Specification-Driven Design

by

Nayel Salah el-Shafei

Submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
at the
Massachusetts Institute of Technology
August 1990

Signature of Author [illegible]
Department of Civil Engineering
August 17, 1990

Certified by [illegible]
Fred Moavenzadeh
Thesis Co-Supervisor

Certified by [illegible]
Gerald Jay Sussman
Thesis Co-Supervisor

Accepted by ______________________
Ole S. Madsen
Chairman, Department Committee

Abstract

The thesis describes a system of computer programs that facilitates engineering design, by automating part of the process of elaborating specifications into detailed parametric designs. It uses classic problem-solving techniques, such as static evaluation, type checking, pattern matching, and constraint propagation. The system can help develop new designs; it can find violations of design rules in an existing design; and it can suggest modifications of an existing design to meet new requirements.

The system is a general tool that can be used in mechanical, structural, or electronic design. For example, the system can be used in the structural design of reinforced-concrete halls. Given the specifications for a hall, it first finds the most appropriate main structural system, whose parametric model is detailed and graphically displayed. The user may then impose additional constraints, which are propagated throughout the design. Attempting to satisfy the additional constraints may cause the entire design to be revised. Some of the constraints that could be applied, automatically, are those that represent standards or codes, such as ASTM, NFPA, or design rules for electrical circuits.

The system elaborates design plans by matching a description of the desired function of the design against a library of fragments, each indexed by the functions it can serve. A fragment is chosen and is inserted into the design. Constraints, propagated from the elaborated plan are then used to either further specify the fragment or to give reasons why it is not really applicable. An accepted fragment may itself require elaboration, thus this process is recursive. Design fragments may have structural as well as functional attributes. They may carry auxilliary information, not directly related to problem solving, such as graphical representations. In its ordinary operation, my system constructs, examines and discards many fragments. All of these fragments, along with their uses and limitations (discovered by rejection), are entered into a case library for later use as starting points for designs.

The work illuminates fundamental ideas for Design Description Languages. Each design fragment must be described in two ways: functional descriptions and structural descriptions. These must form a coherent combination. Specifications, of either a structural or functional kind, may be expressed in terms of constraints. Constraints are general in that they can be applied in synthesis, analysis, and for design-rule checking. Constraint propagation may yield contradictions that require resolution. Another language is provided for the user to express conflict-resolution rules that may be brought to bear on these situations.

The thesis offers the concept of harvesting for finessing the model of an evolving domain using the feedback on previous cases. The system uses credit assignment for discarding pieces of knowledge with low credibility. Using polymorphic generalization the system improves the causal model of the domain. Then, dimensional analysis and stepwise regression analysis are used to build/modify quantitative relationships in the model.

In the Name of God, The Most Beneficient, The Most Merciful

Acknowledgements

I would like to acknowledge the contributions to this work of the following people and other entities, and offer them my profound gratitude:

Fred Moavenzadeh, my advisor, who prodded me when I was stuck, patiently urged and pushed me toward the best I can do, and who has always been open-minded toward new methodologies and techniques.

Gerry Sussman, co-advisor, colleague, friend, who has encouraged me to venture into new horizons of computation, thought, and philosophy; and who is generally a jolly fellow to be around; he cares about his students. He originally inspired many of the good ideas in this dissertation.

Exposure to the insights of Marvin Minsky has influenced me greatly, and probably will continue to do so.
I am very much indebted to Prof. Jerome Connor for his continuous encouragement, Sandy Pentland and Tom Knight for their useful comments and criticism on various versions of this dissertation.

Other members of the MIT Artificial Intelligence Laboratory who have taken interest in my work and/or generally contributed to the comfortable and intellectually stimulating ambience there, including Rick Lathrop, Richard Doyle, Alan Yuille, and Patrick Winston. Pat Vargas, the troubleshooter of the MIT Center for Construction Research and Engineering, whose friendship and dedication were the source of comfort.

Numerous colleagues at Prime Computer, Inc.: Ed Wright who created a work environment where it is fun to do challenging jobs; Joe Slikowski for the comments and criticism, and enthusiasm for/on my work; Massood Zarrabian, Salahuddin Khan, Vincent Chaillou, and Pete Pritchard.

Skip Hommel, of New York State, whose insight and zeal for automation was a great motive for my work in design checking. Diane Rourke, also of NYS, was a great help in testing out the system in all possible ways.
Charles Kemper, of Phillips - Consumer Electronics, whose ideas about design synthesis were great inspiration for my work. Ralph Deakin of Rockwell - NAA provided useful comments.

Ron Wooldridge, of Syska & Hennessy, whose friendship and advice are valuable support.

Dr. Ahmed Moharram, Jr., of the Arab Consulting Engineers, for his trust and encouragement that initiated me into the world of engineering design.
Samir Rizk, of Freyssinet Internationale, for his encouragement and trust that initiated me into construction and project management.

Yehia el-Ezaby, my good friend.

My brother Mohammad, my sisters Taghreed and Lamya, their husbands Tareq and Nasser, whose encouragement is fueling me forward.
Aunt Nahed Sabbour, my other mother and my first teacher. Uncle Mamdouh el-Shafei, the good friend.

Salah eldin el-Shafei, and Ferial Sabbour, my parents, who have provided unflagging and unbounded support, encouragement, opportunity, resources, understanding, and just plain love for the last three decades. Thank you, Baba; Thank you, Mama.

Ali and Rhonda, my children and Mona, my wife, who have been my great inspiration beyond words.

Table of Contents

Table of Figures

"The natural sciences are concerned with how things are. ... Design, on the other hand, is concerned with how things ought to be, with devising artifacts to attain goals." Later on, he says, "... in large part, the proper study of mankind is the science of design, not only as the professional component of a technical education but as a core discipline for every librally educated man."

Herbert Simon, Sciences of the Artificial, 1969

Introduction

The past 150-200 yars have been years of incredible progress in technology. The main criterion on which such an evaluation is based is productivity or efficiency in the engineering process. By the turn of the century, the complex of machinery started to test the talents of the most capable engineers. Now, to modify an item such as a modern machine tool to suit the production of a new product may take the production staff of a factory six months, and to design a new machine may take a year or two. This time is somewhat longer than the ever-changing market for tool machines will allow. In those two years, new concepts can appear, and the new machine may be hopelessly obsolete at the very moment of its birth. The example demonstrates need for rationalizing the design process, and the automation of as much as possible of its tasks.

Though the need for improving and rationalizing the design process was felt even before World War II, progress was impeded by two factors: (1) the absence of a reliable means of representing abstract ideas; and (2) the widespread view that designing is a form of art, not a technical activity. The emergence of computers and data processing techniques helped to foster the adoption of systematic approaches to design (Pahl & Beitz 88). Computers helped to automate several stages of the design process, such as analysis and drafting. Other stages got scattered efforts of automation, such as process planning. The more critical stages, namely conceptual design and checking are still manually performed according to no practical guidelines. One reason for the lack of rationalizing, and consequently automation, of design synthesis up until now is the absence of any data processing tools capable of handling abstractions. New computational techniques introduced by artificial intelligence provides us with an arsenal of tools for knowledge representation.
More important than the automation of one stage or another, is the development of a unified automated systematic approach to the whole engineering lifecycle. Such development needs the establishment of a cognitive model of the human designer. Once the model is established, the computer implementation of it is a straightfor-

ward task. The development of a cognitive model of the designer is yet another area where artificial intelligence approaches can greatly help.

Modern systematic ideas were pioneered by Erkens in the 1920s. He emphasizes an algorithmic approach based on constant testing and evaluation, and also on the balancing of conflicting demands, a process that must be continued untill a network of ideas -the design- emerges (Erkens 28).
In systematic respects, design is the optimization of given objectives within partly conflicting constraints. Requirements change with time, so that a particular solution can only be optimized in a particular set of circumstances (Pahl & Beitz 88).

In sytematic design, great emphasis is placed on the establishment of a *function structure* and combination of the *sub-functions* in order to arrive at different designs. In addition to the function structure, the system needs to be defined as a blackbox with all inputs and outputs (system boundary) determined (Davis 83). Once the outer boundary and functionality of the system have been established, the most suitable process to fulfill every sub-function is found. Then comes the embodiment (layout) design that is form design of the component shapes and selection of materials, can be started.
Every design must meet both task-specific and also general constraints, e.g., standards, that need to be kept in mind at all design stages.

1.1. Hypothesis

This thesis casts the engineering design process mainly as the propagation of a body of specifications (constraints). The propagation of these constraints leads to a series of conflicts between various, occasionally-competing points of view. The resolution of these conflicts is the embodiment of the final design.
This view allows us to capture design as network of specifications, at some state of conflict resolution, that propagated a certain set of values for the design parameters. So, similar to traditional design approaches, we need to document our design both in calculations and illustrative graphics. To be useful for other designers, the design, moreover, has to include the alternatives that were available for the designer, and the dead-ends he tried out. This is the way a design case is added to the memory (experience) of a designer.

The improvement of design productivity of an engineer is attributed to the growth of his experience. Experience is perceived as a case library of all previous designs (cf. above) he went through. On tackling a new design, the access to a similar previous design case, with its specifications and final design, is a shortcut to a conflict-resolved, tried-out initial design. Having access to a similar previous design will shorten the synthesis process required to conform to the new set of specifications.

This constraint-propagation view of the engineering lifecycle is used as a base for a unified approach/framework of **Specification-driven Design** (SDD). This approach allows us to represent the entire engineering process, with its various engineering tasks from conceptual design to maintenance, in the same representation, namely as a *specification/constraint*. The framework is described in detail in chapter three.

The framework of design utilities is used as a system architecture for a programming environment. This environment is composed of a system/suite of computer programs. The system allows the user to declare, in a simple syntax, all the specifications that govern his designs.
In this environment, the framework was tested out in the various aspects of the design process in several real-life examples from various domains.

The experiments support the appropriateness of the framework, though they illuminate some current limitations, which are discussed in chapter (6).

1.2. Purpose & Objectives

The primary purpose of this thesis is to develop a framework for automating the design process, with all its stages. This has been accomplished by establishing a methodology reflecting the design processes followed by human designers. A system architecture that supports the methodology is elaborated in the chapter on Methodolgy. The system architectecture was developed into a system of computer programs. The computer system represents the general problem-solving capabilities of a novice engineer. Since the hypothesis is a general domain-independent one, the system can be used in various domains by initiating it with domain knowledge, previous designs, and/or cookbooks. Similar to human designers, the system should take advantage of previous designs to reach better designs with less computational resources.
The system perceives the design process as a propagation of bodies of constraints. The propagation of these constraints leads to a series of conflicts between various, occasionally-competing points of view. The resolution of these conflicts is the embodiment of the final design.

Developing the thesis has the following objectives:

- ° Developing (rationalizing) a systematic approach to the traditional design process that simulate the capabilities of human designers. To test the workability of the approach, it helps to be automateable.

- Automating as much of the algorithmic design stages, to focus the effort of the designer on critical decision-making, rather than mundane work. Automation must allow the propagation of constraints in all directions.
- Taking advantage of previous designs known by the system to reach better new designs with less computational resources.
- Modeling the decision process of designers, by enforcing bodies of standards and allowing user-defined rules for conflict resolution.

1.3. Scope of Work:

The thesis deals with conventional design rather than innovative design. A conventional design is an assembly of known components. A component is known if a parametric model can represent it functionally, geometrically and graphically. Therefore, a conventional design is a parametric model that assembles parametric components in predefined ways.

Innovative design, on the other hand, requires the synthesis of new components, whose categories (parametric models) were not known before. This synthesis needs more effort in elaborating the mainly-causal relationship between function and shape. While the functional modeling (through pre-defined causalities) is central to the thesis, the manipulation of the causal relationships is beyond the scope of this work. Therefore, innovative design is not handled by this work, although the representations used in the work allows the future extension of the scope to include innovative design.

1.4. Structure of the Thesis

Chapter two discusses related works in the intersecting areas, demonstrating how they relate to, compare to, or affect the work presented in this thesis.

Chapter three is an overview of the system that presents the methodology that was used to examine the hypothesis. This chapter also discusses the methodology-compatible representations used in the system for the functional, geometric and graphical aspects of design. Both the abstract and the detailed representation of functionality are described, along with the the control mechanism to switch between them. Chapter three describes the two modes of use, synthesis and checking. Finally, the chapter describes the case library and how it gets built.

Chapter four offers a detailed description of the computer programs that represent the system.

Chapter five offers some examples to demonstrate the capabilities of the system in various applications. The first example is in the area of conceptual structural design of Reinforced Concrete Exhibition Halls, and its relationship with dimensioning. The example shows how the general system was made to fit the structural design process through a customized knowledge representation. The second example is in the area of checking of design of hospital floor plans for compliance to a body of standards. Brief descriptions of other examples explore the applicability of the system to various aspects of the design process in different domains.

Chapter six is the summary and conclusion. It first describes the attributes of the system compared to traditional approaches. Then the thesis contribution is offered. The third section discusses the limitation of the system, and the future work needed for further development. Finally the conclusion describes how the implemented system fits the philosophy described in chapter one.
Chapter seven is a bibliography for related references.

It could be thought of as an acronym for "*Engineering Knowledge-integrated Design Utilities*", but the name has a different reason. Epic of Gilgamesh is the oldest known novel ever written by mankind (circa 3000 BC). It tells the story of Gilgamesh, the second king of Uruk after Noah's Diluge. Gilgamesh was saddened by the decision of God to reduce the life span of mankind from about 1000 years to 60 years; he thought that man is doomed because he still has the same ambitions in a much shorter life, hampered with the same needs. He was roaming around the city till he met with Enkidu, the man of the jungle. They exchanged stories. Enkidu's view was that man should try to achieve as much of his ambitions as he can, without paying attention to the general constraint on life span (Sandars 1972).
Enkidu's view is very similar to ours in approaching design automation problem. Computer capabilities are very limited when compared to human mind, but that should not dissuade us from trying to automate as much as we can of the design process.

2. Related Work

This chapter discusses the areas that intersect with the scope of the thesis. In every area, a brief discussion of the major works that are related to this thesis demonstrates what has been done and shows how previous work relate to this thesis. Finally the contribution of this thesis is discussed in the last section.

2.0.1. Design Automation & Constraint Models

So far, all Computer-Aided Design (CAD) systems in the market are just for computer-aided drafting. A user has to do his design somewhere else, then come to the CAD package to draw what he already designed, so he can communicate it with the others. The current CAD systems have no knowledge about any domain. The same system is used both for submarine design and for design of orthopedic implants. Design changes are every-day business for engineers. It is becoming more and more evident that engineers need systems that know about the artifacts they design. So that when an ECO (Engineering Change Order) is made, the system can propagate it through the whole design, or tell the user that the requested change is not feasible. An ECO is better perceived as a constraint. A *constraint* is a relationship that can be used in any direction. The capability to preserve the consistency of a design, after imposing an additional change to it, requires two things:

1. Knowledge of the relationships that govern the artifact and its domain, in the form of constraints. This knowledge is called a **Parametric Model (PM).**

2. Flexibility in using this knowledge, which is called **Constraint Satisfaction Process (CSP).**

In the following sections we will discuss Parametric Modeling, and the mechanisms for constraint management. Following that is a discussion of the cognitive models of design and finally, design methodologies.

