Contents

A Logic-Based Approach for Modeling the Organization of Design Standards

William J. Rasdorf*, Member, ASCE
Sivand Lakmazaheri†, Associate Member, ASCE

Abstract

Several studies have been conducted on representing and processing design standards for design automation. One of the main outcomes of these studies is the Standards Analysis, Synthesis, and Expression (SASE) model. To extend the utility of the SASE model for processing (reasoning about) design standards a logic-based approach is proposed. This approach provides (1) a formal language, founded on predicate logic, for representing the standard and (2) a mechanical means for reasoning about the standard using the language. The formal language is used to model the overall organization of a portion of the AISC design specification. The model, called the formal organizational submodel, is composed of a set of axioms which capture the relationships between the classifiers and the provisions of the standard. Reasoning about the formal organizational submodel is accomplished using the resolution theorem proving strategy.

This paper's main contribution is its use of predicate logic and theorem proving for representing and reasoning about the organization of design standards, respectively. This approach facilitates the processing of the organization of design standards in ways which have not been addressed before.

Keywords: *Automated Reasoning, Predicate Logic, Theorem Proving, Standards Processing.*

1 Introduction and Overview

In general, a design standard is composed of a set of provisions, where each provision defines a set of rules (constraints) that have to be satisfied in a given design circumstance. The processing of a design standard involves identifying and satisfying the applicable constraints in a design circumstance. Automating the processing design standards has been an important research area in civil engineering since the late 1960's. One of the outcomes of the research efforts in this area has been the SASE model [Fenves87] for representing and processing design standards. A number of other related research and development efforts have been directed towards automating this model [Nyman73,Garrett86,Rasdorf87,Dym88,Cronembold88,Lopez89].

The SASE model consists of two submodels: a submodel for the overall organization of the design standard and a submodel for representing and processing the standard's provisions. The first

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

submodel, herein referred to as the *organizational submodel*, involves relating a set of classifiers to the set of provisions of the standard. In a design situation, given the appropriate set of classifiers, it is possible to determine the applicable provisions using this submodel. The second submodel defines the logic of the content of a standard's provisions using a set of rules which it processes. In a design situation, given the set of applicable provisions, the rules associated with each provision are processed (1) to check the conformity of the structural component with the standard or (2) to design the structural component based on the standard.

Using the SASE model for designing involves three steps:

1. Identifying the appropriate classifiers,
2. Determining the applicable provisions given the classifiers, and
3. Processing provisions using rules.

The first and second steps define the processing of the organizational submodel of the standard, and the third step defines the processing of the standard's provisions.

Automatic identification of a standard's classifiers has not been effectively dealt with in the past, and only recently has it begun to emerge as a research issue [Elam88,Lopez89]. However, automation of Steps 2 and 3 has been addressed by many researchers for structural component design [Fenves69,Garrett86,Rasdorf87,Elam88,Cronembold88].

This paper is concerned with processing (reasoning about) the organizational submodel of the AISC design specification [AISC80] (step 2 above) by establishing the relationships between the classifiers and the provisions of a portion of the Specification. The conventional approach for processing this organizational submodel has involved using a classifier or a set of classifiers to determine the applicable provisions. However, this paper takes a different approach: we propose to draw all the logical conclusions that can be drawn from the relationships between the classifiers and the provisions of the AISC design Specification.

Reasoning (deductive reasoning) involves drawing conclusions from known knowledge. To automate reasoning one needs to have a language for representing knowledge and a mechanism for manipulating knowledge in order to draw a conclusion [Wos84]. In the context of the AISC organizational submodel, reasoning involves viewing the submodel as a collection of facts and drawing conclusions based on the facts. Hence, to automate reasoning about the AISC organizational submodel, one has to have a language for representing the submodel and a mechanical means for manipulating the submodel.

One powerful way for representing and manipulating, and thus reasoning about, the organizational submodel is predicate logic and theorem proving. One use of predicate logic and theorem proving, in the context of design standards, for checking the robustness of the logic of provisions has been demonstrated by Jain, Law, and Krawinkler [Jain89].

Predicate logic is used for developing a *formal language* ($\mathcal{L}$) for representing the organizational submodel of the AISC design specification. Using $\mathcal{L}$, a set of statements (axioms), denoted by Δ, is developed that represent the relationships between the classifiers and the provisions of the standard.

The formal language and the axioms constitute the *formal organizational submodel* ($\mathcal{M}(\mathcal{L}, \Delta)$) of the standard. Using this formal submodel it is possible to deductively reason about the organization of the standard in many ways. This paper discusses the formal organizational submodel for the AISC design specification and the ability of the submodel to reason about the organization of the standard.

To set the stage for describing $\mathcal{M}(\mathcal{L}, \Delta)$ and to show the utility of the model in reasoning about its domain of discourse (i.e., the organization of the AISC design specification), predicate logic and theorem proving are discussed in Section 2. A conceptual view of the organizational submodel is provided in Section 3. The formal organizational submodel is presented in Section 4 and a logic programming implementation of $\mathcal{M}(\mathcal{L}, \Delta)$ is presented in Section 5. A summary of the paper is provided in Section 6.

2 Predicate Logic and Theorem Proving

This section presents a brief overview of the use of predicate logic for knowledge representation and the unification strategy for theorem proving.

2.1 Predicate Logic

Mathematical logic is a means for representing and reasoning about a problem of interest in a formal way. This involves identifying and representing the set of objects and their inter-relationships that constitute the problem and employing an inference strategy for reasoning about the problem's domain. The term "object" has a broad usage, for it can be taken to mean either a physical or an abstract entity. In short, an object is referred to any entity that we want to say something about. Herein, the set of all objects in the domain of interest is called the *universe of discourse* or simply U_d.

Predicate logic provides a systematic way for representing a U_d using a limited number of constructs. These constructs, described below, are variables, object constants, function constants, relation constants, functional expressions, atomic formulas, logical operators, and quantifiers.

- A variable is a symbol that can be instantiated to any object in U_d.
- An object constant is a symbol representing an object in U_d.
- A function constant (e.g., $+$) defines a function on a subset of U_d.
- A relation constant (e.g., $>$) defines a relation on a subset of U_d.
- A functional expression is an statement consisting of a function constant and several object constant and/or variables. For example, the expression $+(3, 4)$ is a functional expression that maps the object constants 3 and 4 to the object constant 7. The number of object constants and variables in a functional expression is called the function constant's arity. In the above example the function constant $+$ has an arity of 2; it is a binary function.

- An atomic formula is formed using a relation constant and a set of object constants and/or variables. For example, $>(2,1)$ is an atomic formula where $>$ is a relation constant, and 2 and 1 are object constants. The number of object constants and variables in an atomic formula is called the relation constant's arity. In the above example the relation constant $>$ has an arity of 2; it is a binary relation.
- The logical operators mainly used are negation ($\neg$), disjunction ($\vee$), conjunction ($\wedge$), and implication ($\Rightarrow$).
- Two quantifiers are commonly used in predicate logic: the universal quantifier ($\forall$) and the existential quantifier ($\exists$). Quantifiers provide a way of referring to the objects in a U_d without specifically identifying the individual objects [Genesereth88].

Using the above constructs it is possible to represent the objects and their inter-relationships in a U_d using expressions called *well-formed formulas* or $wffs$. Well-formed formulas are defined as follows [Lloyd87], with variables and object constants being called *terms*.

- If f is an n-ary function constant and t_1, t_2, $\cdots$, and t_n are terms, then $f(t_1, t_2, \cdots, t_n)$ is a term.
- If p is a relation constant and t_1, t_2, $\cdots$, and t_n are terms, then $p(t_1, t_2, \cdots, t_n)$ is a wff.
- If A and B are $wffs$, then $\neg A$, $A \vee B$, $A \wedge B$, and $A \Rightarrow B$ are $wffs$.
- If A is a wff and x is a variable appearing in A, then $\forall_x A$ and $\exists_x A$ are $wffs$.

2.2 Theorem Proving

Theorem proving is a mechanical procedure for proving that a given wff called a *theorem* is a logical consequence of a set of $wffs$ called *axioms*. Let Δ represent a set of axioms and τ represent a wff. Then, τ is said to be a theorem of Δ if it can be logically deduced from Δ. The notation $\Delta \vdash \tau$ signifies that τ is a theorem of Δ [Genesereth88].

