their formulation is static, that is, for each truss structure they need to explicitly identify the structure's equilibrium equations as constraints and explicitly formulate them as a set of Prolog rules. Our formulation is dynamic, that is, we provide only one predefined set of rules (axioms) for analyzing all truss structures.
- Their formulation uses local propagation for satisfying constraints. However, local propagation is not capable of solving simultaneous constraints. Our formulation is geared towards satisfying simultaneous constraints.

3.1 Constraint Logic Programming

Constraint logic programming is a generalization of logic programming wherein unification is replaced (generalized) by constraint satisfaction [25]. Constraint logic programming is particularly suitable for solving constraint-based problems. CLP($\mathcal{R}$) [26] is a constraint logic programming language for reasoning about constraints in the domain of real arithmetic. CLP($\mathcal{R}$) uses resolution with constraint satisfaction to reason about constraints. The CLP($\mathcal{R}$) constraint satisfaction mechanism handles equality equations (the same as Prolog) as well as linear and nonlinear constraints. Equality equations are of the form $A = B$ where A and B are terms. The linear and nonlinear constraints are defined using terms and arithmetic operators. $A + B > C - D$ is an example of a linear constraint. A constraint logic program for the analysis and partial synthesis of truss structures in CLP($\mathcal{R}$) is under implementation by the authors.

4 Summary

The behavior of truss structures is modeled using an incremental approach. This approach lends itself to modeling the behavior of structures as the set union of the behavior of their constituent components. The incremental model is formalized using a set of axioms. The axioms are formulated in predicate logic and they provide a means for the analysis and partial synthesis of truss structures. An incomplete definition of a truss structure is considered as a theorem and a theorem proving strategy is used to prove the truth or falsity

of the theorem. As a result of the theorem proving process, the incomplete definition of the structure becomes complete.

The theorem proving process has two interpretations: an analysis interpretation and a synthesis interpretation. If the incompleteness in the definition of the structure is due to the unknown forces and displacements of the structural components, then the theorem proving process is interpreted as the structural analysis process and it determines the unknown forces and displacements associated with the structural components. If the incompleteness in the definition of the structure is due to unknown components, then the theorem proving process is interpreted as the structural synthesis process and it generates alternative truss configurations by determining the appropriate structural components so that the incomplete definition of the structure becomes completed.

The constraint logic programming methodology [26] is suggested as a vehicle for implementing a computer program for the analysis and partial synthesis of truss structures. This methodology enables us to use theorem proving with constraint satisfaction to reason about constraints and thereby to analyze and partially synthesize truss structures using a declaratively defined constraint logic program.

5 Implications for Structural Design Automation

The logic-based formulation for the analysis and synthesis of truss structures, and the constraint logic programming for implementing the formulation, have several highly desirable implications for structural design automation. The logic-based approach for representing knowledge and the theorem proving with constraint satisfaction approach for reasoning about the knowledge provide (1) a uniform framework for automating structural design and (2) a mechanism for maintaining the integrity of design databases. Additionally, the constraint logic programming implementation of the formulation provides a suitable methodology for parallel processing of structural design problems.

A Uniform Framework for Structural Design: This paper dealt with the analysis and partial synthesis of truss structures using numeric constraints only. These constraints are formulated as axioms and are processed using theorem proving. To prove a theorem based on the axioms, a theorem prover engages in proving the satisfiability of the constraints associated with the theorem. Since these constraints are defined over the domain of real numbers one can argue that the formulation is numeric-based and that the theorem prover is involved in reasoning about numeric knowledge only. However, this need not be the case. In principle, the theorem proving strategy provides a formal and uniform framework for reasoning about both numeric and non-numeric knowledge.

Efficiency of the synthesis of truss structures, as demonstrated in this paper using numeric constraints, can be improved by including heuristics (non-numeric constraints) in the formulation. For example, truss structures can be synthesized according to a set of heuristics that are a function of structural stability considerations. Paul and Neville [27] provided six heuristics for synthesizing structures with one dimensional (truss and beam) components based on structural stability. Also, the design evaluation task involves processing

non-numeric constraints associated with design standards. Therefore, the synthesis, analysis, and evaluation of structural systems involve both numeric and non-numeric constraints. It is possible to capture the knowledge associated with these activities using numeric and non-numeric constraints formulated as axioms and process them using a theorem proving strategy. The synthesis, analysis, and evaluation of structural systems involve both numeric and non-numeric knowledge. We believe that it is possible to capture the knowledge associated with these activities using a set of axioms and to process the knowledge using a theorem proving strategy.