2.1. Parametric Modeling:

In designing engineering artifacts, we have to deal with at least three major representations of the artifact:

Functional Representation that describes the relationships between the parameters that govern the functionality of the artifact. These parameters are mostly non-geometric parameters, such as maximum temperature, allowable stress, ... etc.

Geometric Representation that describes the geometric features, or shape, that achieve the above-mentioned functionality in terms of basic geometric entities. The function-shape relationship is quite a profound, domain-specific one.

Graphical Representation that describes the graphical implementation of the above-mentioned basic geometric entities, using a specific CAD system. Implementation is the design of the database in which the entities are saved.

To capture all three facets of knowledge about an artifact, a parametric model has to be polymorphic. Furthermore, to allow for portability of designs across various CAD systems, the graphical representation of a model needs to be separate from the functional and geometric ones. To allow for later innovative design, a separation is needed between the functional representation and the geometric one. Therefore, *a parametric model has to have the three separate representations.*

Since all CAD systems are stationed in the drafting area, their migratory path toward design automation, or Computer-Aided Engineering (CAE)[1], started at the lowest level of parametrization, i.e., graphical representation, with some geometric parametrization mixed in. They lack any notion of functional representation. Systems that started along the parametrization road include, Pro Engineer®, I.DEAS® and Mentor Graphics®. [2] This start affected their selection of the constraint management mechanisms.

2.1.1. Advantages of A Separate Functional Parametric modeler (FPM)

* The capability to add parametrics to existing drawings:
    - ° Makes use of the existing CAD drawings of current users, which is a feature no geometric parametric modeler has. To adapt the FPM to a drawing produced in a specific CAD format, all that is needed is to redefine the graphical representation of the basic geometric entities.
    - ° All other parametric modelers need to start a drawing from scratch, since they mix graphical representation with geometric representation.

1 Electronic CAD systems were ahead of Mechanical CAD systems in moving toward design automation. One reason is the simpler nature of electronic knowledge, and the finite set of basic components compared to mechanical design.

2 These registered trade marks are as follow: Pro Engineer is the name of the CAD system offered by Parametric Technologies Corp. (Waltham, MA). I.DEAS is the CAD system offered by Structural Dynamics Research Corp. (Colombus, OH). Mentor Graphics, Inc. is located in Beaverton, OR.

* It can be used for both synthesizing new designs and checking the compliance of existing designs to a body of standards.

* FPM allows hierarchical assembly through blackboxing:
   ° An FPM would allow the definition of an artifact as a blackbox. It also allows nesting of artifacts (blackboxes) to represent an assembly of artifacts. This capability is a vital one for assembly modeling, which is one of the frontiers of design automation & modeling. Conflicts in such assemblies are resolved on a functional basis, rather than a chronological sequence of construction, as it is the case in all Assembly modelers in the industry. Construction recency, nevertheless, could be used as a rule for conflict resolution.

2.1.2. Hurdles in the way of Functional Parametric Modeling

The major hurdle that dissuades researchers from tackling the functional aspect of modeling is **knowledge acquisition.** To be able to design a fastening joint, for example, it is necessary to encode at least the following sources of knowledge: the chapter on fastening in Mark's Mechanical Engineering Handbook, chapters on fastening in the ASTM, and other relevant standards. Despite the vast benefits to be reaped from this common knowledge once it is captured, the cost of acquiring it is quite prohibitive.

Solution:

No single developer should incur the cost of encoding general knowledge that is potentially useful for many users. Therefore, a format has to be established for the encoded knowledge, such that a user can either encode what he needs, or he can get it from someone else. To encourage a widespread knowldge encoding process, user-friendly knowledge editors need to be developed. In such an editor, knowledge can be encoded either schematically, graphically, or textually through a limited-vocabulary natural language parser. In this thesis, a nucleus of a knowledge editor is demonstrated.

So, while the industry is tackling design automation from bottom up, research is under way in several places to investigate functional (parametric) modeling: the most appropriate cognitive model to suit it, the relationship between shape and function, and design methodologies. These topics are the subjects of sections three through six of this chapter.

Some companies offer conceptual design systems, such as ICAD® (Cambridge, MA) and WISDOM® (Colombus, OH). The problem with these systems is the lack of tools for incremental editing of knowledge. To introduce one of these systems to a new domain requires several man-years worth of customization before it is ready.

The system presented in this thesis maintains the three separate representations. The system is meant to be a Functional Parametric Modeler (FPM) if not also a Geometric Parametric Modeler (GPM). While a FPM may partially work by itself, its ultimate use assumes the existence of a GPM to regenerate the representative geometry.

2.2. Constraint Satisfaction Process

The concept of constraints has a long, though intermittent, history in computer science, going back to Ivan Sutherland's SKETCHPAD in 1962. Amazingly, parametric modeling of an engineering artifact (especially bridges) was the motivation for the introduction of the concept. This section contains a brief description of a map of activities in constraint satisfaction processes.

The design of an artifact is represented by a set of functional and geometric relationships (constraints). One form of these relationships is algebraic equations. To illustrate the essence of constraints in parametric modeling, let us consider the formula for translating temperatures from Celsius to Fahrenheit, and vice versa.

$$F = 1.8 * C + 32$$

There are several mechanisms for computing F given C, or C given F.

The constraint satisfaction process is the mechanism responsible for adding a constraint(s), to a set of constraints such that the new set of constraints is consistent with each other.

In the following subsections we will discuss the major approaches in CSP from a parametric modeling point of view.

2.2.1. Algebraic Equation Solvers

If all the relationships that describe an artifact are algebraic equations, then a suitable system for maintaining consistency of equations is an algebraic solver. In the area of geometric parametrics, a considerable part of the relationships is made of algebraic equations; the rest is inequalities and discrete parameters.

In the Centigrade-to-Fahrenheit conversion example, both the conversion equation and an assignment equation (either F = <some value>, or C = <some value>) will be entered into the simultaneous equation solver, which will solve the two parameters in two equations.

Algebraic solvers range from sophisticated symbolic solvers, such as MACSYMA (Moses 75), to user-friendly, limited-capability, simultaneous equation solvers, such as Maple® and Mathematica®.[3]

3 Mathematica is a registered trade mark of Wolfram Research, Inc. (Champaign, IL).

Within the realm of algebraic constraints, CSP breaks down into three subproblems:

Satisfiability problem: to determine the existence of a solution to a CSP.

Counting problem: to determine the number of solutions.

Enumeration problem: to find all solutions.

The counting problem is a special case of the enumeration problem, and the satisfiability problem is a special case of the counting problem.

Since many *NP*-complete problems, such as graph-coloring, are CSPs, the satisfiability problem is seen to be *NP*-complete (Garey & Johnson, 79). Therefore, the satisfiability, counting, and enumeration problems are expected to be solved in no more than polynomial time. While various CSP methods add some additional cleverness beyond brute-force search, their worst case running time is still $O(m^n)$ (Mackworth 87).

Efforts, nevertheless, are exerted to improve the bounds for several CSPs. (Rivin & Zabih 89), for example, reformulate a CSP as an integer linear programming problem. The reformulated problem can be solved via polynomial multiplication. If the CSP has n variables whose domain size is m, and if the equivalent programming problem involves M equations, then the number of solutions can be determined in time $O(nm\, 2^{M-n})$.

<u>**Pros:**</u>

° Simultaneous equation solvers are thrifty in utilizing computational resources, compared to other approaches, e.g., constraint propagation. Of course, as more symbolic capabilities are added to these systems, as in MACSYMA, the required computational resources soar.

<u>**Cons:**</u>

° Algebraic solvers cannot handle inequalities or discrete parameters. Both types of constraints are widely used in engineering design. Indeed, this is one major limitation of all parametric modelers currently existing.

° Lack of object-orientedness to map specific physical parameters in the real world.
A solution to it is achieved through developing a data structure around every algebraic parameter for non-algebraic manipulations. This hybrid approach is needed for geometric parametric modeling to maintain sound semantic mapping.

° The solution of a system of simultaneous equations uses a sparse matrix, which does not render the approach portable to massively-parallel computers. This inadequacy is due to the large number of processes staying idle most of the time.

- Simultaneous equation solvers cannot tolerate either over-constrained sets of equations or under-constrained ones. **For example**, in the Centigrade-to-Fahrenheit example, if a user says that F = 68, then the system will calculate the value of 20 for C. If the same user, or another one, wants to impose a constraint that C = 25, the system will not accept the new constraint since it will make the system over-constrained.
  On the other hand, if we have a system of 4 unknowns in 3 equations, the system will not be able to process the parameters since the problem is under-constrained.
- At the introduction of any change of value to the model, the equation solver has to evaluate the whole system of simultaneous equations anew to calculate values for every variable. So, in a complex model with thousands of entities, we are dealing with at least a similar number of parameters. Every time a user changes a parameter, all the other parameters have to be recalculated. This is an expensive process regardless of the hardware platform.

These disadvantages are better understood when compared with other approaches, especially Constraint Propagation (CP). Therefore, a comparison is made in the next CP section.

2.2.2. Constraint Propagation

A constraint is a primitive operation such as adder, multiplier, gate, etc. Every operation consists of an operator and operands (called pins if we perceive the operator as an electronic device with pins extending out of it). More complex relationships are built in terms of these primitive constraints. calculation of values takes place by propagating values through these primitive operators. One of the successful representations of constraints is electronic schema.

To illustrate the essence of constraint propagation, let us, once again, consider the formula for translating temperatures from Celsius to Fahrenheit, and vice versa.

$$F = 1.8 * C + 32$$

There are several CP mechanisms for computing F given C, or C given F.

Figure (2.1) shows the schematic illustration of the constraint network that represent the conversion equation.

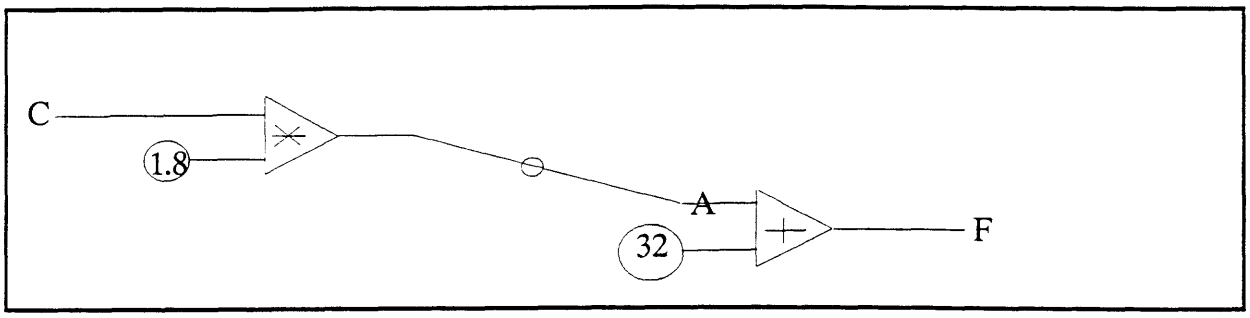


Figure (2.1): A constraint Network for C <-> F Conversion

When a user propagates a value, say 50 for F, it triggers the adder constraint that will find that it has enough known pins to be fired. Adder computes the value 18 for the upper left pin (A). Internal variable (pin) A is connected to the multiplier constraint which will get triggered by the newly-computed value of A. The firing of the multiplier constraint computes the value of 10 for C.

Constraint propagation mechanisms allows for relationships other than algebraic equations, such as:

- Inequalities, such as $X < Y$, and
- Discrete parameters, such as $Z \in \{1,4.9,11.3,25\}$

It is amazing that the concept of constraints was developed at first to be applied in geometric and graphical parametric modeling. The pioneer of the concept of constraints is Ivan Sutherland with his SKETCHPAD (Sutherland 63).

SKETCHPAD:

SKETCHPAD offered the capability of parametric modeling. It was capable of, for example, being told what a bridge looked like, along with the structural properties of its components. A load could be placed on the bridge and the bridge would deform to satisfy all the constraints on it, according to the theory of elasticity, etc.[4]
The constraint satisfaction process in SKETCHPAD consists of two mechanisms. A constraint propagation mechanism preceded by a planning mechanism that used a topological sort to order the constraints.

A *free variable* in SKETCHPAD is one "which has so few constraints applied to it that it can be re-evaluated to completely satisfy them." If the parameter is a scalar, it

4 It is amazing to see how many new concepts of computer science were introduced from the Civil Engineering domain. Other new concepts include the first virtual memory FORTRAN compiler (as part of the ICES project at MIT-Civil Engineering), and the concept of Finite Element by Zienkiwicz.

can have at most one constraint applied to it. That means that if the attached constraint is not satisfied, then the parameter can be changed without fear of invalidating some other constraint.
The ordering algorithm iteratively identifies all free parameters, eliminates them and the constraints they are attached to. This may free up variables for the next iteration. Iteration stops when there are no more free parameters; if there are no parameters left at all in the network then an ordering has been found, otherwise relaxation must be applied. When free variables (parameters) are eliminated, they are placed in a list in the order of elimination. This order is the ordering used in reverse for the propagation mechanism.

Considering SKETCHPAD's age, and its novelties, it is amazing how long it took the engineering community to start following the SKETCHPAD path to develop parametric modelers. These parametric modelers started to trickle down only over the last few years.

ThingLab:
Alan Borning developed the ThingLab (Borning 81) that generalized the capabilities of SKETCHPAD in a more streamlined (drastically different) implementation. It was a generalized simulation laboratory based on constraints. Users sketched a design and named the created part(s), and their behaviors. Based on this information, ThingLab performed a simulation. ThingLab had a powerful class system, based on its implementation language SmallTalk, for representing hierarchically constructed designs.

Similar to SKETCHPAD, ThingLab has two stages, planning, and propagation. Planning is quite different from that of SKETCHPAD. While planning in SKETCHPAD starts from the furthermost parameter in the constraint network and works its way inward in a global change-independent way, planning, in ThingLab, is a local process that starts from the change and considers all the localities around it before starting to outwardly sort parameters.

2.2.3. Local Propagation & Retraction
The work of Gerry Sussman & Guy Steele (Stallman & Sussman 77, Steele & Sussman 78, Steele 80) is the classical piece in the area of constraint propagation, which positioned the area as an established computational technique.
Their work depends on a technique of lazy evaluation called "*Local propagation.*" Operations are performed only when a sufficient number of operands have known values. For example, an ADDER operation needs two known operands, or pins, to allow the calculation of the third; this is done by travesing linked data structures representing the formula. When both *F* and *C* are unknown, the result becomes the formula itself.

Sussman-Stallman-Steele works keep track of the flow of information during propagation. Variables are either bound to a value or they are not. Each time an unbound variable is bound to a value the *premises*, those variables on which the new value is based, are recorded. If the value of a bound variable is to be changed, then it must be unbound first. This '*unbinding*' is called retraction. If a variable to be retracted is a premise of some other variable's value then that variable is retracted too so that its value can be recomputed. If a variable to be retracted is premised on some other variables, then these must be accounted for as well.