Resolution is a theorem proving strategy which proves a theorem by *refutation*. That is, the negated theorem is proved to contradict the axioms. In resolution a set of $wffs$ called *clauses* are used to express both Δ and τ. A clause is a wff with only two types of logical operators: the negation ($\neg$) and the disjunction ($\vee$).

The resolution proof procedure is iterative in nature. In each iteration two clauses are resolved into one. The procedure yields the empty clause ($\{\}$) when $\Delta \vdash \tau$. 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*.

The following example illustrates the resolution theorem proving strategy. In this example variables are shown in lowercase Roman letters, relation constants are shown in uppercase Roman letters, and object constants are shown in lowercase Greek letters.

Let $\Delta = \{A_1, A_2, A_3, A_4\}$ where

$A_1 = \{C(y,z) \vee \neg B(x,z) \vee \neg A(x,y)\}$
$A_2 = \{D(z) \vee \neg B(x,z) \vee \neg C(y,z)\}$
$A_3 = \{B(\alpha,\beta)\}$
$A_4 = \{A(\alpha,\gamma)\}$

and let $\tau = \{D(z)\}$. The following steps demonstrate the deduction process.

Step 0: Negate τ. That is, $\{\neg D(z)\}$.

Step 1: From $\{\neg D(z)\}$ and A_2, deduce $\{\neg B(x,z) \vee \neg C(y,z)\}$.

Step 2: From $\{\neg B(x,z) \vee \neg C(y,z)\}$ and A_3 and substitution $\theta = \{x/\alpha, z/\beta\}$, deduce $\{\neg C(y,\beta)\}$.

Step 3: From $\{\neg C(y,\beta)\}$ and A_1 and substitution $\theta = \{z/\beta\}$, deduce $\{\neg B(x,\beta) \vee \neg A(x,y)\}$.

Step 4: From $\{\neg B(x,\beta) \vee \neg A(x,y)\}$ and A_3 and substitution $\theta = \{x/\alpha\}$, deduce $\{\neg A(\alpha,y)\}$.

Step 5: From $\{\neg A(\alpha,y)\}$ and A_4 and substitution $\theta = \{y/\gamma\}$, deduce $\{\}$.

Thus, $\neg\tau$ contradicts Δ, therefore, $\Delta \vdash \tau$.

3 The Organizational Submodel

This section presents a conceptual model for representing and processing the organizational submodel of a portion of the AISC design Specification.

3.1 Representing the Submodel

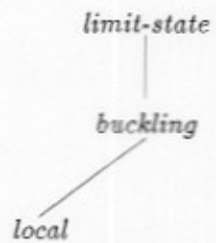
An organizational submodel is composed of a set of tree structures, a set of provisions, and the relationship between the trees and the provisions. Figure 1 depicts an organizational submodel for a portion of the AISC Specification. The relationship between the trees and the provisions of the submodel is shown by the links that connect the provisions to the leaf nodes of the trees. The organizational submodel of Figure 1 is described in terms of identifiers, classifier subtrees, subclassifiers, and classifiers.

Identifiers: The nodes of the tree structures in the organizational submodel are called *identifiers*. Identifiers are keywords used for describing (characterizing) design circumstances and for identifying the applicable design standard's provisions in a design circumstance. The organizational submodel can be viewed as a set of identifier hierarchies, as shown in Figure 1, coupled with a subset of the standard's provisions.

Classifier Subtrees: The tree structured identifier hierarchies of the organizational submodel are called *classifier subtrees*. The three classifier subtrees in the organizational submodel of Figure 1 are the *object-type* subtree, the *limit-state* subtree, and the *stress-state* subtree. Each subtree depicts the overall hierarchical organization of its identifiers. For example, consider the limit-state classifier subtree shown in Figure 1. This subtree is composed of six identifiers, namely, *limit-state*, *yield*, *buckling*, *local*, *global*, and *tearout*. Furthermore, the child nodes of the subtree are considered to be instances of their respective parent node. For example, *yield*, *buckling*, and *tearout* are three instances of *limit-state*, and *local* and *global* are two instances of *buckling*.

These classifier subtrees organize the identifiers in such a way that any design circumstance can be characterized using the limit-state, object-type, and stress-state identifiers associated with a structural component of interest. The organization of the AISC design Specification has been successfully modeled using these three classifier subtrees [Nyman73].

Subclassifiers: A *subclassifier* is a complete path from the root node to a leaf node of a subtree. For example,



is a subclassifier of the limit-state subtree shown in Figure 1. Herein, the set notation is used for representing subclassifiers. The above subclassifier is represented using the set

$$\{limit\text{-}state, buckling, local\}.$$

In general, each leaf node of a classifier subtree is connected to a set of provisions as shown in Figure 1. This implies that the provisions that are connected to the leaf node apply to the design circumstance that is partially characterized by the subclassifier whose last element is the leaf node. For example, if the subclassifier $\{limit\text{-}state, buckling, local\}$ is used to partially characterize a design circumstance, then Figure 1 shows that $\{AISC1.10.5, AISC1.9.1.2, AISC1.9.2.1\}$ is the set of applicable provisions in that circumstance.

Classifiers: To completely characterize any design circumstance, a subclassifier from each classifier subtree is needed. The set of subclassifiers that completely characterize a design circumstance is called a *classifier*. For example, the three subclassifiers

$$\{object\text{-}type, steel, i\text{-}shaped, hot\text{-}rolled, column\},$$

$\{limit-state, buckling, global\}$, and
$\{stress-state, axial, compression\}$

of Figure 1 completely define a design circumstance, thus forming a classifier. The applicable provisions of a classifier are derived from the set intersection of the provision sets associated with each of its three constituent subclassifiers. That is,

$\{AISC1.5.1.3.1, AISC1.5.1.3.2\} \cap \{AISC1.5.1.3.1, AISC1.5.1.3.2, AISC1.5.1.4.5\} \cap$
$\{AISC1.5.1.3.1, AISC1.5.1.3.2\}$,

which yields

$\{AISC1.5.1.3.1, AISC1.5.1.3.2\}$.

Since the organizational submodel of Figure 1 is composed of three classifier subtrees, all the classifiers associated with this submodel have to be defined using three subclassifiers.

3.2 Processing the Submodel

Generally, the organizational submodel of a design standard as presented above can be processed in two ways: (1) by processing a single classifier subtree and (2) by processing multiple classifier subtrees. All examples in this section refer to Figure 1.

3.2.1 Processing Single Classifier Subtrees

A single classifier subtree can be processed in several ways, as discussed below.

A: Given a complete or an incomplete subclassifier, determine its corresponding provisions. This is done by traversing the subtree, starting from the root node, until a leaf node is reached. Then determine the applicable provisions by following the path from the leaf node to the provision set. For example, given the subclassifier $\{limit-state, yield\}$, the limit-state subtree can be traversed to obtain the provision set $\{AISC1.5.1.2.1, AISC1.5.1.4\}$. Or, given the incomplete subclassifier $\{limitstate, buckling, X\}$, the limit-state subtree can be traversed to obtain all the applicable provisions. Two sets of provisions can be found. One of the sets is $\{AISC1.5.1.2.1, AISC1.5.1.4\}$ which corresponds to the subclassifier when X instantiates to *local*. The other set of provisions is $\{AISC1.5.1.3.1, AISC1.5.1.3.2, AISC1.5.1.4.5\}$ which corresponds to the subclassifier when X instantiates to *global*.

B: Given a provision or a set of provisions, determine their corresponding subclassifier(s), thus partially describing the design circumstance(s) under which a set of provisions become applicable. This is done by following the path from the given provision(s) to their corresponding leaf node of the subtree and traversing the tree from the leaf node to its root node. For example, given the provision "AISC 1.5.1.2.2", the limit-state subtree can be traversed to define $\{limit-state, tearout\}$ as the subclassifier for the given provision.

The above items deal with identifying one part of the organizational submodel given another part of the submodel. It is also possible to process the submodel for detecting certain properties associated with the subclassifier submodel itself. Two such properties are discussed below.

C: The property of having more than one subclassifier in a classifier subtree is attributed to the standard's provisions and it is called the *abundance* property. A provision is abundant if it has more than one subclassifier in a classifier subtree. One of the desired characteristics of an organizational submodel is the absence of abundant provisions. Since this is not an inherent characteristic of design standards, it can be checked using the organizational submodel of the standard. The submodel presented in Figure 1 is free of abundant provisions. However, to exemplify the abundance property, let us assume that the provision "AISC 1.5.1.4" is also connected to the "tearout" leaf node of the limit-state subtree. Then, the provision "AISC 1.5.1.4" would be abundant since it would have two subclassifiers, namely, $\{limit-state, tearout\}$ and $\{limit-state, yield\}$.