A Mechanism for Maintaining the Integrity of Design Databases: There is a close relationship between relational databases and predicate logic [28] [23]. A logic program is composed of a set of facts and a set of rules. The facts can be viewed as database relations and the rules can be viewed as integrity constraints imposed on data in the database.

The set of axioms that describe the behavior of truss structures can be viewed as the integrity constraints for a truss design relational database. These constraints define the behavioral relationships between structural components and they can be used for checking and enforcing the integrity of the database.

Parallel Processing of Structural Design Problems: Logic programming is a particularly suitable methodology for parallel processing of engineering design problems. Two types of parallelism can be used with logic programs: *And-Parallelism* and *Or-Parallelism*. The And-Parallelism lends itself to parallel execution of the predicates forming a rule (axiom); Or-Parallelism lends itself to parallel execution of rules themselves. Both types of parallelisms can be utilized in a constraint logic program to enhance the efficiency of the program execution. For example, Or-Parallelism can be used to synthesize all the solutions of the truss synthesis problem of example III in parallel.

6 Further Work

The approach taken here is to be extended in two directions. From a theoretical standpoint we intend to explore the possibility of axiomatizing the structural design process for different structural systems so that reasoning about their physical behavior can be facilitated. This involves capturing the design knowledge associated with synthesis, analysis, and evaluation activities using axioms, and utilizing theorem proving strategy to reason about these activities. Mathematical logic provides a uniform framework for which these activities can be formalized and studied.

The importance of design databases [29] [30] [31] in general, and generative (deductive) design databases [32] [33] in particular, as well as design standards processing systems [34] in the automation of structural design is evident to most researchers. From a more practical standpoint we will investigate the use of constraint logic programming for developing engineering design aids such as generative design databases, standards processing systems, and analysis and synthesis systems for different types of structures. We postulate that constraint logic programming provides a uniform and mathematically sound methodology for

developing such systems. In this context we will consider issues such as efficiency, generality, modifiability, ease of use, reasoning capability, modularity, integrability, and robustness of the constraint logic programming.

7 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

- [1] Chan (1986) *Logic Programming For Managing Constraint-Based Engineering Design*, PhD Thesis, Stanford University.
- [2] Dixon, J.R. (1988) "On Research Methodology Towards a Scientific Theory of Engineering Design," *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, Volume 4, Number 3.
- [3] Newsome, S.L., and Spillers, W.J. (1988) Design Theory'88 Preprints, The 1988 NSF Grantee Workshop on Design Theory and Methodology, Rensselaer Polytechnic Institute, Troy, New York, June 3-5.
- [4] Rabins, M.J., Ardayfio, D., Clark, H., Fenves, S., Nadler, G., Richardson, H., and Seireg, A. (1986) Goals and Priorities for Research in Design Theory and Methodology, A Report to the National Science Foundation, The American Society of Mechanical Engineers.
- [5] Barwise, J., and Feferman, S. (1985) *Prespectives in Mathematical Logic, Model-Theoretic Logics*, Springer-Verlag.
- [6] Barwise, J. (1977) *Handbook of Mathematical Logic*, North-Holland Publishing Company.
- [7] Hayes, P.J. (1977) "In Defense of Logic," Proceedings of the Fifth International Joint Conference on AI, Pages 559-565.
- [8] 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), Pages 336-341.
- [9] 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.
- [10] Frost, R. (1986) *Introduction to Knowledge Base Systems*, Macmillan Publishing Company, New York.