Every primitive constraint has hand-coded rules defining the propagation of values in all possible directions at that low level. Propagation in networks and macro constraints is handled by the system.
Values come either from the user, or from inside the system due to constraint triggering (i.e., rule firing). Each time a new value is computed for a variable cell, the rule used for computation is recorded.
Dependency information is recorded twice, once within the premise cell and once within the premised cell.
Steele achieves constraint satisfaction by first retracting old values, if any, then propagating the new value.

The major limitation of this approach is circular dependency.

The work described in this dissertation is based on their work with major differences in the areas of contradiction-triggering and handling, checking for circular dependencies, and schematic synthesis of networks.

Example:

I. Let us start with the constraint network in figure (1), that says:

$A * B = C$

$C * D = E$

where $A = 2,\ B = 3,\ E = 24.$

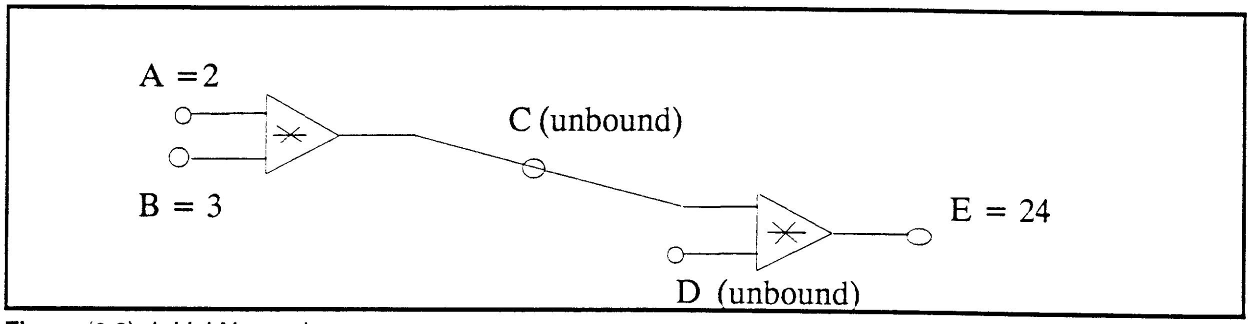


Figure (2.2): Initial Network.

II. The propagation mechanism will propagate (compute) the value of 6 for variable (C). Once C has a value, then the right multiplier gets triggered, since a sufficient number of its pins (2) got values. That will fire the rule that will compute the value of 4 for variable D, as figure (2).

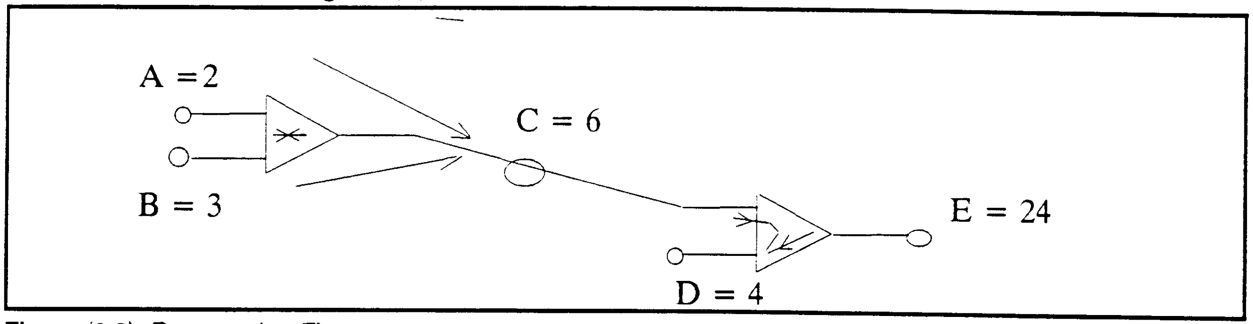


Figure (2.3): Propagation Flow.

III. Suppose that we want to change (A). A is a variable with no premises. Therefore it is retracted. With it also, all the variables that are premised on A will be retracted. So C is retracted. Since D is premised on C, then it also gets retracted, as in figure (3).

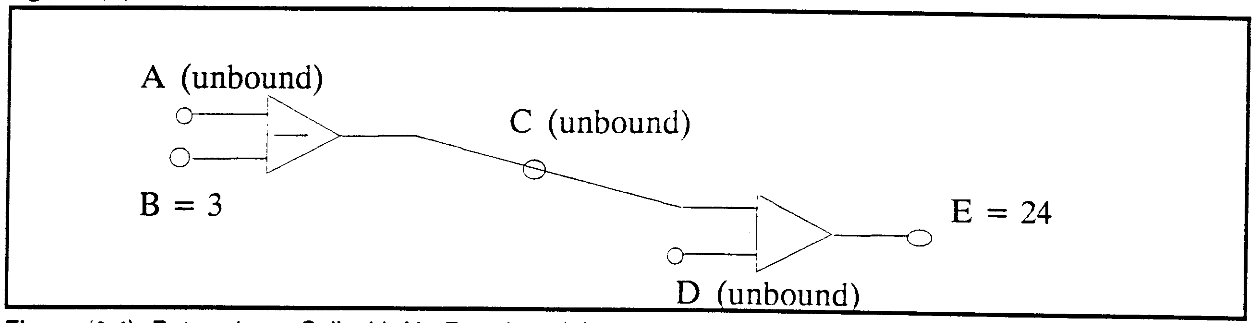


Figure (2.4): Retracting a Cell with No Premises (A)

IV. Now the system can propagate the new value of A. C gets recomputed based on A & B. Then D gets computed based on C and E, as in figure (4).

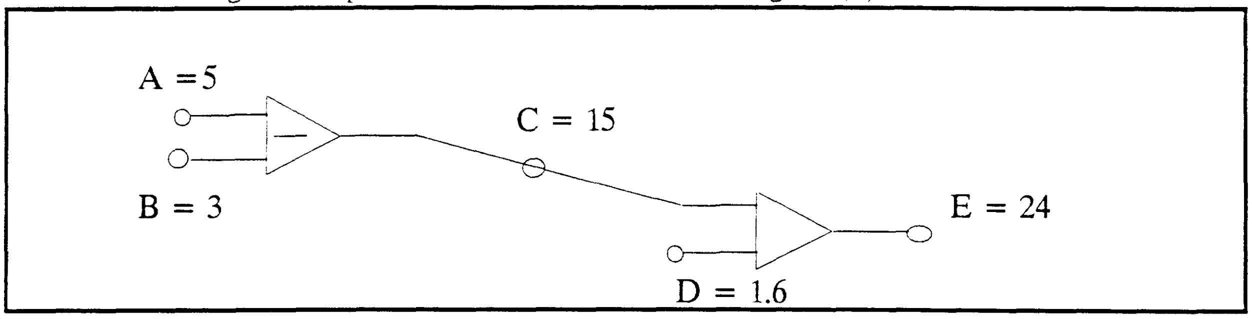


Figure (5): Propagation after Retraction A & B -> C = 15 C & E -> D = 1.6

V. Let us assume that the user wants to give a new value for a premised variable, such as C. C and one of its premises , say B, needs to be retracted. Then D will be also retracted since it is premised on C, as in figure (5).

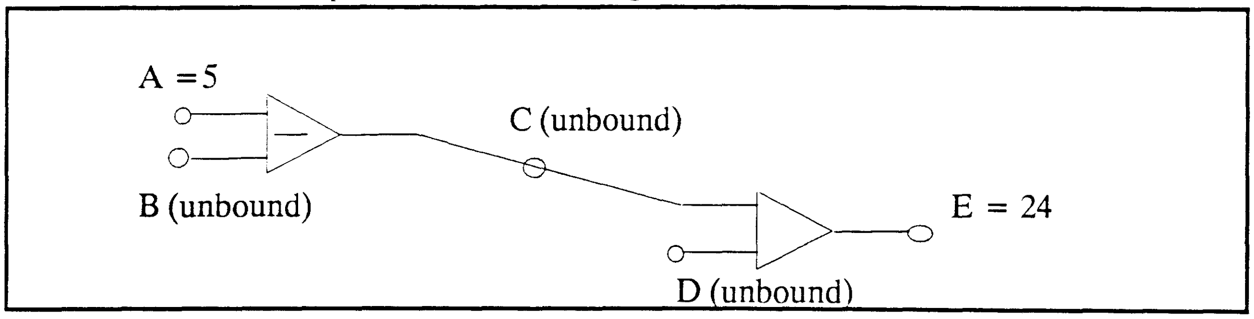


Figure (2.6): Retracting a Premised Cell (C)

VI. Then the new value of C is propagated through the network, as shown in figure (6).

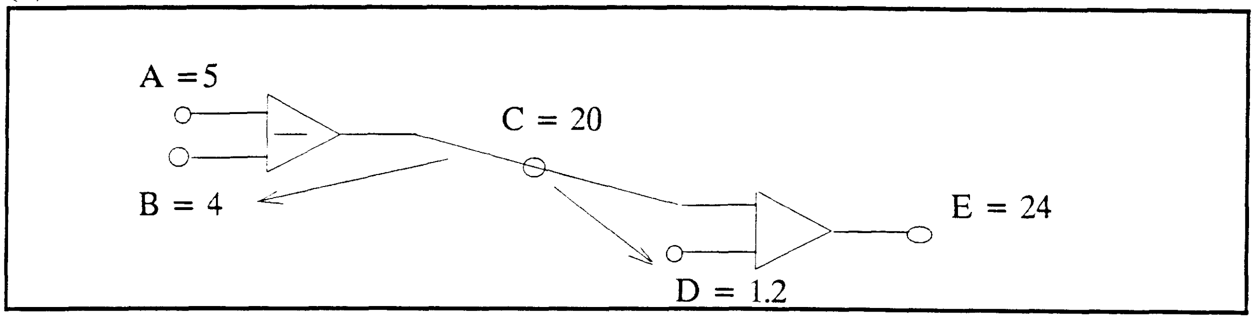


Figure (2.7): Propagation after Retraction with Premise Z LABEL FIG

Magritte:

Gosling offers a way out of the circular dependency problem, faced in local propagation, through the dependence on an algebraic system (Gosling 84). Gosling's system, Magritte, depends heavily upon the algebraic system in transforming any circular dependency subnetwork into a primitive constraint. Magritte checks for patterns of circular dependency before starting propogation. All detected loops undergo a **transformation** process to reduce them into a new single primitive constraint that replaces the loop. The transformation is made using a subsystem that takes the constraints that comprise the circular dependency and feeds them into a simultaneous equation solver that symbolically solves the system of equations several times, each time for one parameter. Each symbolic solution is used to specify the effect of applying the new synthesized constraint in one direction. This dependency on simultaneous equation solvers may deprive the system of the following:

* Flexibility of inequalities, which is needed in tolerancing, e.g., nominal diameter of the bolt (f) is less than the diameter of the nut (d), and

* Necessity of discrete parameters, e.g., calculated nominal diameter of a bolt has to equal one of the existing nominal diameters available in inventory.

However, a similar mechanism was developed for detecting patterns of circular dependencies, in macro constraints, and transforming them into newly-synthesized constraints. It is left to the user to decide whether to invoke that module or not for the reasons discussed above.

Another important reason not to constantly use the circularity-checking module, and to limit its application to within macro constraints, is the possible damaging of functional mapping of a macro constraint if the circularity stretches across two component macro-constraints. In such a case, the newly-synthesized constraint does not map into a specific functionality.

Other CSP-Related Work:

Several papers were published at the Eleventh International Joint Conference on Artificial Intelligence, August 1989, on search aspects of constraints (Freuder 89), (deKleer 89). Their major emphasis is on the set-theoritic issues of node consistency, arc-consistency and path-consistency. The three issues are polymorphisms of the problem of handling backtracking-ridden constraint networks. The solution, in short, lies in a *preprocessing phase* that should immediately follow the *network construction* phase and preceed the *propagation phase*.

Preprocessing may depend on local pattern matching (as we did in the system presented in this dessertation to detect circular dependencies).

Preprocessing of a constraint network is meant to reduce, or eliminate, futile backtracking, which wastes considerable computational resources. On the other hand, the preprocessing phase itself, consumes considerable resources. Faced with this trade-off, one tries to limit the use of preprocessing to the cases of hopelessly futile backtracking, as in circular dependency.

Inequalities are a very important tool in engineering modeling. They can be used in constraint propagation, but not with the algebraic solvers. (Grigorev & Vorobjov 88) offer an algorithm for solving systems of polynomial inequalities in subexponential time.

2.2.3.1 Constraint Propagation vs. Equation Solver

- ° At the introduction of any change of value to the model, the equation solver has to evaluate the whole system of simultaneous equations anew to calculate values for every variable. A constraint propagator, on the other hand, recalculates only the variables connected to the changed variable. So, in a complex model with tens of thousands of entities, we are dealing with at least a similar number of parameters. Everytime a user changes a parameter, all the other parameters have to be recalculated. This is an expensive process regardless of the hardware platform.
- ° Every constraint is represented as an object to which we can attach any properties (owner, conditions, ...) and methods. Each parameter in that constraint is also an object.
- ° The previous points hold in the case of sequential Von Neumann paradigm (single-processor) computers, super or otherwise. In moving into parallel architectures, the network nature of a constraint propagation system maps directly into the Hyper Cube architecture of almost all parallel computers. Compare that with the sparse matrix that gets generated by any equation solver, which leaves most processors idle most of the time.

2.2.3.2 Improvements in the Constraint Propagation System:

- ° Tolerates both over-constrained specification (contradictions), and under-constrained specification (through either null values or default ones).
- ° User-defined rules for contradiction handling.
- ° Accepts any data type for propagation (integers, reals, strings, ...)
- ° Graphical (iconic) schematic editor for construction of constraint networks.
- ° Detection of circular dependencies, and demonstrating it through the graphical editor.

2.2.4. Constraint Logic Programming (CLP):

Constraint Logic Programming (CLP) can be viewed from two perspectives: one related to mathematical logic and automated theorem proving, the other to the development of programming languages based on logic. Within the mathematical logic context, CLP represents an effort to establish a class of first-order theories which preserve the basic (simple) computational properties of Horn-clause logic. From the programming languages point of view, the purpose is to establish a class of logic programming languages in which the variables can have values in a diverse set of domains including trees, booleans, reals, rationals, list, etc. (Cohen 90).