D: Another desired characteristic of the organizational submodel is the association of each of its subclassifiers with one or more provisions. A subclassifier which is not connected to any provision is called *free*. The existence of a free subclassifier indicates the presence of a design circumstance not addressed by the standard. Free subclassifiers can be identified by traversing every branch of the subtrees until a leaf node is reached. For example, the subclassifier

$$\{object-type, steel, i-shaped, hot-rolled, beam-column\}$$

in the object-type subtree and the subclassifier

$$\{stress-state, axial, tension\}$$

in the stress-state subtree are free.

3.2.2 Processing Multiple Classifier Subtrees

Multiple classifier subtrees can be processed in several ways as discussed below.

E: Given a classifier (i.e., a set of subclassifiers), determine its corresponding provisions. This involves identifying the applicable provisions for each subclassifier and determining their set intersection as discussed in Section 3.1.

F: Given a provision or a set of provisions, determine their corresponding classifier(s). This involves identifying the set of subclassifiers (one subclassifier for each classifier subtree) that is associated with with the provisions. For example, given the provision set

$$\{AISC1.5.1.3.1, AISC1.5.1.3.2\},$$

the corresponding design circumstance is characterized by the following classifier (i.e., three subclassifiers):

$\{limit-state, buckling, global\}$,
$\{stress-state, axial, compression\}$, and
$\{object-type, steel, i-shaped, hot-rolled, column\}$.

G: Given an incomplete classifier (a classifier with some known and some unknown subclassifiers), determine all of its unknown subclassifiers, thus completing a characterization of a given

design circumstance that is initially incomplete. This is accomplished by first finding the provision set associated with the known part of the classifier and then determining the unknown part of the classifier based on the determined provision set. For example, given a classifier with one known subclassifier $\{limit-state, tearout\}$ and two unknown subclassifiers (X,Y), the unknown subclassifiers are determined as follows. First, the provision set associated with the known subclassifier is identified, that is, $\{AISC1.5.1.2.2\}$. Then, the unknown subclassifiers are determined using the set $\{AISC1.5.1.2.2\}$. This gives

$$X = \{object-type, steel, i-shaped, hot-rolled, beam\}$$
$$Y = \{stress-state, bending, shear\}.$$

Again, the above items deal with identifying one part of the organizational submodel given another part of the submodel. It is also possible to process the submodel for detecting certain properties associated with the classifier submodel itself. Two such properties are discussed below.

H: A provision which has the same number of subclassifiers as there are classifier subtrees is *complete* if each of its subclassifiers belongs to a distinct classifier subtree. This is referred to as the completeness property. If there exists a classifier subtree such that none of its subclassifiers are associated with a particular provision that is present in all other classifier subtrees, then the provision is incomplete. If a provision is not incomplete, then it is complete. The provisions shown in Figure 1 are all complete. However, to exemplify the incompleteness of a provision, assume that the provision "AISC 1.5.1.2.2" is not connected to the "*tearout*" leaf node of the limit-state subtree. Then, provision "AISC 1.5.1.2.2" would be incomplete since it does not have a subclassifier in the limit-state subtree.

I: A *complete* and *not-abundant* provision is said to be *unique*. Uniqueness is a desired property for all of a standard's provisions. However, in general, uniqueness which is not an inherent property of design standards, needs to be checked using the organizational submodel of the standard.

As seen above, the organizational submodel for a design standard can be processed in several ways. In the past, the processing of an organizational submodel was limited to (1) identifying applicable provisions given the classifiers, and (2) identifying the corresponding classifiers given the provisions [Garrett86,Elam88]. In this paper additional processing is made possible by allowing the submodel to be processed in the extended manner as discussed in items A through H above.

4 The Formal Organizational Submodel

In this section we illustrate how the organizational submodel of Figure 1 can be formally represented using predicate logic and how it can be reasoned about using theorem proving.

4.1 Representing the Formal Submodel $\mathcal{M}(\mathcal{L}, \Delta)$

The organizational submodel is viewed as two sets: the set of classifiers (C) and the set of provisions (P) as shown in Figure 2. The submodel also embodies the relations between the two sets. Notice

that the elements of C themselves are sets. That is, each C_i is composed of three elements, S_{i_1}, S_{i_2}, S_{i_3}, where S_{i_j} is a subclassifier. In turn, each S_{i_j} is a set of identifiers. The formal organizational submodel is formulated by capturing these sets and their interrelationship using predicate logic.

The formal organizational submodel is composed of the formal language $\mathcal{L}$ and a set of axioms Δ, both of which are discussed below.

4.1.1 The Formal Language: $\mathcal{L}$

A language of predicate logic is used for representing the formal organizational submodel. This language is denoted by $\mathcal{L}$; its vocabulary is composed of a set of functions, a set of relations, a set of logical operators, and two quantifiers; and its well-formed formulas are defined as discussed is Section 2.1.

Functions

- *head(X)*: This function returns the first element of the set X.
- *tail(X)*: This function returns all of the elements of the set X except its first element.

Relations

Problem Dependent

- *abundant(P)*: P is an abundant provision.
- *complete(P)*: P is a complete provision.
- *root(B)*: B is the root node of a classifier subtree.
- *cset(C,B)*: C is the set of subclassifiers in the subtree whose root node is B.
- *connected(X,P)*: X is a leaf node connected to the provision P.
- *free(C,B)*: C is a free subclassifier in the classifier subtree whose root node is B.
- *identifier(X)*: X is an identifier.
- *leafnode(X)*: X is a leaf node of a classifier subtree.
- *link(X,Y)*: X is the parent of Y in a classifier subtree.
- *r11(X,P)*: X is a subclassifier for the provision P.
- *r1n(X,T,S)*: X is a subclassifier for the set of provisions T. T is a subset of S where S is the set of all the standard's provisions.

- *rn1(T,P,B,C)*: T is a set of subclassifiers for the provision P in the classifier subtree whose root node is B. T is a subset of C where C is the set of all the subclassifiers in the subtree.
- *path(X,T)*: T is a set of identifiers defining a path from the node X to a leaf node of a classifier subtree.
- *provision(P)*: P is a provision.
- *pset(P)*: P is the set of all the standard's provisions.
- *subclassifier(X)*: X is a subclassifier.
- *unique(P)*: P is a unique provision.

Problem Independent

- *belongsto(X,Y)*: X is an element of the set Y.
- *equal(X,Y)*: X and Y are identical.
- *intersect(X,Y,L)*: L is the set intersection of X and Y.
- *last(L,Z)*: Z is the last element of the set L.
- *union(X,Y,Z)*: Z is the set union of X and Y.

Logical Operators

- Implication ($\Rightarrow$)
- Conjunction ($\wedge$)
- Disjunction ($\vee$)
- Negation ($\neg$)

Quantifiers

- Universal quantifier ($\forall$)
- Existential quantifier ($\exists$)

Although the above quantifiers are a part of the language, they are not explicitly used in writing the valid statements of the language. Rather, they are used implicitly in expressing axioms and theorems. All the variables in the axioms are universally quantified. All the variables in the theorems are existentially quantified.

4.1.2 The Axioms: Δ

The axioms of the formal organizational submodel are defined using two types of $wffs$: *unit clauses* and *proper axioms*. A unit clause, or a *fact*, is a special kind of axiom which is composed of a single relation; an axiom with no logical operators. The facts and the proper axioms of $\mathcal{M}(\mathcal{L}, \Delta)$ are discussed below.

Facts: Facts, used to express a portion of $\mathcal{M}(\mathcal{L}, \Delta)$, represent the following: Identifiers, provisions, the parent-child relationship between the nodes of the classifier subtrees, the root node of the subtrees, leaf nodes of the subtrees, and the connection between leaf nodes and provisions.