- [11] Maher, M.L. (1985) *HI-RISE: A Knowledge-Based Expert System For The Preliminary Structural Design of High Rise Buildings*, Report No. R-85-146, Department of Civil Engineering, Carnegie-Mellon University.
- [12] Sriram, D. (1987) "ALL-RISE: A Case Study in Constraint-Based Design," *Artificial Intelligence in Engineering*, Volume 2, Number 4, Pages 186-203.
- [13] Stiny, G. (1980) "An Introduction to Shape and Shape Grammars," *Environment and Planning B*, Volume 7, Pages 343-351.
- [14] Fenves, S.J., and Baker, N.C. (1988) "Grammars for Functional and Spatial Reasoning in Design," *Proceedings of the Fifth Conference on Computing in Civil Engineering: Microcomputers to Supercomputers*, Will, K.M. (Editor), The American Society of Civil Engineers, Pages 819-828.
- [15] Kim, S.H., and Suh, N.P. (1985) "Application of Symbolic Logic to the Design Axioms," *Robotics and Computer-Integrated Manufacturing*, Volume 2, Number 1, Pages 55-64.
- [16] Davis, E. (1988) "A Logical Framework for Commonsense Prediction of Solid Object Behavior," *Artificial Intelligence in Engineering*, Volume 3, Number 3, Pages 125-140.
- [17] Hayes, P.J. (1985) "Naive Physics I: Ontology for Liquids," in *Formal Theories of the Commonsense World*, Hobbs, J.R., and Moore, R.C. (Editors), Ablex Publishing Corporation, Pages 71-107.
- [18] Loveland, D. (1978) *Automated Theorem Proving: A Logical Basis*, North-Holland.
- [19] Chang, C.L., and Lee, R.C. (1973) *Symbolic Logic and Mechanical Theorem Proving*, Academic Press.
- [20] Rich, E. (1983) *Artificial Intelligence*, McGraw-Hill.
- [21] Genesereth, M.R., and Nilsson, N.J. (1988) *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann.
- [22] Jaffer, J., and Lassez, J-L. (1987) "From Unification to Constraints," *Proceedings of the Sixth Conference on Logic Programming*, Furukawa, H., Tanaka, H., and Fujisaki, T. (Editors), Pages 1-18.
- [23] Lloyd, J.W. (1987) *Foundations of Logic Programming*, Second Edition, Springer-Verlag.
- [24] Chan, W.T., and Paulson, B.C. Jr. (1987) *Exploratory Design Using Constraints, Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, Volume 1, Number 1, Pages 59-71.
- [25] Jaffer, J., Lassez, J-L., and Maher, M.J. (1986) "Logic Programming Language Scheme," in *Logic Programming Functions, Relations, and Equations*, DeGroot, D., and Lindstrom, G. (Editors), Pages 441-467.

- [26] Jaffer, J., Michaylov, S. (1986) "Methodology and Implementation of a CLP System," Proceedings of the Fourth International Conference on Logic Programming, Lassez, J-L. (Editor), Pages 196-218.
- [27] Paul, G.H., and Neville, G.E. (1988) "Heuristics Evaluation of Partial Structural Designs," Proceedings of the 1988 ASME International Computers in Engineering Conference, The American Society of Mechanical Engineers, Volume 1, Tipnis, V.A., and Pattun, E.M. (Editors), Pages 161-166.
- [28] Gallaire, H., and Minker, J. (1987) *Logic and Database*, Plenum Press.
- [29] Murthy, T.S., and Arora, J.S. (1986) "DataBase Design Methodology for Structural Analysis and Design Optimization," *Engineering with Computers*, Volume 1, Number 3, Pages 149-160.
- [30] Nicklin, P.J., Powell, G.H., and Hollings, J.P. (1985) "Hierarchical Data Management for Structural Analysis," *Engineering with Computers*, Volume 1, Number 1, Pages 45-54.
- [31] Rasdorf, W.J., Ulberg, K.J., and Baugh, J.W. (1987) "A Structure-Based Model of Semantic Integrity Constraints for Relational databases," *Engineering with Computers*, Volume 2, Pages 31-39.
- [32] Rasdorf, W.J., and Salley, G.C. (1985) "Generative Engineering Databases- Toward Expert Systems," *Computers and Structures*, Volume 20, Number(s) 1-3, Pages 11-15.
- [33] 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.
- [34] Garrett, J.H., and Fenves, S.J. (1987) "A Knowledge-Based Standards Processor for Structural Component Design," *Engineering with Computers*, Volume 2, Number 4, Pages 219-238.

Figure 1: The Force Equilibrium Equations of a Node.

Figure 2: Incremental Structural Modeling

Figure 3: A Four-Component Truss Structure

Figure 4: The Structure Space and Partial Structure for Example III

Figure 5: The Partially Defined Structure for Example III