Prolog is the most famous language in the family of logic programming languages. Prolog allows the programmer to write statements of predicate calculus in Horn-clause form. A Prolog statement is an implication whose antecedent is the conjuction of predicates and whose consequent is a single predicate form. A typical Prolog statement is:

```
arrange(cons(X,L),tree(T1,X,T2)) :-
          partition(L,X,L1,L2), arrange(L1,T1), arrange(L2,T2).
```

This may be interpreted declaratively as the statement

$$\forall X \forall L \forall T_1 \forall T_2 \forall L_1 \forall L_2\ (p(L,X,L_1,L_2)\ \Lambda a(L_1,T_1)\ \Lambda a(L_2,T_2)) = a(c(X,L),t(T_1,X,T_2))$$

However, Prolog provides an imperative interpretation, called *unification*. The term before the ":-" is considered to be a procedure declaration, and the terms to the right are statements of the procedure. Thus the statement above may be read, "*If you need to call procedure arrange, then its first argument must be a cons and its second a tree, and also you must execute three other procedure calls*". Moreover, there may be more than one "declaration" of a "procedure"; when a procedure must be executed, its various declarations must be chosen among "non-deterministically". Non-determinism is implemented using chronological backtracking to allow the use of stack operations.

The spread of Prolog is due to the development of a target language which can be efficiently used in executing programs. Warren's Abstract Machine (WAM) is the de facto lower-level code generated by current Prolog compilers (Warren 83). One of its advantages is that it replaces unification, whenever possible, by simpler sequences of tests and assignments. No well-accepted WAM model for CLP has been established yet.

2.2.4.1 Why Not Current Prolog for Parametric Modeling

Current Prolog has the following limitations:

- ° Chronological backtracking is less efficient than non-chronological, dependency-directed backtracking used in CP mechanisms.
- ° Prolog, like constraint propagation mechanisms, has a tree-parsing capability to be used in guiding deductive mechanisms. Unfortunately, dependency information is kept internal within Prolog, i.e., it is inaccessible by the users. This is a limitation of Prolog when compared with CP.
- ° Prolog has no notion of various levels of merit for a variable (such as Default, Parameter, Constant, and other user-defined merit levels).
- ° The CUT (!) command defies any notion of tractability, and makes dependency-directed backtracking quite awkward.

2.2.4.2 CLP: Rationale for Introducing Constraints to Prolog

Since *unification* could be viewed as a method of solving systems of equations involving trees, it could be generalized to new domains in which equations can be tested for solvability. The following example demonstrates Prolog's need for constraint capabilities: in Prolog, the equality

$$1 + X = 3$$

results in a failure, since the operation + is considered as an unevaluated function symbol, and the unification algorithm fails.

There are two major approaches to remedy the inference mechanism.
The first one is a *meta-level interpretation* allowing the description of interpreters for the languages (such as Prolog & Lisp) using the languages themselves. The interpreter assumes that the program rules are stored as unit clauses:

```
clause(Head,Body).
```

each corresponds to a rule:

```
Head :- Body.
```

where Head is a literal and body is a list of literals. Unit clauses are stored as:

```
clause(Head,[]).
```

The interpreter for Prolog is:

```
solve([]).                     % Special case for empty agenda
solve([Goal|RestGoal]):-       % In case of a full agenda:
      solve(Goal),             %    Solve the first item of the agenda, and
       solve(RestGoal).        %    Solve the remaining agenda.
solve(Goal):-                  % Special case for single-entry agenda:
      clause(Goal,Body),       % If there is a clause whose head is Goal, then
       solve(Body).            % Solve the body of clause.
```

In CLP languages, a rule is represented by:

```
clause(Head,Body,Constraints)
```

corresponding to a rule:

```
Head :- Body {Constraints}.
```

The modified procedure *solve* contains three parameters:

° 1. The list of goals to be processed,

° 2. The current set of constraints, and

° 3. The new set of constraints obtained by updating the previous set.

The meta-level interpreter for CLP, written in Prolog is:

```
solve([],C,C).
solve([Goal|RestGoal],Previous_C,New_C):-
      solve(Goal,Previous_C,Temp_C),
      solve(RestGoal,Temp_C,New_C).
solve(Goal,Previous_C,New_C):-
      clause(Goal,Body,Current_C),
      merge_constraints(Previous_C,Current_C,Temp_C),
      solve(Body,Temp_C,New_C).
```

The second approach is based on Automata theory, such as the abstract machine suggested by (Colmerauer 90) for Prolog III. The abstract machine resembles a push-down automaton whose stack is updated whenever a program rule is applied.

Two major proposals are emerging within the logic programming community to extend Prolog into a CLP. These approaches alleviate the above-mentioned limitations of Prolog, especially unification.
The first approach offered in (Jaffar & Lassez 87) provides a meta theory insuring that the basic theoretical propositions of logic programming remain applicable in the case of CLP, provided the domains being considered satisfy certain conditions.

The second approach, taken by Colmerauer (Colmerauer 90) in specifying Prolog III, is to describe the meaning of programs by establishing the relationship between rewriting rules representing the programs and the set of solutions obtained by applying these rules. In Prolog III the basic domain is that of infinite trees. A program rule consists syntactically of a pair: *rewriting rule* {*constraints*} in which: (1) a rewriting rule specifies that a term can be rewritten into a (possibly empty) sequence of terms, and (2) a constraint is a relation on terms.

In Colmerauer's Prolog III, backtracking is initiated when: (1) the current set of constraints is unsatisfiable, or (2) it is meaningless to apply an operation to the given operands.

2.3. Neuronal Network Models

Experience plays an important role in design problem-solving. Previous designs are utilized by engineers to synthesize a new design. In design problem solving, a goal is posted, then it is recursively broken down into subgoals. For every (sub-)goal we try to retrieve an analogous case from our memory that fosters or defeats that goal. The process of finding analogs, to build or reason about a structure, is called Case-Based Reasoning (CBR). A case retrieval system works according to a cognitive (neuronal) model. In the following sections a brief discussion is offered for both case-based reasoning and the various cognitive models.

2.3.1. Case-Based Reasoning

Case-Based Reasoning (CBR) is the category of techniques that takes advantage of experience in problem-solving. To be able to find analogies between the cue, in hand, and similar analogs in memory, every case retrieval mechanisms needs to have a cognitive model of memory organization (Schank 82). CBR is central to the system demonstrated in this dissertation, since it enables it to take advantage of its experience (previous design cases). Most of CBR mechanisms use search, as their case retrieval mechanism, over a space of indexed cases.

Indexing is the addition of keywords to a case before saving it in memory. You have to add as many indexes as possible to allow a search in whatever dimension. For example, a story could be used as an analog in numerous contexts, therefore, you have to be imaginative during *pre-processing* (indexing) of the case to anticipate all possible future uses.

2.3.1.1 Problems with Indexing

1. Preprocessing (indexing) of cases (before saving) is prohibitively expensive for any application with a real-life size.

2. Our memory (space of cases) is too vast to be searched. Even with an intricate system of indexes, the search is intractable. Heuristics were thought to alleviate this problem.

3. Making judgements very rapidly is a characteristic of the human brain to which man owes his survival. Experiments show that some decisions, such as

avoiding a speeding car, can take as low as 100 milliseconds (Feldman & Ballard 82). A neuron can be compared, in signal processing, to a transistor or a single bit of computer memory. Each neuron in a chain requires on the order of a millisecond to (electro- chemically) respond to its input (Kuffler & Nichols 76). That guided Feldman and Ballard to advance the *hundred-step rule*, which argues that the longest chain of neurons - including sensory and motor neurons - involves no more than 100 steps to tackle anyone of a wide range of decisions.

No significant amount of searching, or trial and error, can be supported within 100 single-bit steps.

Thus search, and consequently indexing, is precluded as an explanation for understanding natural language - where new words arrive every two hundred milliseconds or so - and also as as an explanation for object identification, situation assessment, emergency decision making, etc. (Waltz 89).

2.3.1.2 Non-indexing Approaches

Fortunately, there are proposed architectures that are consistent with these requirements for speed, such as the Society of Mind model (Minsky 86), and the connectionist models (Rumelhart & McClelland 86, Waltz & Feldman 88) . The emerging parallel hardware can implement such models efficiently (Hillis 85). These cognitive models are discussed in the next section.

A major plausible approach to case retrieval is through constraint satisfaction, as proposed by (Thagard & Holyoak 89) in their ARCS model. In that approach they identify three major kinds of constraints to govern how parts of two analogs can be placed in correspondence with each other (ordered according to importance):

1. **Semantic similarity:** Numerous psychological experiments indicate that retrieval of analogs by humans is very sensitive to the degree of semantic overlap between the target analog that provides retrieval cues and the source analog found in memory.

2. **Structural consistency:** There has to be a one-to-one mapping of (isomorphic) structures that perform analogous functions.

3. **Pragmatic centrality:** The purpose of analogies in problem solving is to help accomplish the goals of the problem. A case retrieval system, attuned to increase the retrieval of analogs relevant to goal accomplishment, would contribute more to problem-solving effectiveness than a retrieval system that lacked sensitivity to goals. Attunement is accomplished mainly by taking causality into consideration.

A major advantage of a constraint satisfaction approach is that it renders itself modelable to parallel architectures that drastically speed up retrievals.

Another approach for speeding up how knowledge processing in memory is to compile case retrievals into reflex arcs which has been the topic of two major classes of approaches.
The first approach is represented by a number of machine learning efforts such as the SOAR system that investigates chunking mechanisms for speeding up problem-solving (Laird, Rosenbloom and Newell 86).
The second class of connectionist network modelers are aiming at the goal of trainable systems for performing arbitrary input/output mappings in rapid time (Hinton, Rumelhart and Williams 86). Both methods require relatively large numbers of input variables for encoding goals, current situation, and context (together forming a kind of problem space), and involve little or no chaining. No items are preselected to serve as indexes (Waltz 89).

The major drawback of that parallel trend is that our high-level thinking is primarily serial. So, it is difficult to use introspection - AI's main source of knowledge over the years - to gain much information about such processes which are both rapid and largely inaccessible to conciousness.

2.3.2. Cognitive Models:

As design automation moves toward developing more general robust systems, it is becoming evident that the most powerful of the current generation of computers will not be able to compete with the brains of designers in tackling decision-making.
The most eminent reason for this disparity is that the human brain works in a massively-parallel mode, while our traditional computers are not. Therefore the need becomes more evident for massively-parallel approaches to the design problem (Hillis 85). The human brain is made up of several billions of neurons, most of which are made of a synapse (with capabilities similar to a transistor's) and a tiny memory attached to it (Kuffler & Nichols 76)[5]. The challenge is in understanding how our brains work as minds. In other words, what could be the organization of the vast number of neurons in the human brain to perform the subtle cognitive faculties that we have. Marvin Minsky offers his perception of such organization in his model, the *Society of Mind* (Minsky 86). In the Society of Mind, Minsky offers the *K-line* theory for how the same neuron can get involved in different organizations at different times to perform various tasks. Similar to Minsky's Society of Mind, Jean-Pierre Changeaux offers his *Neuronal Man* model (Channgeaux 85).

5 Interest in neurological models, among computer scientists, goes back to John von Neumann, refer to his lectures in (von Neumann 58).

The implementation of these theories was not possible till the development of the Connection Machine (Hillis), the first massively-parallel computer. The Connection Machine (CM) is a departure from the von Neumann paradigm of computation.[6]

The Connection Machine offers a large number, about 65000, of 1-bit arithmatic logic unit (ALU), each of which has attached to it a local memory of about 128 bits. To allow different communication paradigms between the processors, the CM uses the HyperCube communication paradigm. Unfortunately, most CM applications are geared towards machine vision. Nonetheless, there is some research in developing Finite Element analysis applications on it.

Vijay Saraswat (Saraswat 89) investigates a massively parallel implementation for constraint propagation, where every cell can be presented by a processor. Such strong semantics is very powerful and replaces the queue-based approach of the current implementations.

2.4. Truth Maintenance Systems

A Truth Maintenance System (TMS) is used with a continuously-changing knowledge base, to make sure that all the clauses inside the knowledge base are consistent with each other. TMS is used to assist an inference engine. The inference engine provides the TMS with queries about the consistency of its clauses. TMS, therefore utilize constraint propagation in reasoning.

Having such a generic capability is quite appealing especially in the ever changing world of parametric models of engineering artifacts. TMS has nodes to represent parameters, or literals. A node can have only one of three values (McAllester 78):

° *in* meaning that a fact is known to be true,

° *out* meaning that a fact is known to be false, and

° *unknown* meaning that a fact is not known to be true.

This is not sufficient for a day-to-day application in the world of engineering design. In design, what is *in* from a structural engineering point of view, could be *out* from an architectural point of view. Therefore TMS needs the following added capabilities:

° Larger, user-defined domain of values.

6 John Von Neumann perceived his first computer as a Central Processing Unit (CPU), and attached a large memory, and peripherals (von Neumann 58). In his design of the first generation of computers, he faced a vast gap in price between the material for manufacturing the ALU (such as the expensive vacuum tubes), and the material for manufacturing memories (such as the very cheap lead delay line). Therefore, he tried to maximize the utilization of the expensive CPU at the expense of the cheap memory. Now both the CPU and the memory are made of the same material, Silicon, but we still use the von Neumann paradigm. So we ended up with a paradigm that maximized the utilization of about 3% of the silicon, at the expense of the rest of that very same silicon.

° Conflict resolution that would allow for user-defined resolution rules.

The same limitation applies to Assumption-based TMS (ATMS). Johan deKleer, in (deKleer 89), compares ATMS and CSP, illiciting the analogies of trade-offs that need to be made on both sides and in the propositional encoding.

2.5. Design Methodology

The area of design automation is one of the major applications for computers. It started with application specific programs. Then as computers became more wide-spread, the areas of potential applications grew up. Developers discovered that there are a lot of techniques that work in more than one area, or in other words, that so many problems can be reduced, at some level of abstraction, to a finite set of general problems. This line of thought led to the tempting idea of the General Problem Solver (GPS), where a system armed with a library of general tools can tackle almost any problem. It took sometime to discover the severe limitation of such reductionism.

For the past 30 years, specific engineering artifacts, such as bridges (of fixed design) and speed transmission systems, were good applications of design automation. On the other hand, some other areas of design, such as architectural design, proved too formidable to be automated, because of the lack of understanding of how human designers work. The need for better understanding of the design protocol motivated many automation engineers to study the cognitive protocols that human designers follow.

In this section, we will first look at cognitive design protocols, then hierarchical modular approaches, then the cognitive architectures for the design process, and finally the area of rule-based approaches.

2.5.1. Design Protocols

A design protocol is a behavioral model that describes the way a designer creates his design. Such a model is transformed into an algorithm, which gets fleshed out into a computer program to examine the protocol and its resemblance to the human approach. A design protocol is very important in areas of design where it is difficult to weed through alternatives in order to formulate an objective function. Since architectural design is an example of such an area, most of the researchers in the area of design protocols are involved in architectural design. Architectural design is complicated by two problems:

° The high inter-connectivity of all parameters. Since the job is dividing one limited space to perform several functions under a heavy load of constraints. This problem is the topic of what architectural automation researchers call **space grammar.**

° Aesthetics, which is hard to quantify, is an important factor in the architectural design process.

Charles Eastman (Eastman 68) was the pioneer of this approach, followed by Omer Akin (Akin 88). They devised protocols (behavioral models) of architectural design to describe the different ways architects traverse the design space alternatives. These protocols could be used as domain-specific methodologies or search strategies.

For example, a top-level protocol is a methodology to synthesize the design specifications into a conceptual design. On the other hand, a detailed, or the same, protocol may tell you if a room is violating the specification (say, by being too small), and what are the **ranked** remedies to be examined.