Identifiers are represented using the relation '*identifier*,' as shown below.

$$identifier(limit\text{-}state)$$
$$identifier(yield)$$
$$identifier(buckling)$$
$$identifier(local)$$
$$identifier(global)$$
$$identifier(tearout)$$
$$identifier(stress\text{-}state)$$
$$\vdots$$
$$identifier(object\text{-}type)$$
$$\vdots$$

Provisions are represented using the relation '*provision*,' as shown below.

$$provision(\text{“}AISC1.5.1.2.1\text{”})$$
$$provision(\text{“}AISC1.5.1.4\text{”})$$
$$provision(\text{“}AISC1.10.5\text{”})$$
$$\vdots$$

The parent-child relationship between classifier subtree nodes is represented using the relation '*link*,' as shown below.

$$link(limit\text{-}state, yield)$$
$$link(limit\text{-}state, buckling)$$
$$link(buckling, local)$$
$$link(buckling, global)$$
$$link(limit\text{-}state, tearout)$$
$$link(stress\text{-}state, axial)$$
$$\vdots$$

$link(object\text{-}type, steel)$

$\vdots$

Root nodes of the subtrees are represented using the relation '*root*,' as shown below.

$root(limit\text{-}state)$
$root(stress\text{-}state)$
$root(object\text{-}type)$

Leaf nodes of subtrees are represented using the relation '*leafnode*,' as shown below.

$leafnode(yield)$
$leafnode(local)$
$leafnode(global)$
$leafnode(tearout)$

$\vdots$

The connection between leaf nodes and provisions is represented using the relation '*connected*,' as shown below.

$connected(yield, \text{"}AISC1.5.1.2.1\text{"})$
$connected(yield, \text{"}AISC1.5.1.4\text{"})$

$\vdots$

The set of all the facts given above is denoted by $\mathcal{F}$ and represents a portion of $\mathcal{M}(\mathcal{L}, \Delta)$.

Proper Axioms: Above, a set of relations is used for representing a portion of $\mathcal{M}(\mathcal{L}, \Delta)$ in terms of the set of facts $\mathcal{F}$. Using those relations it is possible to deductively define the remaining portion of the submodel. That portion of $\mathcal{M}(\mathcal{L}, \Delta)$ involves defining the subclassifiers, the relationship between single subclassifiers and single provisions, the relationship between single subclassifiers and multiple provisions, the relationship between multiple subclassifiers and single provisions, and the abundance, freedom, completeness, and uniqueness properties.

The relation '*subclassifier*' is used for defining the subclassifiers. The relation 'r_{11}' is used for defining the relationship between single subclassifiers and single provisions. The relationship r_{1n} is used for defining the relationship between single subclassifiers and multiple provisions. The relation 'r_{n1}' is used for defining the relationship between multiple subclassifiers and single provisions. The relation '*abundant*' is used for checking the abundance property of the provisions. The relation '*free*' is used for checking the freedom property of the subclassifiers. The relation '*complete*' is used for checking the completeness property of the provisions. And the relation '*unique*' is used for checking the uniqueness property of the provisions. These relations are deductively derived from $\mathcal{F}$ using the following proper axioms.

Axiom 1

$$identifier(X) \wedge identifier(head(Z)) \wedge link(X, head(Z)) \wedge (path(head(Z), tail(Z)) \vee$$
$$(leafnode(head(Z)) \wedge equal(tail(Z), \emptyset))) \Rightarrow path(X, Z)$$

If X is an identifier and $head(Z)$ is an identifier and X is the parent of $head(Z)$ in a classifier subtree and ($tail(Z)$ is a path from $head(Z)$ to a leaf node of the subtree or ($head(Z)$ is a leaf node of the subtree and $tail(Z)$ is null)), then Z is the path from X to a leaf node of the classifier subtree.

Axiom 2

$$root(head(Z)) \wedge path(head(Z), tail(Z)) \Rightarrow subclassifier(Z)$$

If $head(Z)$ is a root node identifier and $tail(Z)$ is the path from $head(Z)$ to a leaf node of a subtree, then Z is a subclassifier in the classifier subtree.

Axiom 3

$$last(C, M) \wedge connected(M, P) \Rightarrow r_{11}(C, P)$$

If the last element of the set C is M and M is connected to the provision P, then C is a subclassifier for P.

Axiom 4

$$provision(head(P)) \wedge r_{11}(X, head(P)) \wedge belongsto(head(P), S) \wedge r_{1n}(X, tail(P), S)$$
$$\Rightarrow r_{1n}(X, P, S)$$

If $head(P)$ is a provision and X is a subclassifier of the provision $head(P)$ and $head(P)$ is a member of the set of provisions S and X is a subclassifier for the set of provisions $tail(P)$, then X is a subclassifier for the set of provisions P.

Axiom 5

$$subclassifier(head(C)) \wedge r_{11}(head(C), P) \wedge belongsto(head(C), S) \wedge$$
$$r_{n1}(tail(C), P, head(head(C)), S) \Rightarrow r_{n1}(C, P, head(C), S)$$

If $head(C)$ is a subclassifier and $head(C)$ is associated with the provision P and $head(C)$ is an element of the set of subclassifiers S and the provision P is identified by the set of subclassifiers $tail(C)$, then the provision P is identified by the set subclassifiers C.

Axiom 6

$$root(B) \wedge subclassifier(X) \wedge subclassifier(Y) \wedge equal(B, head(X)) \wedge equal(B, head(Y)) \wedge$$
$$r_{11}(X, P) \wedge r_{11}(Y, P) \wedge \neg equal(X, Y) \Rightarrow abundant(P)$$

If B is the root node of a classifier subtree and X is a subclassifier and Y is a subclassifier and First element of X is equal to B and First element of Y is equal to B and X is a subclassifier for the provision P and Y is a subclassifier for the provision P and X is not equal to Y, then P is an abundant provision.

Axiom 7

$$\neg r_{11}(C, P) \Rightarrow free(C)$$

If C is not a subclassifier for a provision, then C is free.

Axiom 8 A provision is complete if it is not incomplete. The incompleteness of a provision can be expressed using the following wff.

$$root(head(X)) \wedge \neg(subclassifier(X) \wedge r_{11}(X, P))$$

That is, $head(X)$ is the root node of a classifier subtree and in that subtree there is a provision P which does not have any subclassifier. The completeness property is expressed in terms of the following axiom using the above wff.

$$\neg(root(head(X)) \wedge \neg(subclassifier(X) \wedge r_{11}(X, P))) \Rightarrow complete(P)$$

If the incompleteness property does not hold for a provision, then the provision is complete.

Axiom 9

$$complete(P) \wedge \neg abundant(P) \Rightarrow unique(P)$$

If the provision P is complete and the provision P is not abundant, then P is unique.

The above axioms and the set of facts $\mathcal{F}$ are denoted by Δ. The formal organization submodel is the pair $< \mathcal{L}, \Delta >$ and is denoted by $\mathcal{M}(\mathcal{L}, \Delta)$. The relations *belongsto*, *equal*, *intersect*, *last*, and *union* are problem independent and their axiomatic definitions are not given here.

4.2 Reasoning about the Formal Submodel

Processing (reasoning about) the formal organizational submodel $\mathcal{M}(\mathcal{L}, \Delta)$ involves defining theorems using $\mathcal{L}$ and proving the theorems based on Δ using the resolution theorem proving strategy. A theorem is a valid expression of $\mathcal{L}$. Several theorems of interest which correspond to the items discussed in Section 3.2 are presented in this section. Each item below provides an example for its corresponding item in Section 3.2.

4.2.1 Reasoning about Single Classifier Subtrees

A: Suppose we are given the completely defined subclassifier $\{limit-state, buckling, local\}$ and we are asked to determine its corresponding provisions. This can be formulated using the following theorem.

Theorem 1

$$root(limitstate) \wedge subclassifier(\{limit-state, buckling, local\}) \wedge pset(S) \wedge$$
$$r_{1n}(\{limit-state, buckling, local\}, P, S).$$

That is, is it true that $limit-state$ is a root node identifier and $\{limit-state, buckling, local\}$ is a subclassifier and S is the set of all the provisions and {limit-state,buckling,local} is the subclassifier for the set of provisions P. The result of proving this theorem is

$$P = \{AISC1.10.5, AISC1.9.1.2, AISC1.9.2.1\},$$

where P is the set of provisions corresponding to the given subclassifier.

Now, suppose we are given the incomplete subclassifier $\{limit-state, buckling, X\}$ where X denotes the unknown part of the subclassifier, and we are asked to determine its corresponding provisions. This is formulated using the following theorem.

Theorem 2

$$root(limit-state) \wedge subclassifier(\{limit-state, buckling, X\}) \wedge pset(S) \wedge$$
$$r_{1n}(\{limit-state, buckling, X\}, P, S).$$

That is, is it true that $limit-state$ is a root node identifier and $\{limit-state, buckling, X\}$ is a subclassifier and S is the set of all the provisions and {limit-state,buckling,X} is the subclassifier for the set of provisions P. The result of proving this theorem is

$$P = \{AISC1.10.5, AISC1.9.1.2, AISC1.9.2.1\}$$
$$X = \{local\}$$

and

$$P = \{AISC1.5.1.3.1, AISC1.5.1.3.2, AISC1.5.1.4.5\}$$
$$X = \{global\}$$

Two sets of results are obtained. In each set of results P is the set of applicable provisions, and X denotes the unspecified portion of the subclassifier which was obtained as a result of proving the theorem.

B: Suppose we are given the provision "AISC 1.5.1.2.1" and we are asked to determine its subclassifier in the object-type classifier subtree. This is accomplished by formulating the following theorem.

Theorem 3

$$provision(\text{“}AISC1.5.1.2.1\text{”}) \wedge root(object-type) \wedge cset(S, object-type) \wedge$$
$$r_{n1}(X, \text{“}AISC1.5.1.2.1\text{”}, object-type, S).$$

That is, is it true that "$AISC1.5.1.2.1$" is a provision and $object-type$ is a root node identifier of a subtree and the set of subclassifiers in that subtree is S and X is the set of subclassifiers which corresponds to P. The result of proving this theorem is

$$X = \{\{object-type, steel, i-shaped, hot-rolled, beam\}\}$$

where X is the set of subclassifiers in the object-type subtree for the provision "AISC 1.5.1.2.1".

C: To determine the abundant provisions the following theorem is used.

Theorem 4

$provision(P) \wedge abundant(P)$.

That is, is it true that P is a provision and P is abundant. This theorem cannot be proven true because no abundant provision exists in the organizational submodel of Figure 1.

To determine the non-abundant provisions the following theorem can be used.

Theorem 5

$provision(P) \wedge \neg abundant(P)$.

The result of proving the above theorem is

$P = AISC1.5.1.2.1$

$P = AISC1.5.1.4$

$P = AISC1.10.5$

$P = AISC1.9.1.2$

$P = AISC1.9.2.1$

$P = AISC1.5.1.4.5$

$P = AISC1.5.1.2.2$

$P = AISC1.5.1.3.1$

$P = AISC1.5.1.3.2$

That is, the above provisions are not abundant.

D: To identify the free subclassifiers in a specific classifier subtree (e.g., the object-type subtree) the following theorem is used.

Theorem 6

$root(object\text{-}type) \wedge equal(object\text{-}type, head(X)) \wedge subclassifier(X) \wedge free(X)$.

That is, is it true that *object-type* is a root node identifier and *object-type* is the first element of a set named X and X is a subclassifier and X is free. The result of proving the above theorem is

$X = \{object\text{-}type, steel, i\text{-}shaped, hot\text{-}rolled, beam\text{-}column\}$

To identify all the free subclassifiers in the organizational submodel it is sufficient to replace the object constant *object-type* with a variable in the above theorem:

$root(B) \wedge equal(B, head(X)) \wedge subclassifier(X) \wedge free(X)$

That is, is it true that B is a root node identifier and B is the first element of a set named X and X is a subclassifier and X is free. The result of proving the above theorem is

$X = \{object\text{-}type, steel, i\text{-}shaped, hot\text{-}rolled, beam\text{-}column\}$

$B = object\text{-}type$

and

$X = \{stress\text{-}state, axial, tension\}$

$B = stress\text{-}state$

That is, there are two free subclassifiers in the submodel: one is in the object-type subtree, and the other is in stress-state subtree.

4.2.2 Reasoning about Multiple Classifier Subtrees

E: To determine the applicable provisions for the classifier whose three subclassifiers are

$\{object-type, steel, i-shaped, hot-rolled, column\}$,
$\{limit-state, buckling, global\}$, and
$\{stress-state, axial, compression\}$

the following theorem is used.

Theorem 7

$pset(S) \wedge root(object-type) \wedge root(limit-state) \wedge root(stress-state) \wedge$
$subclassifier(\{object-type, steel, i-shaped, hot-rolled, column\}) \wedge$
$subclassifier(\{limit-state, buckling, global\}) \wedge$
$subclassifier(\{stress-state, axial, compression\}) \wedge$
$r_{1n}(\{object-type, steel, i-shaped, hot-rolled, column\}, P1, S) \wedge$
$r_{1n}(\{limit-state, buckling, global\}, P2, S) \wedge$
$r_{1n}(\{stress-state, axial, compression\}, P3, S) \wedge$
$intersect(P1, P2, P0) \wedge intersect(P0, P3, P).$

That is, is it true that S is the set of all the provisions and *object−type* is a root node identifier and *limit−state* is a root node identifier and *stress−state* is a root node identifier and $\{object-type, steel, i-shaped, hot-rolled, column\}$, $\{limit-state, buckling, global\}$, and $\{stress-state, axial, compression\}$ are three subclassifiers and $P1$ is a set of provisions associated with the subclassifier $\{object-type, steel, i-shaped, hot-rolled, column\}$ and $P2$ is a set of provisions associated with the subclassifier $\{stress-state, axial, compression\}$ and $P3$ is a set of provisions associated with the subclassifier $\{stress-state, axial, compression\}$ and the set intersection of $P1$ and $P2$ is $P0$ and the set intersection of $P3$ and $P0$ is P. The result of proving the above theorem is

$$P = \{AISC1.5.1.3.1, AISC1.5.1.3.2\},$$

where P is the set of provisions applicable to the design circumstance characterized by the classifier.

F: Suppose we are given the provision "AISC 1.5.1.2.2", and we are asked to determine its corresponding classifier. This can be done using the following theorem.

Theorem 8

$provision(\text{“}AISC1.5.1.2.2\text{”}) \wedge$
$root(limit-state) \wedge root(stress-state) \wedge root(object-type) \wedge$
$subclassifier(L) \wedge subclassifier(S) \wedge subclassifier(O) \wedge$
$equal(limit-state, head(L)) \wedge equal(stress-state, head(S)) \wedge$
$equal(object-type, head(O)) \wedge r_{11}(L, \text{“}AISC1.5.1.2.2\text{”}) \wedge$
$r_{11}(S, \text{“}AISC1.5.1.2.2\text{”}) \wedge r_{11}(O, \text{“}AISC1.5.1.2.2\text{”}).$

That is, is it true that "AISC 1.5.1.2.2" is a provision and $limit-state$ is a root node identifier and $stress-state$ is a root node identifier and $object-type$ is a root node identifier and $limit-state$ is the first element of a set named L and $stress-state$ is the first element of a set named S and $object-type$ is the first element of a set named O and L is a subclassifier for the provision "AISC 1.5.1.2.2" and S is a subclassifier for the provision "AISC 1.5.1.2.2" and O is a subclassifier for the provision "AISC 1.5.1.2.2". The result of proving the above theorem is

$$L = \{limit-state, tearout\}$$
$$S = \{stress-state, bending, shear\}$$
$$O = \{object-type, steel, i-shaped, hot-rolled, beam\},$$

where O, L, and S are the subclassifiers of the classifier corresponding to the provision "AISC 1.5.1.2.2".

G: Suppose we are given a classifier with one known and two unknown subclassifiers and we are asked to determine the unknown subclassifiers. Let $\{limit-state, buckling, local\}$ represent the known, and X and Y denote the unknown subclassifiers. The following theorem can be used to determine X and Y.

Theorem 9

$$pset(S) \wedge root(limit-state) \wedge root(object-type) \wedge root(stress-state) \wedge$$
$$subclassifier(\{limit-state, buckling, local\}) \wedge subclassifier(X) \wedge subclassifier(Y) \wedge$$
$$equal(object-type, X) \wedge equal(stress-state, Y) \wedge$$
$$r_{1n}(\{limit-state, buckling, local\}, P1, S) \wedge r_{1n}(X, P2, S) \wedge r_{1n}(Y, P3, S) \wedge$$
$$intersection(P1, P2, P0) \wedge intersection(P3, P0, P) \wedge \neg equal(P, \emptyset)$$

That is, is it true that S is the set of all the provisions and $limit-state$ is a root node identifier and $object-type$ is a root node identifier and $stress-state$ is a root node identifier and $\{limit-state, buckling, local\}$ is a subclassifier and X is a subclassifier and Y is a subclassifier and $object-type$ is the first element of X and $stress-state$ is the first element of Y and $\{limit-state, buckling, local\}$ is a subclassifier for the provision set $P1$ and X is a subclassifier for the provision set $P2$ and Y is a subclassifier for the provision set $P3$ and $P0$ is the set intersection of $P1$ and $P2$ and P is the set intersection of $P3$ and $P0$ and P is not an empty set.