Eastman's work accentuates the issue that the conflict resolution mechanism in our system should accept not only user-defined strength levels, but also domain-specific design methodologies.

2.5.2. Hierarchical (Modular) Approach

In some other engineering domains, a design is more of an assembly of components. Each component in itself is either an assembly or a basic component, whose design is well understood and straightforward. Design synthesis, in such cases, is more of a configuration problem where the major task is selcting the right components that satisfy the imposed constraints. A good example for such areas is mechanical design.

Separating Functionality from Geometry:

Karl Ulrich (Ulrich 88) breaks the (conceptual innovative) design problem into 2 stages: generating a schematic (functional) description and then generating a physical (geometrical) description.

The approach is similar to the one followed in our system, though our system does not tackle the area of innovative design in this version.

Top-down Hierarchy:

Ressler (Ressler 84) offers a top-down (hierarchical) methodology and an implementation for the design of operational amplifiers.

His approach has the following limitations:

° First, the user cannot introduce a change in the top-level specifications after the whole design is made.

° Second, the design criteria are defined a priori. They have to be user-defineable. Distilling alternatives, a priori, into a scalar value is too infringing on the designer's freedom.

Richard Doyle (Doyle 88) follows a similar approach in his system, Jack. Jack uses hierarchical planning in breaking a task/goal into blackboxes of sub-functionalities. Causality is the major factor in fleshing out the components of a blackbox.

<u>**Assembly Modeling:**</u>
Assembly modeling is tightly-coupled with parametric modeling which was discussed at the beginning of this chapter. Nevertheless, research in the causality and its impact on assemblies needs to be pointed out here for its relationship with the topic of hierachical modeling. David Gossard and his team have been researching parametric design of assemblies. For a good review of their work refer to (Kim & Gossard 89).

<u>**Virtual Construction:**</u>
Pentland's concept of virtual construction could have been covered in the section on algebraic solvers as well. Pentland offers an alternative approach to constraint propagation, called virtual construction, perception of non-rigid motion (Pentland & Williams 89). In his approach, an artifact is represented by a set of equations describing its vibration modes that gets solved by successive relaxation. Through this approach "*it is plausible to use sensory (tactile, visual) data to recover a nearly complete physical model of an object, so that we can, for instance, predict its response to impinging forces*". The approach is a special-purpose algebraic solver. Therefore, all the limitations of the use of algebraic solvers in design modeling, as described before, apply to it. It is not clear how the system handles design aspects that are not based on stress/strain relationships.

2.5.3. Rule-based Approach to Design:

Rule-based representation, because of its ease of use, is the most widespread paradigm for modeling the design process. Before discussing this approach, a description of a couple of stereotypical applications is given, then a general discussion of the approach ensues.

MaryLou Maher's HiRise system for the design of high rise buildings (of fixed rectangular plan) is an example of AI in structural design. The system is written in variants of the OPS-5 & FRL programming shells for production systems. This implementational issue restricts the representation used in the system to mainly production rules. Constraint propagation, or any other truth preserving mechanism, is needed for modeling the design process. Redeployment of the system using more powerful, flexible tools will help accentuate the capabilities and features of the HiRise system (Maher 88). Being a student of the same school of thought, Duvvuru Sriram follows a similar implementational approach in design with more explicit emphasis on the role of *blackboard mechanisms* (Sriram 85). Blackboard mechanisms allow for the representation of design through multiple *agents*, that represent various aspects of design (Nii et al. 88). The blackboard manager is responsible for both managing plans and resolving contradictions. *Agents*, in blackboard architectures, play a role that combines both advisors and part of the knowledge base, in our system.

J. Connor & F. Chehayeb (Connor & Chehayeb 87) tend to tackle serious applications like the main structural systems for medium size buildings. They are not reductionists when it comes to scoping the problem. Similar to Maher's HiRise, they have one limitation in their adherence to the rule-based approach, using the GEPSE system, which hinders them from developing a robust system. Nevertheless, the Gypsie system is comparable to the OPS-x family of rule-based programming shells, with emphasis on engineering applications. Addition of truth-preserving capabilities to Gypsie will make it a powerful engineering design tool.

Use of a rule-based model has its appeal in the following points:

- Simplicity: where engineers with minimal computer acquaintance can start using the approach;
- Inexpensiveness, in terms of computational resources. "*If-then*" rules are easy to implement, and do not take much of computational resources.

Nevertheless, the rule-based approach has the following limitations:

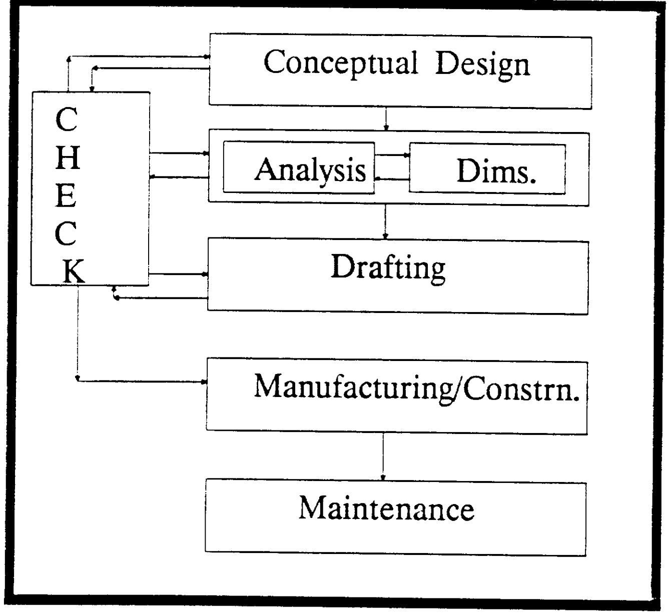
- It strictly enforces a one-way, top-down approach to design.
- Lack of truth-preserving capabilities.
- It is quite difficult to model all the cognitive faculties of the human designer in the form of "*if-then*" production rules.

Methodology

To present the methodology of the work described in this thesis, this chapter starts with background about the design process - then a section about the human approach to the process - then a framework, or system architecture, is developed for modeling the human approach. Once the architecture is established, the methodology describes how this architecture works. The last section deals with the representations that were required for the implementation of the methodology.

3.1. Background: The Design Process

The design process, as depicted in figure (3.1), starts with a conceptual design, then the engineer enters a loop of Analysis and Dimensioning until a fully-detailed version of the product is reached. To communicate this design to other engineers, the design gets drafted. Then, various entities perform a multitude of design checking procedures on the initial design. If a modification is needed, the design goes back to the Analysis/Dimensioning loop, and so on. After a satisfactory design is reached, it is released for Construction/Manufacturing.



To reach the ultimate design of one category of products, designers have to learn from all the problems that may arise during checking, manufacturing, and maintenance, so next design of a similar product will be a better design made in shorter time [Design for Manufacturability (Boothroyde 84)].

While some stages of the design process, such as Analysis and Drafting, got a great deal of automation, others got scattered efforts of automation, such as process planning, which is a part of the manufacturing block in figure (3.1). The more critical stages, namely conceptual design & checking, let alone the integration of the whole lifecycle, are still manually performed.

The development of an integrated environment that represent the real engineering lifecycle is still the hope of engineers in various domains. It is called Simultaneous

Engineering in the automotive industry and Concurrent Engineering in the aerospace industry. So far, it amounts to common database accessed by different engineering tasks.

3.2. Human Approach

If you ask a structural engineer to design a clinker silo, he will not start scribbling on a calculation sheet right away.

1. He will first ask himself if he ever designed a clinker silo (or anything with similar functionality) before.

2. If the answer is yes, he will grab the design case(s) from his experience and modify it to fit the new specifications. By doing that, the engineer has saved himself the effort of trying dead-ends, utilizing the short cuts that worked before.

3. If he has not met a similar case, as it is the case for a novice engineer, he will look for a design (cook) book on *How to Design a Clinker Silo*.
   If such a book exists, he will follow the steps in it. The steps in the book will certainly be more general and based on trial-and-error. The design will generally take longer.

4. If neither a similar experience, nor a specific design cookbook is available, there is no alternative but to synthesize a solution based on the specifications he has in the textbooks (knowledge bases) and the standards/specs of the project. The textbook will tell him about the functional description of a clinker silo and its major functional components (silo, handling unit, and temperature/ventilation control).

5. For every functional component of the project, the engineer will repeat steps 1 through 4, until he reaches a sub-component level that he knows how to design (through experience or from a catalog).

6. Unlike analysis, in design we are overwhelmed with alternatives to the final set of values. The weeding/selection is made by propagating the constraints imposed by the various bodies of standards.

7. By the end of the design of the clinker silo, the engineer will save the whole design case (including corrections & later revisions) in his experience and a generic version of it into the catalog (cookbook).

Next time our structural engineer is asked to design a grain silo (not only a clinker silo), he will be able to use, with modification, the silo unit of the clinker silo in the design of grain silo.

3.3. Description of the Design Utilities Framework

A novice engineer coming out of school may not know how to specifically design a lot of things, but he sure has the problem solving tools, and techniques that he can use to design almost anything once he has the functional specification.

As time goes on, he goes through more projects, his experience grows, and consequently, his performance improves.

Based on the human approach to design, as perceived in section (3.1), a **framework for design utilities** was designed, as illustrated in figure (3.2).

The framework consists of five major components, which are discussed in more detail in the next subsection:

1. A central core of capabilities that resemble the problem solving capabilities of a novice engineer.
2. A set of knowledge bases that represent a specific domain. The core uses the KBs for functional description of new objects and domains.

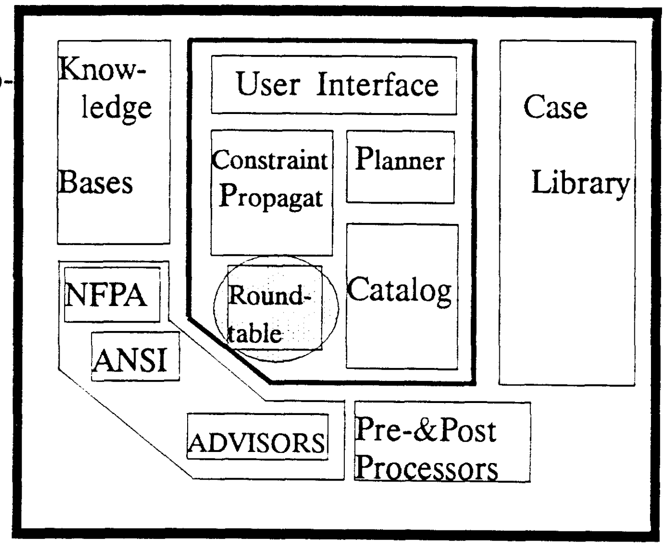


Figure (3-2): System Architecture.

3. A case library of complete designs. The core uses it as its experience. It retrieves previous designs from it, and it saves new designs in it.
4. Advisors (bodies of standards & specifications) to be enforced upon a design.
5. Pre-processors, like Finite Element Analysis, and post-processors, like numerical control programs for manufacturing the designed part.

All we need to provide to a designer, is the central core of the environment, denoted by the thick line in the middle of figure (3.2), since he can attach to it whatever knowledge bases, case libraries, advisors, and pre-/post-processors, he needs..

3.4. Harvesting Mechanism

Harvesting is a technique to improve the model of an evolving domain (in the knowledge base) using the feedback on previous cases. In other words, it uses case-based reasoning, and feedback, to improve its model-based reasoning.
The mechanism has three stages:

[I] The first stage is the broadcasting of copies of the current Knowledge Base (KB).

[II] The second stage is the modification of the local copy of the knowledge base according to the feedback on the current case and on the previous cases.

[III] The third stage is the harvesting of all the previously-identical copies of the knowledge base, after they have adapted themselves to the cases they encountered. The same knowledge modification process of stage [II], that was performed locally, is repeated on the old version of the KB using all the cases encountered by all the copies of the KB, with priority in stepwise regression to the relationships already proposed by the various copies.
Go to stage [I].

3.4.1. Knowledge Modification:

1. The system collects feedback on every design case after an appropriate period of time (6 - 12 months).

2. The system uses credit assignment for accumulating credit (positive and negative) along the reasoning path used in the design.

3. The system discards any piece of knowledge (causal constraint) that accumulates a certain negative credibility threshold.

4. Using generalization across the polymorphic[1] representation of knowledge, the system improves the causal model of the model, as described in (Pople 76).

1 By polymorphic representation means mainly causal and taxonomical representations. Other representations could be added such as nosological (in case of heat transfer, or plate stress transfer).

5. *Dimensional analysis* is used to cluster all parameters that may influence the design parameter under scrutiny in non-dimensional groups. Since a dimensionally-correct equation is more probable to be semantically more sound than an empirical formula (that does not preserve dimensional correctness).

6. The system builds an array of all the occurrances (in the current case and all other cases in the case library) of the parameters that constitute the dimensionless groups. Then calculate the value of every dimensionless group in every case.

7. The system uses the values of the dimensionless groups, formed in the previous step, as occurrances of parameters in *stepwise regression analysis* to build/modify a quantitative relationship with correlation factor above a user-set threshold. The new relationship is added to the model.

Because of this adaptive capability of the system, knowledge base gets changed by the end of the year to adapt itself to the feedback on the encountered cases. So, the previously-identical copies of knowledge base will grow different after being re-leased in different sites for a year, or so.

3.4.2. The Need for Harvest

1. To make sure that newly discovered relationships are not just perturbances due to local circumstances.

2. Making the new knowledge gained at one site available for the other sites.

3. The result of generalizing the knowledge gains of various sites exceeds the mere addition of the gains. Henry Kyburg makes a strong philosophical argument, supported by formal logic and theory of probability, that "*all acceptable generalizations are analytic*", and the importance of probability in supporting generalizations (Kyburg 83 - part IV). So, generalization allows us to broadcast a better model of the domain to all users.

3.5. Methodology

This section describes how the system works, and what we get out of it.

While the aforementioned system architecture should work in various design/engineering tasks, it had to be tested it in two major modes of operation: Design Synthesis and Design Checking.

3.5.1. (A). Design Synthesis

The user posts his specifications, which guides the system in selecting the most appropriate main structural system, whose parametric model is detailed and graphically displayed. The user may impose additional constraints, which are propagated through the design. The attempt to satisfy the additional constraints may cause the entire design to be revised. The user can query the system about the way (reasoning) any parameter got calculated.

3.5.1.1 Initial selection of fragments

Dimorphism of our world models has been argued from the early sixties. Marvin Minsky had the following statement in (Minsky 68):

> *"A man's model of the world has a distinctly bipartite structure: One part is concerned with matters of mechanical, geometrical, physical character, while the other is associated with things like goals, meanings, social matters, and the like."*

Whenever a functionality can be accomplished by more than one alternative, the system uses two representations for every one of the major alternatives to design the artifact:

Abstract representation: It is a mostly-functional representation that may include heuristic knowledge relevant to comparison of this alternative to other alternatives. So, instead of using the exhaustive brute force approach of fleshing out all alternatives to decide which one to be picked up, these heuristics consist of pre-compiled knowledge of experts to rapidly guide the system to a probably-right selection. Selection may turn, nevertheless, to be inaccurate due to the peculiarities of the specific case.