The result of proving this theorem is

$$X = \{object-type, steel, i-shaped, hot-rolled, column\}$$
$$Y = \{stress-state, bending, shear\}$$

and

$$X = \{object-type, steel, i-shaped, hot-rolled, column\}$$
$$Y = \{stress-state, bending, compressive\}.$$

Note that any of the above answers satisfies the specified theorem.

H : The completeness of provisions can be examined using the following theorem.

Theorem 10

$provision(P) \wedge complete(P)$.

That is, is it true that P is a provision and P is complete. The result of proving this theorem is

$P = AISC1.5.1.2.1$
$P = AISC1.5.1.4$
$P = AISC1.10.5$
$P = AISC1.9.1.2$
$P = AISC1.9.2.1$
$P = AISC1.5.1.4.5$
$P = AISC1.5.1.2.2$
$P = AISC1.5.1.3.1$
$P = AISC1.5.1.3.2$

which implies that the above provisions are complete.

To identify the incomplete provisions the following theorem is used.

Theorem 11

$provision(P) \wedge \neg complete(P)$.

That is, is it true that P is a provision and P is not complete. This theorem cannot be proven true since no incomplete provision exists in the given submodel.

I: To check the uniqueness of the standard's provision the following theorem is used.

Theorem 12

$provision(P) \wedge unique(P)$.

That is, is it true that P is a provision and P is unique. The result of proving the above theorem is

$P = AISC1.5.1.2.1$
$P = AISC1.5.1.4$
$P = AISC1.10.5$
$P = AISC1.9.1.2$
$P = AISC1.9.2.1$
$P = AISC1.5.1.4.5$
$P = AISC1.5.1.2.2$
$P = AISC1.5.1.3.1$
$P = AISC1.5.1.3.2$

which implies the above provisions are unique.

To identify the non-unique provisions the following theorem is used.

Theorem 13

$$provision(P) \wedge \neg unique(P).$$

That is, is it true that P is a provision and P is not unique. This theorem cannot be proven to be true since all the provisions are unique.

The above theorems contain object constants and variables. In general, a theorem can be used in two ways: (1) given a wff containing only object constants, *check* the truth value of the wff; and (2) given a wff containing several variables, *generate* a value for each variable such that the wff becomes true. This dual nature of theorems makes it possible to use theorem proving as a means for *checking* a design and for *generating* values for the design variables.

In this section several theorems of interest were discussed. Realize that these are not the only provable theorems of Δ. In fact any valid expression of the language $\mathcal{L}$ can be considered as a theorem and can be proven true or false in Δ. Thus, a general language for representing and reasoning about the organization of a design standard can be developed using predicate logic, and the language can be used to reason about the organization of the standard via theorem proving. In this paper one such language for representing and reasoning about the organization of a portion of the AISC design specification was discussed.

5 A Logic Programming Implementation

In this section a *logic programming* implementation of Δ and its use are discussed. Since this implementation differs in some ways from Δ we refer to it as Δ'.

5.1 The Program

Δ' consists of a set of rules that model Axioms 1 through 4, a set of rules that are not found in Δ but are used to facilitate user interaction with the program, and a set of facts corresponding to those defined in Δ.

In the following program the symbol ',' denotes the conjunction ($\wedge$) operator, the symbol ';' denotes the disjunction ($\vee$) operator, the symbol ':-' denotes the reverse implication ($\Leftarrow$), and [] denotes the empty list. Also, $X.T$ denotes a list where X is the first element of the list ($X = head(X.T)$) and T is the remaining element of the list ($T = tail(X.T)$).

Axiom 1 is modeled using the following rule.

```
path(X,Y.L):- identifier(X), identifier(Y), link(X,Y),
              (path(Y,L); leafnode(Y), L = []).
```

Axiom 2 is modeled using the following rule.

```
subclassifier(X.T):-  root(X), path(X,T).
```

Axiom 3 is modeled using the following rule.

```
r11(C,P):-  last(C,M), connected(M,P).
```

Axiom 4 is modeled using the following two rules.

```
r1n(X,P1.T,S):-  provision(P1), r11(X,P1), belongsto(P1,S,true),
                 subtract(P1,S,W), r1n(X,T,W).
r1n(_,[],_).
```

The first rule above differs from Axiom 4 in that instead of using the set of all the provisions (S) in the relation $r1n$ in the body of the rule, a subset of S is used. This subset is denoted by W and is defined by subtracting the first element of P (which is denoted by $P1$ in the rule) from S. This change was necessary to facilitate the generation of the provision set $P1.T$ (i.e., the set P in Axiom 4) with distinct elements. The second rule is used to stop the recursion in the first rule.

Axiom 5 is modeled using the following rules.

```
rn1(C1.T,P,B,S):-  subclassifier(B.C1), r11(B.C1,P), belongsto(C1,S,true),
                   subtract(C1,S,W), rn1(T,P,B,W).
rn1([],_,_,_).
```

A modification similar to the previous rule is also made for this rule. That is, instead of using S in the relation $rn1$ in the body of the rule, W is used where W is obtained by subtracting $C1$ from S.

Axiom 6 is modeled using the following rule.

```
abundant(P):- root(B), subclassifier(B.X), subclassifier(B.Y),
              r11(B.X,P), r11(B.Y,P), not X = Y.
```

In this rule the infix operator "=" is used instead of the "*equal*" relation in the axiom.

Axiom 7 is modeled using the following rule.

```
free(C):- not r11(C,P).
```

Axiom 8 is modeled using the following rule.

```
complete(P):- not (root(B), not(subclassifier(B.X), r11(B.X,P))).
```

Axiom 9 is modeled using the following rule.

```
unique(P):- complete(P), not abundant(P).

pset(S):-  findall(X,provision(X),S).
```

This rule is used to generate the set of all the provisions. The relation *findall* has the following interpretation. $findall(X, G(X), P)$ implies P is the set of elements X such that the goal $G(X)$ is satisfied.

```
cset(S,B):-  findall(X,subclassifier(B.X),S).
```

This rule is used to generate the set of all the subclassifiers for the classifier subtree B.

To demonstrate how the formal organizational submodel handles abundance, incompleteness, and uniqueness properties, two modifications are made to the set of facts $\mathcal{F}$. The provision "$AISC$1.5.1.4" is connected to the leaf node '*local*' and the provision "$AISC$1.5.1.3.1" is disconnected from the leaf node '*global*' in the limit-state subtree. These modifications are reflected in the following facts.

```
identifier(limitstate).
identifier(stresstate).
identifier(yield).
identifier(buckling).
identifier(local).
identifier(hotrolled).
identifier(axial).
identifier(bending).
identifier(compression).
identifier(tension).
identifier(compressive).
provision("AISC 1.5.1.3.1").
provision("AISC 1.5.1.3.2").
provision("AISC 1.5.1.4").
provision("AISC 1.5.1.2.1").
provision("AISC 1.5.1.2.2").
provision("AISC 1.10.5").
provision("AISC 1.9.1.2").
provision("AISC 1.9.2.1").
provision("AISC 1.5.1.4.4").
root(limitstate).
root(stresstate).
root(objectype).
leafnode(yield).
leafnode(local).
leafnode(global).
leafnode(tearout).
leafnode(beam).
identifier(objectype).
identifier(ishaped).
identifier(steel).
identifier(tearout).
identifier(global).
identifier(beam).
identifier(column).
identifier(beamcolumn).
identifier(objectype).
identifier(shear).
identifier(tensile).
link(limitstate,yield).
link(limitstate,buckling).
link(limitstate,tearout).
link(buckling,local).
link(buckling,global).
link(yield,"AISC 1.5.1.2.1").
link(yield,"AISC 1.5.1.4").
link(local,"AISC 1.10.5").
link(compression,"AISC 1.5.1.3.2").
link(local,"AISC 1.9.1.2").
link(local,"AISC 1.9.2.1").
link(global,"AISC 1.5.1.4.5").
link(beam,"AISC 1.5.1.4.5").
link(beam,"AISC 1.9.1.2").
link(bending,tensile).
link(beam,"AISC 1.5.1.2.1").
link(column,"AISC 1.5.1.3.1").
```

```
leafnode(column).
leafnode(beamcolumn).
leafnode(axial).
leafnode(compression).
leafnode(shear).
leafnode(tensile).
leafnode(compressive).
link(global,"AISC 1.5.1.3.2").
link(tearout,"AISC 1.5.1.2.2").
link(hotrolled,beam).
link(hotrolled,column).
link(objectype,steel).
link(steel,ishaped).
link(ishaped,hotrolled).
link(beam,"AISC 1.5.1.2.2").
link(beam,"AISC 1.10.5").
link(beam,"AISC 1.5.1.4").
link(compressive,"AISC 1.5.1.4.5").
link(column,"AISC 1.5.1.3.2").
link(beam,"AISC 1.9.2.1").
link(shear,"AISC 1.5.1.4").
link(shear,"AISC 1.10.5").
link(shear,"AISC 1.5.1.2.2").
link(compressive,"AISC 1.9.1.2").
link(global,"AISC 1.5.1.3.1").
link(stresstate,axial).
link(stresstate,bending).
link(axial,compression).
link(tensile,"AISC 1.5.1.2.1").
link(compressive,"AISC 1.9.2.1").
link(compression,"AISC 1.5.1.3.1").
link(axial,tension).
link(bending,shear).
link(bending,compressive).
```

5.2 Some Illustrative Examples

In this section we illustrate the use of logic programming for proving the theorems expressed in Section 4.2. In logic programming the term *goal* replaces the term theorem.

Theorem 1

```
? root(limitstate), subclassifier([limitstate,buckling,local]),
  pset(S), rin([limitstate,buckling,local],P,S).
```

In response to the above goal the system returns

```
P = [AISC 1.5.1.4,AISC 1.10.5,AISC 1.9.1.2,AISC 1.9.2.1]
```

Theorem 2

```
? root(limitstate), subclassifier(limitstate.buckling.X),
  pset(S), rin(limitstate.buckling.X,P,S).
```

In response to the above goal the system returns

```
P = [AISC 1.5.1.4, AISC 1.10.5,AISC 1.9.1.2,AISC 1.9.2.1]
X = local

P = [AISC 1.5.1.3.2,AISC 1.5.1.4.5]
X = global
```

Theorem 3

```
? provision("AISC 1.5.1.2.1"), root(objectype), cset(S,objectype),
  rn1(C,"AISC 1.5.1.2.1",objectype,S).
```

In response to the above goal the system returns

```
C = [[objectype,steel,ishaped,hotrolled,beam]]
```

Theorem 4

```
? provision(P), abundant(P).
```

In response to the above goal the system returns

```
P = AISC 1.5.1.4
```

Theorem 5

```
? provision(P), not abundant(P).
```

In response to the above goal the system returns

```
P = AISC 1.5.1.3.1
P = AISC 1.5.1.3.2
P = AISC 1.5.1.2.1
P = AISC 1.5.1.2.2
P = AISC 1.10.5
P = AISC 1.9.1.2
P = AISC 1.9.2.1
P = AISC 1.5.1.4.5
```

Theorem 6

```
? root(objectype), subclassifier(objectype.X), free(X).
```

In response to the above goal the system returns

```
P = [objectype,steel,ishaped,hotrolled,beamcolumn]
```

```
? root(B), subclassifier(B.X), free(X).
```

In response to the above goal the system returns

```
P = [stresstate,axial,tension]
P = [objectype,steel,ishaped,hotrolled,beamcolumn]
```

Theorem 7

```
? pset(S), pset(S1), pset(S2),
  root(objectype), root(limitstate), root(stresstate),
  subclassifier([objectype,steel,ishaped,hotrolled,column]),
  subclassifier([limitstate,buckling,global]),
  subclassifier([stresstate,axial,compression]),
  r1n(C1,[objectype,steel,ishaped,hotrolled,column],S1),
  r1n(C2,[limitstate,buckling,global],S2),
  r1n(C3,[stresstate,axial,compression],S3),
  intersect(P1,P2,P0), intersect(P3,P0,P).
```

In response to the above goal the system returns

```
P = [AISC 1.5.1.3.2]
```

Theorem 8

```
? provision("AISC 1.10.5"),
  subclassifier(limitstate.L),
  subclassifier(objectype.O),
  subclassifier(stresstate.S),
  r11(limitstate.L,P),
  r11(objectype.O,P),
  r11(stresstate.S,P).
```

In response to the above goal the system returns

```
L = [limitstate,buckling,local]
S = [stresstate,bending,shear]
O = [objectype,steel,ishaped,hotrolled,beam]
```

Theorem 9

```
? pset(S), pset(S1), pset(S2),
  subclassifier([limitstate,buckling,local]),
  subclassifier(objectype.X),
  subclassifier(stresstate.Y),
  r1n([limitstate,buckling,local],P1,S1),
  r1n(objectype.X,P2,S2),
  r1n(stresstate.Y,P3,S3),
  intersection(P1,P2,P0),
  intersection(P3,P0,P),
  not (P = []).
```

In response to the above goal the system returns

```
X = [objectype,steel,ishaped,hotrolled,beam]
Y = [stresstate,bending,shear]

X = [objectype,steel,ishaped,hotrolled,beam]
Y = [stresstate,bending,compressive]

X = [objectype,steel,ishaped,hotrolled,beam]
Y = [stresstate,bending,tensile]
```

Theorem 10

```
? provision(P), complete(P).
```

In response to the above goal the system returns

```
P = AISC 1.5.1.3.2
P = AISC 1.5.1.4
P = AISC 1.5.1.2.2
P = AISC 1.10.5
P = AISC 1.9.1.2
P = AISC 1.9.2.1
P = AISC 1.5.1.4.5
```

Theorem 11

```
? provision(P), not complete(P).
```

In response to the above goal the system returns

```
P = AISC 1.5.1.3.1
P = AISC 1.5.1.2.1
```

Theorem 12

```
? provision(P), unique(P).
```

In response to the above goal the system returns

```
P = AISC 1.5.1.3.2
P = AISC 1.5.1.2.2
P = AISC 1.10.5
P = AISC 1.9.1.2
P = AISC 1.9.2.1
P = AISC 1.5.1.4.5
```

Theorem 13

```
? provision(P), not unique(P).
```

In response to the above goal the system returns

```
P = 1.5.1.3.1
P = 1.5.1.2.1
P = 1.5.1.4
```

6 Summary

The overall organization of a portion of the AISC design specification is formally modeled using predicate logic. This model called the formal organizational submodel of the AISC design specification, is composed of a formal language and a set of axioms. Reasoning about the organization of the AISC design specification is accomplished using the formal language to first formulate expressions called theorems and then to prove the theorems via the resolution theorem proving strategy. Theorems are used (1) to prove the uniqueness and completeness of the provisions and (2) to query the formal submodel.

A logic programming implementation of the submodel is presented and is used to prove the theorems discussed in Section 4.2.1.

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

[AISC80] *Manual of Steel Construction*, Eight Edition, American Institute of Steel Construction, 1980.

[Cronembold88] Cronembold, J.R., and Law, K.H. (1988). "Automated Processing of Design Standards," *Journal of Computing in Civil Engineering*, Volume 2, Number 3, Pages 255-273.

[Dym88] Dym, C.L., Henchey, R.P., Delis, E.A., and Gonick, S. (1988). "A Knowledge-Based System for Automated Architectural Code Checking," *Computer Aided Design*, Volume 20, Number 3, Pages 137-145.

[Elam88] Elam, S.L., and Lopez, L.A. (1988). "Knowledge Based Approach to Checking Designs for Conformance with Standards," Civil Engineering Studies, Civil Engineering Systems Laboratory, Research Series Number 9, University of Illinois, Urbana, Illinois.

[Fenves69] Fenves, S.J., Gaylord, E.H., and Goel, S.K. (1969). "Decision Tables Formulation of the 1969 AISC Specification," Civil Engineering Studies, Structural Research Series, Number 347, University of Illinois, Urbana, Illinois.

[Fenves87] Fenves, S.J., Wright, R.N., Stahl, F.I., and Reed, K.A. (1987). "Introduction to SASE: Standards Analysis, Synthesis, and Expression," National Bureau of Standards, NBSIR 87-3513.