Detailed representation: It is used after the initial selection of an alternative is made. The detailed representation elaborate the processing of the various aspects of the object (such as analysis and dimensioning).

Without this dimorphism, the system would have spent large resources to exhastively elaborate the design using every alternative encountered, before making a selection. The abstract (functional) representation is used to compare alternatives (using a selector constraint network), to save the time of elaborating details of the various alternatives; such detailing is irrelevant at that level. After all, if details are of

importance in meeting the selection constraints, they will be considered at a later step.
Once the initial selection is made the abstract representations are discarded, and replaced by the detailed representations.

"In abstractionism we have not a picture that can be closely inspected and criticized, but a mirage that disappears when closer inspection is attempted."

(Geach 83)

The Selection Algorithm:

1. Build the initial selector constraint network that include the abstract representations of all alternatives.

2. Impose the user top-level (functionality) requirements upon the initial selector network.

3. Propagate the functionality constraints to make a selection (as detailed in the next chapter) of one alternative to satisfy the functionality, and save its name in the variable *SELECTED-SYSTEM*.[2]

4. Sever the link between the *SELECTED-SYSTEM* variable and the selector network. to avoid considering (propagation through) them at every small change.

5. Connect the *SELECTED-SYSTEM* variable to a ONE-OF or-tree of the names of the various alternatives. Since *SELECTED-SYSTEM* variable has a value, it will be satisfying the one-of constraint.

6. Build the detailed-design selector constraint network that include the detailed representations of all alternatives.

7. Impose the user top-level (functionality) requirements upon the detailed-design selector network.

8. Connect the *SELECTED-SYSTEM* variable to the detailed-design selector network, which will detail the design of the selected alternative.

2 Every abstract representation has a pointer to the detailed representation which is used for the rest of the design process.

9. Propagate all constraints to assure the fullfilment of all functionality constraints. If functionality was not met, the detailed-design selector will reject the currently-selected alternative, and pick up another alternative that satisfies the functionality. The newly-selected alternative will be assigned to *SELECTED-SYSTEM* variable.

The Design Synthesis Algorithm:

1. The user defines a functionality, for which a design will be synthesized. The user also determines any specifications the new design needs to meet.

2. The system searches the catalog for a structure (macro constraint) that has the given functionality as a name (using symbolp & macro-constraint-type-p) or as a synonym (through a hash table of synonyms)[3]. The catalog has a categorical description for every artifact the system knows, whether it is a top-level artifact or a component.

3A. If a catalog item is found, the system reads the list-of-instantiations that has pointers to all the instantiations of that item in the case library.
The system notifies the user with the names of the cases that possibly are analogous to the design problem at hand.
The system searches for the first case that includes the macro-constraint (pointed at by one of the pointers in the list) as a top-level design rather than a component of a larger design.

3A1. If no top-level instantiation was found, the system creates an instantiation of the catalog item, without the use of the case library.

3A2. If a top-level instantiation was found, the system makes a copy of the whole case (the macro constraint, and the the final status of the queues). For every parameter, whose value in the new problem is known, the old value is retracted.
The constraints that were found to be needed after the deployment of the old design (and hence were listed as an attribute of the old case) are instantiated and imposed upon the new copy of the macro constraint.

3 The hash table is not implemented yet. Instead, the system currently uses the synonym entry in the case data structure in the case library.

3B. If no items in the catalog meet the required functionality, use the knowledge bases (such as reference books, and cookbooks) to synthesize a catalog item (macro- constraint-type, or several macros connected by a detailed-design selector network) using the afore-mentioned *selection algorithm*.
Make an instantiation of the newly synthesized catalog item.

4. The known values of the parameters (of the new design problem) are propagated.

5. For every article of the specifications, if the object (being designed) passes the filter of that article, that means that the article is relevant. Every relevant article of specs is unified (i.e., a copy of its constraint is created after replacing the generic variables by the specific names of the parts to be constrained). The specific (unified) constraint is imposed upon the macro constraint of the object.

6. Resolve any contradictions in the new macro constraint.

7. Take any additional constraints from the user, propagate them, and resolve any contradictions. The macro constraint, by now, represents the (stable) new design.

8. The macro-constraint, the final status of queues, and a copy of the constraint network representing the specifications imposed upon the design are saved in a new design case in the case library.

9. Any further modifications in the design, during checking or manufacturing, are added to the design as additional constraints. Now comes the issue of version control.

9A. if the user wants the additional constraints in the same revision, they are added to the same macro constraint.

9B. If the user considers the modifications as a revision to the original design, they should be reflected in a new version. A copy of the whole case is made and the modification constraints are added to it.

Whether it is a new revision or not, the additional constraints get propagated.

10. Modifications suggested after deployment (manufacturing) of the design, are formulated as generic (i.e., yet to be unified) constraints whose listing is saved in an attribute of the case.

3.5.2. (B). Design Checking:

The user can post any body of standards, as specifications, and then enforce them upon a CAD file of a design. The system will flag out all the violations in the design.

The algorithm:

1. The design drawings are translated into a constraint representation using an intelligent scanning capability, CAD package-specific object declarations, and/or a translator.

2. The user specifies the bodies of standards against which the design is checked for compliance. These standards are in a specification format.
   The user also specifies whether the goal is just flagging the violations, or going the extra step and trying to find a solution, i.e., a resolved (violation-free) constraint network.

3A. If the purpose of checking the compliance of a design is just flagging out the violations, the system takes every object in the design and checks its compliance with all the relevant articles of the imposed standards. Every violation is graphically flagged by a numbered icon on a separate sheet (reserved for violations) of the graphical representation. The number inside the icon refers to a detailed description of the violation to be found in a textual violation report.

3B. If the purpose of checking the compliance of a design is also modifying the design to conform to the standards, then all relevant articles of the standards are determined for each object in the design. For every relevant article, the generic constraint is unified (instantiated) to the specific name of the object. The instantiation is hooked up to the macro constraint that represents the object.
Then follow steps (6 through 8) of the design synthesis algorithm.

3.5.3. (C). Other Uses of the Architecture:

3. **Hybridizing two designs into a new one:** For example, we can use the main structural system of design (A), with the Heating, Ventilation, Air Conditioning (HVAC) of design (B) to create design (C).

4. You can **make use of the case library (experience) of another colleague/designer**.

5. With **cost estimate** function, automatically translated into a constraint, the user can use the system as an engineering spreadsheet to evaluate cost/specification relationship.

3.6. Knowledge Representation: Constraint Propagation

3.6.1. The Need for a Parametric Design

Engineering design is a dynamic process that seldom follows a specific sequence. Parametric design is the kind of design that allows for any additional constraint to be imposed, anytime during the engineering process, upon any variable. That constraint is expected to trigger a change in that first variable. This change will propagate all over the system, until the whole design is compatible with the newly-introduced constraint.

3.6.2. Why Constraint Propagation Representation

1. Local propagation vs. simultaneous equations solver:
In a Simultaneous equations solver, if a change is made to a variable, say the diameter of a door knob in the 45th floor of a skyscraper, to stay parameteric the system has to redesign the whole skyscraper in all aspects, structurally, mechanically, electrically, etc. While in Constraint propagation, the system only needs to recalculate the parameters that are really affected by change.

2. The need to post any kind of relationship between parameters, such as equations, inequalities, discrete variables, etc., is only doable in constraint format.

3. The need to accomodate under-constrained, and over-constrainted systems, which cannot be tolerated by any algebraic system.

4. The need for a reasoning capability in the system, where every parameter knows the following:

° How it got its value, if it has one,

° How to get a value, if it does not have one, and

° The other variables dependent on it.

Of course, constraint propagation has its own limitations that will be discussed in the Limitations sections.

3.6.3. Representing Non-Algorithmic Knowledge:

Constraint propagation is a suitable representation for most algorithmic knowledge of design. One major reason for the elegance of constraint propagation as a representation is its adherence to mathematical logic. This elegance is achieved by keeping primitive constraints to a minimum. These minimum primitive constraints are the basic mathematical and logical operations. Any furthur capabilities could be built as macro constraints on top of the primitive ones.

Non-algorithmic knowledge, such as the causal relationships, may be computationally taxing if we want to preserve the minimalistic elegance of mathematical logic. Nevertheless, by hardwiring few logically-redundant constraints, e.g., IF, as primitive constraints instead of defining them as a well-founded formula of basic logic operators (in a constraint network), we can avoid the unneeded network parsing.

if x then y, i.e., $x \rightarrow y$,

which could rephrased (in primitive constraints) as:

$\sim x \mid y$

Taxonomical relationships, e.g. A-KIND-OF, on the other hand, are more subtle to handle. To handle them, we not only need to devise new constraints, but also need to devise an inheritance mechanism.[4]

we may represent attributes in the following fashion:

if X then Y, $x \rightarrow y$

where y is an attribute statement of x

Then, if we say that A ako X

then the system invokes an instantiation of Y in terms of A.

4 The ultimate form of constraint propagation should be implemented in an object-oriented system, such as the yet unreleased CLOS, or it should be implemented on top of an object-oriented database (Keene 89).

A direct consequence of such a capability, is *learning* some generalization rules (Michalski 84) to be embedded in the propagation mechanism, which is implemented through the procedure **run!** in the computer program.

3.6.4. Knowledge Representation:

The following paragraphs describe the representations that was designed to allow the implementation of the afore-mentioned methodology. Almost every object specified in (Steele 80) got modified heavily for handling procedures and methods.

Cell:

Every argument (parameter, variable, or constant) or constraint pin is represented by a cell. Every cell has a repository where it keeps its value. In addition to value, every cell has both merit and strength.

Repository:

It is a data structure that describes the place where a cell value is to be stored. Repository itself does not hold a value. If two cells are equated, then their repositories are merged.

Prim:

The structure PRIM is used for the graphical representation of a constraint. The structure is saved in the association list of the GRAPHICS slot of the CONSTRAINT structure under the locator GRAPHICS.

Queue:

Is a stack of operations yet to be performed. The system has several queues for control of propagation. The whole propagation mechanism is based on performing various operations on the entries of the various queues. Types of queues:
CONTRA-QUEUE, VANILLA-QUEUE, NOGOOD-QUEUE, PUNT-QUEUE.
An additional queue that was added for handling geometric conflicts is GEOMETRIC-CONTRA-QUEUE. This queue is not to be processed while geometric editing is going on. For further discussion about geometric constraints, see p. 57

Constraint-type:

It is the description of a category of constraints, such as ADDER. The description includes the various rules that represent the behavior of the constraint type in the various direction, and for different uses, such as adding a new value, forgeting a value or making a nogood list.

Constraint:
A relationship between parameters. It could be an equation, inequality, qualitative condition, etc. It is an instantiation of the category of constraint-type.

Macro-constraint:
A constraint network wrapped in a blackbox, so the user can deal only with a limited set of parameters (pins) governing the behavior of the blackbox. A macro constraint could be used to represent a complex (part of a) mechanism.

Macro-constraint-type:
The stereotypical description of a macro constraint. So it can be used for the generic description of a complex (part of a) mechanism.

Vector:
A macro constraint that include several copies of a smaller macro constraint attached to each other in the same way. Vector is good for representing an artifact with repetitive components, such as a multi-story building, where every floor can be represented by a macro constraint, and the whole building is represented by a vector of the FLOOR macro constraints.

3.6.4.1 Checking & Advisors:
An advisor is a body of specifications (standards) that is used in design. Standards are collections of specifications that are frequently used in various designs. They are organized, and enforced, by code-permitting authority, whether national, local, or commercial. Standards are used during the engineering process in one of the following ways:

* **During design synthesis:** an advisor may help in constraining the design space, i.e., helping in selection of a design alternative

* **During checking:** an advisor shows the standards in a format to be compared against the actual design. A discrepancy between the two is considered a violation.

Every *article* of a *standard* is represented by a *specification.*
Chapter 5 of the National Fire Prevention Association Code 101 (NFPA 101.5) has been implemented as one of the advisors.

Standard:

It is the body of standards, or code, such as NFPA 101. It can be enforced upon an object, or a group of objects.

Enforcement is made by loading a body of standards, in their abstract (un-unified) article format, into the design process. Everytime an object is saved, the standards are automatically enforced upon it.

The user can use the same command, say **ENFORCE**, to enforce either an article of code or a whole body of standards. Hence is the development of the virtual **ARTICLE-ROOT.**

The user may turn off a group of articles, for whatever reasons. The system allows this deactivation by listing the article numbers to be turned off, the name of user who made the decision, and the time of the decision all in one list within an attribute of the standard data structure.

Article:

This is an article of specifications, that the user can enforce upon an object. It is represented by a constraint network. It may break down into sub-articles. It include the following attributes, among others:

antecedent: A logical statement that works as a filter of pertinence an object has to pass before enforcing the real body of article constraint.

constraint: The real constraint network that represents the article of specification written in constraint representation. It is a constraint network that gets unified with every pertinent object, i.e., every object that passes the antecedent filter. Unification takes place by replacing a keyword, *OBJECT*, by the unique name of the object being checked before evaluating the constraint network.

Domain-independent parsers can be developed to translate the limited-vocabulary verbiage of articles into the antecedent and the constraint portions. An Augmented Transition Tree (ATN) parser has been tried in the New York State Space Conditioning Program for Life Safety.

Violation:

It is the flag that gets raised whenever the system discovers uncompliance of an object to a body of specifications. Violations are flagged out for one of two purposes:

* reporting the incompliance
* enqueueing the violation for remedial action.

Structure (Artifact):

This object is created for inheritance purposes. It describes a major structure (artifact). A structure is a topmost component of the project. Its inter-dependence with the other structures of the project is minimal; an example for that is the Gymnasium of an industrial plant. While the gym is a part of the plant, it is loosely related to the other buildings in the plant. A structure is to be inherited by all the topmost domain-specific objects.

__Project:__
A project may consist of several structures to collectively serve one functionality. In this data structure, all the project-specific information is saved.

3.6.5. Representation of Geometry & Graphics:

This, by no way, is a full description of the data structure of the geometry/graphics of a CAD system. This section contains a rudimentary investigation of the difference between functional constraints and geometric ones is given. Only the geometric/graphical objects that are central to the development of the system environment are described.

3.6.5.1 Difference between Functional Constraints and Geometric Constraints:

In ordinary functional/geometric model, where every geometric detail is mapped into a functionality, there should be no difference between functional constraints and geometric ones. Nevertheless, it is expected that designers used to current Computer Aided Drafting systems will be more interested in using systems similar to the one presented in this dissertation because it builds on top of existing CAD systems. This happens if functionality is assigned to a clump of geometric constraints, with no clear breakdown of geometry to map into the functionality breakdown. In such a case, the following differences will emerge:

° Far higher frequency of conflicts between geometric constraints

° Far higher number of initial premises, or degrees of freedom, in every geometric conflict.