[Garrett86] Garrett, J.H., and Fenves, S.J. (1986). "A Knowledge-Based Standards Processor for Structural Component Design," Report R-86-157, Civil Engineering Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

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

[Harris81] Harris, J.R., and Wright, R.N. (1981). "Organization of Building Standards: Systematic Techniques for Scope and Arrangement," National Bureau of Standards, Building Science Series 136.

[Jain89] Jain, D., Law, K.H., and Krawinkler, H. (1989). "On Processing Standards with Predicate Calculus," Proceedings of the Sixth Conference on Computing in Civil Engineering, Will, K. (Editor), Atlanta, Georgia.

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

[Lopez89] Lopez, L.A., Elam, S., and Reed, K. (1989). "Software Concept for Checking Engineering Designs for Conformance with Codes and Standards," *Engineering with Computers*, Volume 5, Number 2, Pages 63-78.

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

[Nyman73] Nyman, D.J., and Fenves, S.J. (1973). "An Organizational Model for Design Specification," Technical Report R73-4, Department of Civil Engineering, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

[Rasdorf87] Rasdorf, W.J., and Wang, T. (1987). "Generic Design Standards Processing in a Knowledge-Based Expert System Environment," Proceedings of the NSF Workshop on the Design Process, Waldron, M.B., (Editor), Ohio State University, Columbus, Ohio, Pages 267-291.

[Wos84] Wos, L., Overbeek, R., Lust, E., and Boyle, J. (1984). *Automated Reasoning Introduction and Applications*, Prentice-Hall Inc.

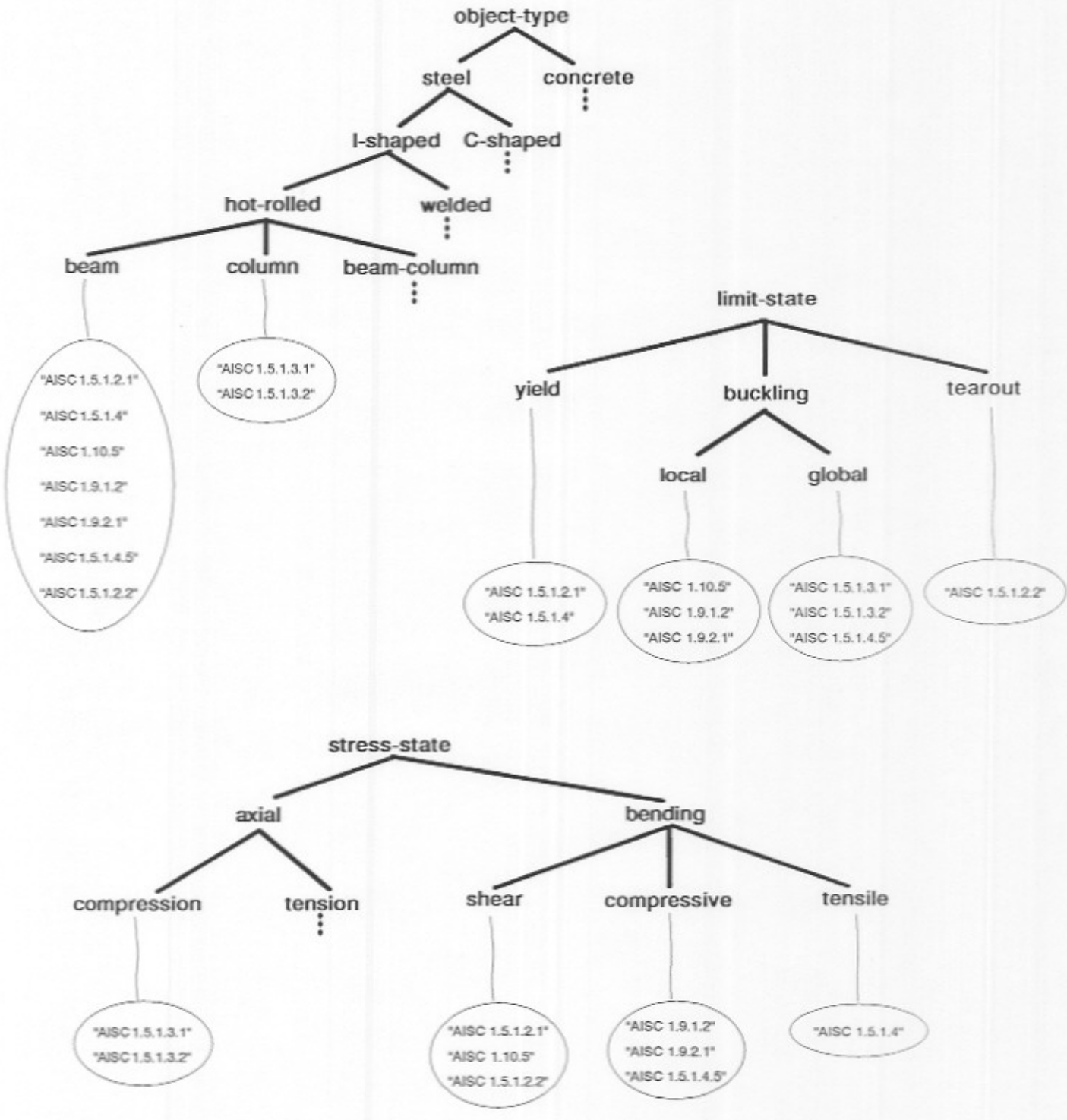
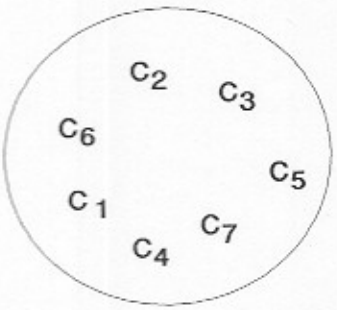
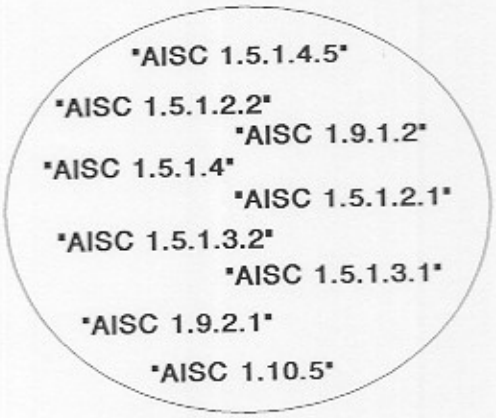


Figure 1: An Organizational Submodel for a Portion of the AISC Desing Specification

c:\grad\sivand\paper5\paper5f2.gem



C



P

$$C_1 = \{S_{11}, S_{12}, S_{13}\}$$
$$C_2 = \{S_{21}, S_{22}, S_{23}\}$$
$$C_3 = \{S_{31}, S_{32}, S_{33}\}$$
$$C_4 = \{S_{41}, S_{42}, S_{43}\}$$
$$C_5 = \{S_{51}, S_{52}, S_{53}\}$$
$$C_6 = \{S_{61}, S_{62}, S_{63}\}$$
$$C_7 = \{S_{71}, S_{72}, S_{73}\}$$

where

$$S_{11} = \{object-type, steel, i-shaped, hot-rolled, beam\}$$
$$S_{12} = \{limit-state, yield\}$$
$$S_{13} = \{stress-state, bending, shear\}$$

$$S_{21} = S_{11}$$
$$S_{22} = S_{12}$$
$$S_{23} = \{stress-state, bending, tensile\}$$

$$S_{31} = S_{11}$$
$$S_{32} = \{limit-state, buckling, local\}$$
$$S_{33} = S_{13}$$

$$S_{41} = S_{11}$$
$$S_{42} = S_{32}$$
$$S_{43} = \{stress-state, benidng, compressive\}$$

$$S_{51} = S_{11}$$
$$S_{52} = \{limit-state, buckling, global\}$$
$$S_{53} = S_{43}$$

$$S_{61} = S_{11}$$
$$S_{62} = \{limit-state, tearout\}$$
$$S_{63} = S_{13}$$

$$S_{71} = \{object-type, steel, i-shaped, hot-rolled, column\}$$
$$S_{72} = S_{52}$$
$$S_{73} = \{stress-state, axial, compression\}$$

Figure 2: Classifier (C) and Provision (P) Sets