None of the examples, in chapter five, did run into such a problem because proper functional mapping of all geometric details was used. Therefore, the following suggested solutions were not needed in the current version of the system.

3.6.5.2 How to Handle These Differences:

One approach to the problem is to introduce a **different kind of constraints** to be applied to geometry and graphics. Geometric constraints need to have the following features:

° lowest persistence (lower strength in the value system which forces the conflict resolution mechanism to try a change in the geometric constraints before the rest of network.)

° resolution of conflicts between geometric constraints should take chronological precedence into consideration[5]. It should not be, the only criterion in conflict resolution.

° Most of the geometric constraints can be condensed into a primitive constraint, either manually or by using an algebraic simultaneous equation solver (Gosling 83).

<u>**Pros:**</u> The separation of geometric and functional constraints is an architecturally-sound decision allowing the functional modeler to run on a separate processor.
<u>**Cons:**</u> Additional complexity.

Another line of thought is that we do not need another category of constraints to work on nodes different from the one working on cells. We can can use the same type constraint with one slight modification, **delayed geometric conflict resolution.**[6] This could be done as follows:

° Whenever the user intends to geometrically edit a constrained body, the system automatically halts the propagation mechanism for all (geometric?) constraints.

° The user introduces a set of changes (constraints).

° These changes are literally executed, i.e., graphically-displayed, without propagation. In other words, the changes will be carried out in the narrowest graphical sense until the geometric editing is finished.

° When the geometric editing is finished (the user announces that), the system resumes the propagation mechanism.

° An additional contradiction queue is assigned only to geometric constraints. Conflicts are not to be dequeued from it while geometric editing is going on.[7]

° By the time the system starts to dequeue the geometric-contra-queue, conflicts may not exist anymore.

5 Chronological sequence is the only criterion used by the existing parametric solid modelers, as in CADDS, Pro Engineer, etc. This is similar to Schank's Script (Schank 1983).

6 We owe the general idea of delayed geometric conflict resolution to discussions with Francois Leger of Prime Netherlands about preservation of geometric consistency. Geometric consistency is of perennial importance in the Design Automation Language (DAL) of the CALMA's DDM CAD system. Interpretive DAL deals with geometry in two levels, Graphics and pointers. DAL allows for dynamic scoping of pointers that will not transpire to graphics until the whole DAL macro is evaluated. So a step within the DAL macro may violate the geometric consistency, but that does not matter since a later step may re-establish the geometric consistency again. So by the time the whole macro is evaluated, only real errors will be detected.

7 That raises the issue of distiguishing geometric constraints from functional constraints. The simplest way is through nomenclature. Any geometric constraint type has to be preceeded by the prefix, "**geom-**."

Example:

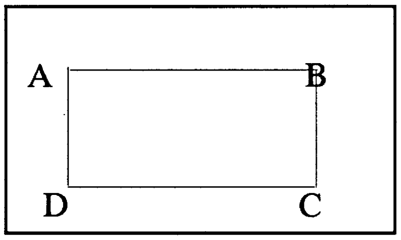


Figure (3.3): A. Insert Rectangle

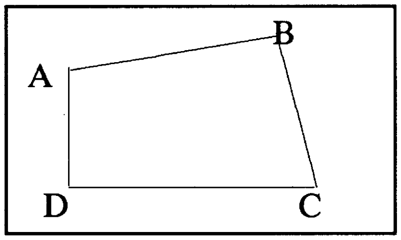


Figure (3.4): B. Point (B) is

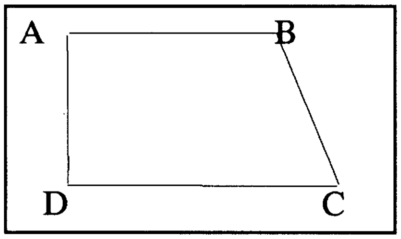


Figure (3.5): C. Point (A) is

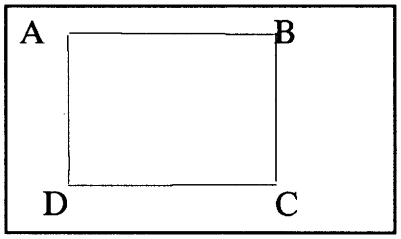


Figure (3.6): D. Point (C) is

- The user enters geometric editing mode.
- The user creates a rectangle ABCD.
- The user moves point (B).
- The user moves point (A).
- The user exits the geometric editing mode.
- The system propagates all constraints and discovers a contradiction between absissa of B and that of C.
- The user selects the absissa of C to be relaxed, as in figure (D).

3.6.5.3 Objects and Constraints Needed

We need to represent the elementary blocks for geometric modeling. The traditional approach is implementing the wide hierarchy of objects of node, line, polygon, circle, arc, box, sphere, cone, etc. This approach was followed in the system presented in this dissertation. Nevertheless, before describing the approach, we need to point to another approach, the polynomial representation.

Polynomial representation of objects:

Using a single representation, which is a polynomial series, we can represent any object. A node will be represented by scalar values. A line will be a polynomial of the first order. A spline curve could be a polynomial of the second order and so on.

Back to the traditional representation:

Coordinate (Cell):

The basic element in this representation is a (coordinate) value, which is represented by a cell. All geometric entities are constraints built in terms of these coordinate cells.

Node:

It represents a point in the space by the three coordinates and a unique name (locator). The node structure is made for semantic reasons, since processing happens at the level of individual coordinate values.

Line:

It represents a straight line in space. The attributes are intentionally redundant to allow flexibility.

start-node, end-node: where you can enter either a point name or a list of coordinates.

length, slope

type: CAD applications require several types of lines. Each line has its own methods, e.g., SL1 is to envelope an area, SL2 is a solid line that you cannot walk across or see through, etc.

Polygon:

It is similar to a vector macro constraint. Similar to vectors, number of structures (lines or nodes in this case) is fixed. So versions of polygon could be: rectangle, triangle, pentagon, etc.

Problem: Try to think of a polygon vector with variable number of vertices!

Polynomial representation might be a solution for the indeterminacy in the number of nodes.

Curve:

Any conic curve can be presented by a macro-constraint, since its parameters are finite. The problem is in representing splines, where the user can define undefined number of vertices through which the curve has to pass. A polynomial representation may be a solution; for every additional vertix, all that is needed for the representation to fit the curve is adding one or more terms to the polynomial.

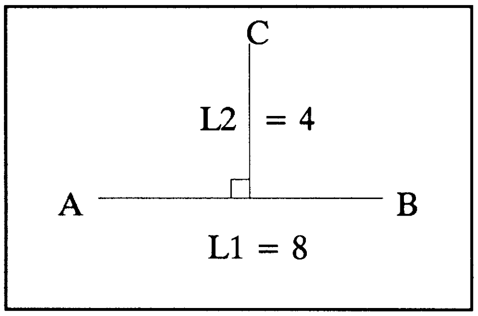
angle:

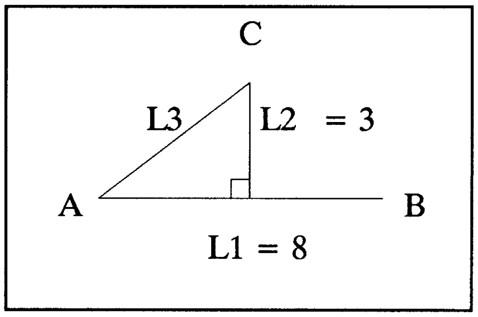
It is a syntactic constraint between two lines, in the user language, that gets translated into a constraint between the slopes of the two lines.

Example:

The user types the following commands:

- **Insert point A (0, 4, 7)**
- **Insert line L1 from point A with a length of 8 at a horizontal slope.**
- **Insert line L2 from midpoint of L1 and perpendicular to L1 and with length of 4.**
- **Insert line L3 from point A to end of L2 at slope of 0.75 at length of 5.**





The system will report a contradiction, since L3 connects point A and end of L2. The system cannot satisfy both the length constraint and the slope constraint.
The system will ask the user to move A, B or C.
In this case, the user selects C to be moved, which will shorten L2 to 3 units of length.

3.6.6. Solids:

A solid geometric model is the unambiguous and informationally complete mathematical representation of the shape of a physical object in a form that a computer can process. There are three principal types of solid models: wireframe, boundary surface, and solid. We can say that *A* is a model of *B* if *A* can be used to answer questions about *B*. In solid modeling this means asking and answering questions about an object's volumetric properties, such as weight and moment of inertia, and about similarly appropriate topological properties, such as connectivity and containment relationships (Mortenson 85).

There are two major representation schemes for solid models: *Constructive Solid Geometry (CSG), and Boundary Representation (B-Rep).*
In **CSG**, an object is described in terms of elementary shapes (half-spaces) or primitives (bounded primitive solids). The solid is represented by a binary tree of set or Boolean operations. The leaf nodes are solid primitive shapes sized and positioned in space, and the branch nodes are the set operators of union, difference, and intersection. In the case of half spaces, a simple block can be represented as the intersection of six planar half-spaces.

In **Boundary representation**, a solid is represented in terms of its spatial boundary, usually the enclosing surface with some convention to indicate on which side of the surface the solid lies. A solid can be represented as a union of faces, bound by edges, which are bound by vertices. Faces lie on surfaces, edges lie on curves, and vertices are at edge end-points.

A solid modeling system, whatever the scheme it uses, has to maintain two principal types of data describing a model - geometric data and topological data. Geometric data consists of the basic shape-defining parameters, e.g., the coefficients of a bicubic surface. Topological data include the connectivity relationships among geometric components.

One of the limitations of solid modelers is the way they handle topological information. Whenever a solid is edited, the modeler backtrack (de-construct) the solid in the exact reverse of the chronological order of construction. The problem with chronological backtracking is that it triggers a lot of unneeded evaluation of steps that happen to be made between the relevant steps. Dependency-directed backtracking, as the one used in this system, avoids this problem. Therefore, functional constraints can be used for topological aspects of geometrical models. The current system, nevertheless, does not handle solids but there is no reason for it not to be able to handle it.

3.6.7. The Interface Language:

The Design Description Language used for the interface between the modules is **Specification**, which is a Lisp-like language. Nevertheless, a limited-vocabulary "Verb Noun modifiers" syntax of plain English (using an Augmented Transition Network parser) is developed for various applications. The parser will translate the English description into the Lisp-like constraint language.

Examples of the verb-noun-modifier syntax, in the area of geometric modeling:

Insert point 4,5,8.32

Insert line from point F at slope of 1.2 for length of 9.45

Examples from other areas

Enforce article NFPA-101.5.3.2 on floor-3

Set turbine-A.discharge to 100.

4. Description of the Computer System

In this chapter, a description is given to the various parts of the system architecture and the computer programs that represent it.

4.1. Terminology

Constraint:

A relationship between parameters that could be used in any direction. It could be an equation, inequality, qualitative condition, etc. The following is a constraint: X + Y = Z.

Constraint type:

Constraint type is a general description of a constraint, such as ADDER, rather than an instantiation of an adder, as the one given in the previous definition. It has rules - one for propagation in each possible direction.

Rule:

A rule is the procedure, defined within the constraint type, to propagate (compute) values in one specific direction. Every rule has triggers, which are a subset of the pins (arguments) of the constraint type. A pin is triggered if it changes value. If all triggers of a rule are triggered, then the rule gets fired, i.e., it performs the propagation (computation) procedure defined in it.

Macro Constraint:

A constraint network wrapped in a blackbox, so the user can deal only with a limited set of parameters (pins) governing the behavior of that black box. A macro constraint could be used to represent parts of mechanisms.

Cell:

Every argument (parameter, variable or constant) and constraint pin is represented by a cell. Every cell has its value kept in a *repository*. If two cells got equated, they share the same repository. In addition to value, every cell has both merit and strength among other attributes.

Merit:

If x is f(y), then y should have a higher merit than x. If z is assigned a value by the user, then it has the highest merit.

Strength:

The values range in strength between unchangeable constant to default. The user can assign his own strength values, such as Set-by-Mechanical-Engineer, Set-by-Electrical-Engineer, Set-by-Manufacturing-Engineer. These different strengths are to be used in conflict resolution.

Node:

When two cells are equated, they share the same value repository and thus they form a node.

Initial Premises:

The ultimate source(s) for a parameter to have a specific value. The reasoning chain is traversed through a dependency-directed backtracking of the merits of parameters.

Contradiction:

It happens when two cells, with unequal repository values, are equated. They cannot share a repository to form a node.

Conflict Resolution:

When a contradiction is detected, the system sets the initial premises of the opponents as culprits and submits them to a court. Judgement is based upon the strength of values of the culprits.

Circular Dependency:

It happens when a cell belongs to the set of its initial premises.

4.2. Syntax

The system has an interpreter/compiler that uses LISP-like syntax.
The following are predefined features:

DEFPRIM	to define a primitive constraint type.
DEFCON	to define a macro constraint type.
= =	to equate two cells.
DISEQUATE	to undo = =.
WHAT	queries the system for a value for a cell. If it finds a value it reports it with a brief reasoning, otherwise it tells the user the possible ways (parameter assignments) that will grant a value to the queried cell.
WHY	It gives one-constraint-deep reasoning for a cell value.
WHY-ULTIMATELY	It gives a detailed reasoning for a cell value back to the initial premises.
DISSOLVE	to dissolve a cell's links to any other cell, or constraint.
STATS	Vital statistics about the different activities of the system.
CHECK-CIRCULARITY throws	It checks a macro constraint for circular dependency. It the user into a schematic editor to edit the constraint network.
SCHEMA	It invokes the schematic (mouse-sensitive) editor.
QUEUE-STATS	Gives status report about all queues.
RESET-QUEUES	It makes all queues empty. In other words, it indefinitely sto ps propagation of current constraints at the current status.

Some of the primitive constraints that come with the system:

Basic arithmatic operations:
ADDER, MULTIPLIER, EXPONENT, LOG

Inequalities:
LESS, LESSER!, ?LESSER, MAXER, MINNER

Trigonometric Functions:
COS, SIN, TAN

Set Operations (which are vital for discrete parametric modeling):
ONEOF,
FIRSTONEOF,
ASSUMPTION

4.3. Knowledge Representation

The following paragraphs describe the data structures of the various objects necessary to carry out the methodology presented in the previous chapter. Almost every object specified in (Steele 80) got modified heavily for handling procedures and methods. New attributes are described in detail. The attributes that have been defined in (Steele 80) are mentioned, hereafter, without explanation though.

The section starts by establishing some axioms of representation, then it moves to describing the representation used for implementing the constraint propagation mechanism. Finally, the objects used for checking and case-building are described.

4.3.1. Polymorphic Representation of Objects

Every object has to be described in three levels:

Functional level: which deals with design functionality issues.

Geometric level: which deals with the geometric representation of the functionality, as described above.

Graphical level: which is the only level that is platform-dependent.

Every level of them is represented along three dimensions:

* **Attributes:** These are the passive entries that describe the object.

* **Methods:** which are the active procedures characteristic to the object.

* **Tools** to manipulate attributes and methods, and the object at large.

Therefore, for every object the following table has to exist, in one form or another.

	Object/Attributes	. Methods	. Tools
Functionality			
Geometry			
Graphics			

4.3.2. Constraint Propagation

The following are the objects needed for implementing the constraint propagation mechanism, as described in the previous section.

<u>**Cell:**</u>

Every argument (parameter, variable, or constant) or constraint pin is represented by a cell. Every cell has a repository where it keeps its value. In addition to value, every cell has both merit and strength. The data structure has the following attributes:

id	repository
owner	name
contents	state
rule	equivs
link	mark

<u>**Repository:**</u>

It is a data structure that describes the place where a cell value is to be stored. Repository itself does not hold a value. If two cells are equated, then their repositories are merged. It has the following attributes:

id	cells
supplier	nogoods
contra	

<u>**Prim:**</u>

The structure PRIM is used for the graphical representation of a constraint. The structure is saved in the association list of the GRAPHICS slot of the CONSTRAINT structure under the locator GRAPHICS. It has the following attributes:

ID: A unique identifier.

ORDER: The hierarchical order of the constraint with respect to the topmost (macro-) constraint being displayed. Order will be 0 if the schema of the prim's constraint is the one intended to be displayed. The order will be 1 if the prim's constraint is a device in the macro constraint being displayed. This attribute is important to control the level of explosion in graphical display.

TYPE: It shows whether the prim is top-level or not.

NAME: It is automatically generated by the system. The system uses the name of the constraint followed by "**-prim**".

X, Y: Coordinates of the center of prim. It gets generated everytime the prim is displayed.

CON: The constraint represented by this prim.

PIN-POINTS: A list of the vertices of symbol (prim) that represents the constaint.

HOTSPOT: It is the mouse-sensitive area associated with prim symbol in the schema.

ROTATION: The rotation angle, in degrees, measured from eastward axis. Rotation is determined according to the relative position of a prim with resepect to the other prims in the enclosing macro constraint.

Queue:

Is a stack of operations yet to be performed. The system has several queues for control of propagation. The whole propagation mechanism is based on performing various operations on the entries of the various queues. Types of queues: CONTRA-QUEUE, VANILLA-QUEUE, NOGOOD-QUEUE, PUNT-QUEUE. An additional queue that was added for handling geometric conflicts is GEOMETRIC-CONTRA-QUEUE. This queue is not to be processed while geometric editing is going on. For further discussion about geometric constraints, see p. 57. The data structure has the following attributes:

name	entries
count	

Rule:

A rule is a method that represents the use of the constraint type in a specific direction when triggered by new values of some pins to do something, such as calculating a pin or more.

triggers	outvar
code	bits
ctype	id-bit

Constraint-type:

It is the description of a category of constraints, such as ADDER. The description includes the various rules that represent the behavior of the constraint type in various directions, and for other uses such as adding a new value, forgetting a value, or making a nogood list. The attributes of the constraint-type data structure are:

name	vars
added-rules	forget-rules
nogood-rules	symbol
initfn	

instantiations: It is a list of buckets of all the instantiations (constraints) that were saved as part of design cases (as described in the case library section, p. 86). So the list of instantiations starts as an empty list. Every time a case is being saved, a pair-list bucket gets appended to the instantiations' list by an **after** method of the object CASE. The bucket is a list of two items:

(a) the case's partname

(b) feedback keyword[1], which is one of the following: unevaluated, successful, failure.

In case of failure, an intricate mechanism for inductive learning (from failures)
needs to be developed.

graphics: the prim used to graphically represent the constraint-type in any schema.

condensed-net: if this attribute is non-nil, that means that this constraint-type was automatically synthesized to replace a device's[2] subnetwork that contained a circular dependency loop. The replaced subnet is listed in this attribute.

Constraint:

A relationship between parameters. It could be an equation, inequality, qualitative condition, etc. It is an instantiation of the category of constraint-type.

name	owner
ctype	values
info	queued-rules

graphics: a pointer to the data structure that represent the graphical representation of the constraint if it has a geometry.

causal?: A flag to be true be true if the constraint represents a causal relationship.

Macro-constraint:

A constraint network wrapped in a blackbox, so the user can deal only with a limited set of parameters (pins) governing the behavior of the blackbox. A macro constraint could be used to represent a complex (part of a) mechanism. It has the following attributes:

name	owner
mctype	values

graphics: a pointer to the data structure that represent the graphical representation of the macro-constraint if it has a geometry.

causal?: A flag to be true be true if the macro-constraint represents a causal relationship.

Macro-constraint-type:

The stereotypical description of a macro constraint. So it can be used for the generic description of a complex (part of a) mechanism.

1 Not worked out yet, since it is left as a germ of inductive learning capability in the system.

2 A device is a constraint-type as opposed to real constraint.

name	pins
allvars	creations
connector	graphics

instantiations: a list of all instantiations (macro-constraints) in the *case library*.

<u>**Vector:**</u>
A macro constraint that include several copies of a smaller macro constraint attached to each other in the same way. Vector is good for representing an artifact with repetitive components, such as a multi-story building, where every floor can be represented by a macro constraint, and the whole building is represented by a vector of the FLOOR macro constraints.

4.3.3. Checking & Case Building Representations

The following iare the data structures needed for design checking and for building design cases into the *case library*.

<u>**Standard:**</u>
It is the body of standards, or code, such as NFPA 101. It can be enforced upon an object, or a group of objects, in either of two ways:

* **Checking** for violation, mildly (i.e., just flagging out), or

* **Synthesis:** It could be invoked in the background of the design process, such that everytime a new object is created, all pertinent articles of the code are imposed upon it for flagging active violations. These violations will force the system to look for acceptable solutions for the design.

Enforcement is made by loading a body of standards, in their abstract (un-unified) article format, into the design process. Everytime an object is saved, the standards are automatically enforced upon it.

The **Standard** data structure has the following attributes:

name: The short name of the standards, that will be used as a prefix for all articles.

full-name, year: self explanatory

root-article: a virtual article whose children are all the topmost articles of the code, such as chapters. It gets generated automatically. This feature allows the user to treat articles and standards interchangeably.

constituents: A list of all the direct children of the root-article.

deactivation: The user may turn off a group of articles, for whatever reasons. The system allows this deactivation by listing the article numbers to be turned off, the name of user who made the decision, and the time of the decision - all in one list within an attribute of the standard data structure. The default value of this list is nil, the empty list.

The user can use the same command, say **ENFORCE**, to enforce either an article of code or a whole body of standards. Hence is the development of the virtual **ARTICLE-ROOT.**

<u>Article:</u>

This is an article of specifications, that the user can enforce upon an object. It is represented by a constraint network. It may break down into sub-articles.

name: The unique name for the article within the realm of the standards.

standard: The standard to which the article belongs.

title: The title of the article, if any.

parent: The parent article, if the article is a sub-article. An after-method will add this article to the list of children of the parent article.

children: A list of child articles.

verbiage: The exact words of the specifications, whose constraint translation is enforced upon object(s).

related-articles: A list of the related articles. The invocation of them depends on the degree of strictness of enforcement.[3]

antecedent: A logical statement, that works as a filter of pertinence an object has to pass before enforcing the real body of article constraint.

constraint: The real constraint network that represents the article of specification written in constraint representation. It is a constraint network that gets unified with every pertinent object, i.e., every object that passes the antecedent filter. Unification takes place by replacing a keyword, *OBJECT*, by the unique name of the object being checked before evaluating the constraint network.

Domain-independent parsers can be developed to translate the limited-vocabulary verbiage of articles into the antecedent and the constraint portions. An Augmented Transition Tree (ATN) parser has been tried in the New York State Space Conditioning Program for Life Safety.

<u>**Violation:**</u>

It is the flag that gets raised whenever the system discovers uncompliance of an object to a body of specifications. Violations are flagged out for one of two purposes:

* reporting the incompliance

3 It is expected that the use of systems such as the one described here, will encourage the use of more specific language, especially in endorsing related specifications, unlike the following common statement:
"**The door will comply with the spirit of Chapters 3 through 28 of the ASTM-105.**"

* enqueuing the violation for remedial action.

id: It is the unique identification for the system.

site: The object within which the violation was discovered. It might not be the culprit, since the culprit could be a related object that got accessed through the site object. Take the following article for example:

"For every patient room that has more than 2 beds, the door has to be wider than 36 inches."

If this article is violated, then the violation-site will be room-238, while the culprit is door-67A.

article: The article that was violated.

culprit: The specific object that triggered the violation. The determination of the culprit needs an elaborate bookkeeping mechanism that is achieved by the tree-parsing reasoning of the conflict resolution system.

description: A text string generated to fully describe the violation, i.e., id, site, article, culprit, its current value and the required value for it.

Structure (Artifact):

This object is created for inheritance purposes. It describes a major structure (artifact). A structure is a topmost component of the project. Its inter-dependence with the other structures of the project is minimal; an example for that is the Gymnasium of an industrial plant. While the gym is a part of the plant, it is loosely related to the other buildings in the plant. A structure is to be inherited by all the topmost domain-specific objects. It includes the following attributes:

name: a unique name for the structure within the case library.

library: The name of the case library that includes the structure in it.

graphical file: This is the file, or directory, that includes the graphical representation of the structure.

and some attributes for the adaptive learning purposes.

mcon

Project:

A project may consist of several structures to collectively serve one functionality. In this data structure, all the project-specific information is saved.

name: a unique name

structures: a list of all major structures (artifacts) that comprise the project. For example, a hospital project may be comprised of the main hospital building, the guard house, the power generation room, the fence, the checkpoint, ... etc.

case: The pathname of the case file (of the Case Library) in which the project is represented.

designer: the name of a designer, or design firm for tracking purposes and for future inductive learning capabilities of the system.

date: submission date of the design.

4.4. The Central Core

This is the group of generic problem solving capabilites denoted by the thick-line shield in the center of figure (3-2). It is the group of domain-independent capabilities, similar to the capabilities of a novice engineer. It consists of the following units:

4.4.1. Constraint Propagation & the Roundtable

The *constraint propagation and roundtable* mechanism is provided as part of the Design Description Language. It is based on Steele's constraint system (Steele 80) with major differences. The following paragraphs describe those major differences:

* The system utilizes an extended architecture whose elements are described in the last section.

* Steele's Constraints have a weak, often-inconclusive conflict resolution mechanism. The system has the capability of conflict resolution according to user-defined rules. It handles conflicts between the various advisors.

* For a system to be used in any engineering application, it has to handle real numbers and symbolic values. Steele's constraints can not handle either. This system, on the other hand, handles both real numbers and symbolic (alphanumeric) values.

* The system allows saving queue images. A queue image is a freeze shot of the queues at a specific instance. You can halt the propagation at any point and save the queue image. You can resume the propagation on another day or in another process, whenever you want. This capability is quite useful in a case of working with limited memory (specifically heap space). Such situation may arise on a PC-based Lisp or on a workstation whose memory is exhausted in other applications, e.g., CAD/CAM.

* All constraint propagation systems suffer from circular dependency. The system has a circular dependency checker, which searches for circularity within a macro constraint. If it finds one, it reports its smallest manifestation[4] both textually and schematically. If the discovered circularity fits one of the paradigms the the system can handle, the system will ask for user authorization to replace the loop by an instantiation of a pre-defined primitive constraint. If the loop does not fit into one of the known paradigms, it will be reported to the user and he is responsible for replacing it.

Limitiations of that solution:
1. The system should have an extended[5] solver that can replace any loop by a primitive constraint synthesized on the fly.

2. For pragmatic reasons, the system searches within a macro constraint. Circularity loops extending across more than one macro constraint, or outside any macro constraints, will not be found. But this limitation is applied also to our human logic, where every premise is supposedly syllogistism-free, while circularity may exist across the premise boundaries. In constraint propagation, circularity has a more pronounced shortcoming.

4 Extracting the minimal manifestation of a circular dependency in a constraint network is done by making sure that no loop (closed sequence of cells) is a sub-sequence of another loop.

5 The system needs a solver that can handle all the kinds of primitive constraints, i.e., both algebraic operators, and non-algebraic one (such as <, gate, one-of, etc.), and in the meanwhile can replaces loops by primitive constraints synthesized on the fly. This capability is easier to be achieved for algebraic constraints.

4.4.2. User Interface

The Design Description Language handles constraints, in an extended version of Steele's Constraint syntax.

Domain-specific user interface extensions are developed outside the central core. Natural language parsers, using Augmented Transition Trees (ATN), were developed for various applications (Winston & Horn 88, Gazdar & Mellish 89), including the design checking of hospitals described in Chapter five, p. 114. Figure (4.1) shows part of the object declaration. Figure (4.2) shows one syntax of a statement.

```
(compile-tree  object
   (brnchs
       (door            rtn          'doors)
       (doors           rtn          'doors)
       (window          rtn          'windows)
       (room            rtn          'rooms)
       (exit            rtn          'exit)
       (corridor        rtn          'corridors)
       (ward            rtn          'wards)
       (wards           rtn          'wards)
       (floor           rtn          'floors)
       (cabinet         rtn          'cabinet)
       (nurse-station rtn            'nurse-station)
       ...
       ))
```

Figure (4.1): objects in the Hospital Design Checker

```
(compile-tree attributes
   (brnchs
     ( attribute  attributes rtn
                (cons attribute attributes))
        (and  attribute      rtn (list attribute))
        ( attribute          rtn (list attribute))))
```

Figure (4.2): The Syntax of one possible sentence.

4.4.3. Planner:

This is where the system gets the functional description from the knowledge base. The planner takes care of breaking it down into functional components and making sure that every component gets fully designed before starting to handle the next component.

For the time being, in the tradition of *Case-Based Reasoning*, the system builds its models in a depth-first approach. Conflict resolution, on the other hand, proceeds in a breadth-first fashion (Thagard & Holyoak 89).
No standalone planner is currently in place inside the presented environment, since the propagation algorithm offers some rudimentary planning capabilities.

This system elaborates design plans by matching a description of the desired function of the design against a catalog of fragments, each indexed by the functions it serves.
A fragment is chosen and is inserted into the design. Constraints, propagated from the elaborated plan are then used to either further specify the fragment or to give reasons why it is not really applicable. An accepted fragment may itself require elaboration, and so this process is recursive.
Design fragments may have structural as well as functional attributes. They may carry auxilliary information, not directly related to problem solving, such as graphical representations.
In its ordinary operation, the system constructs, examines, and discards many fragments. All of these fragments, along with their uses and limitations (discovered by rejection), are entered into a case library to be used as starting points for designs.

4.4.4. Catalog

This is where the system keeps a generic description about every category of objects, such as beam, column, portal frame, support, hinged-support, roller-support, adder, amplifier of a car radio, etc.

Category is a synonym of constraint-type, described in page 54.

Every category has pointers to every instantiation saved in a design case.

Catalog files are designated the extension "**.cat**" for conventional distinction and gathered under one subdirectory.

Every synthesis process starts with searching the catalog for an item (constraint-type) that has the same, or synonymous, functionality.

4.4.5. Checker:

This is the module responsible for enforcing a body of specifications upon a design. The user can enforce an article, with all its sub-articles, upon a design part.

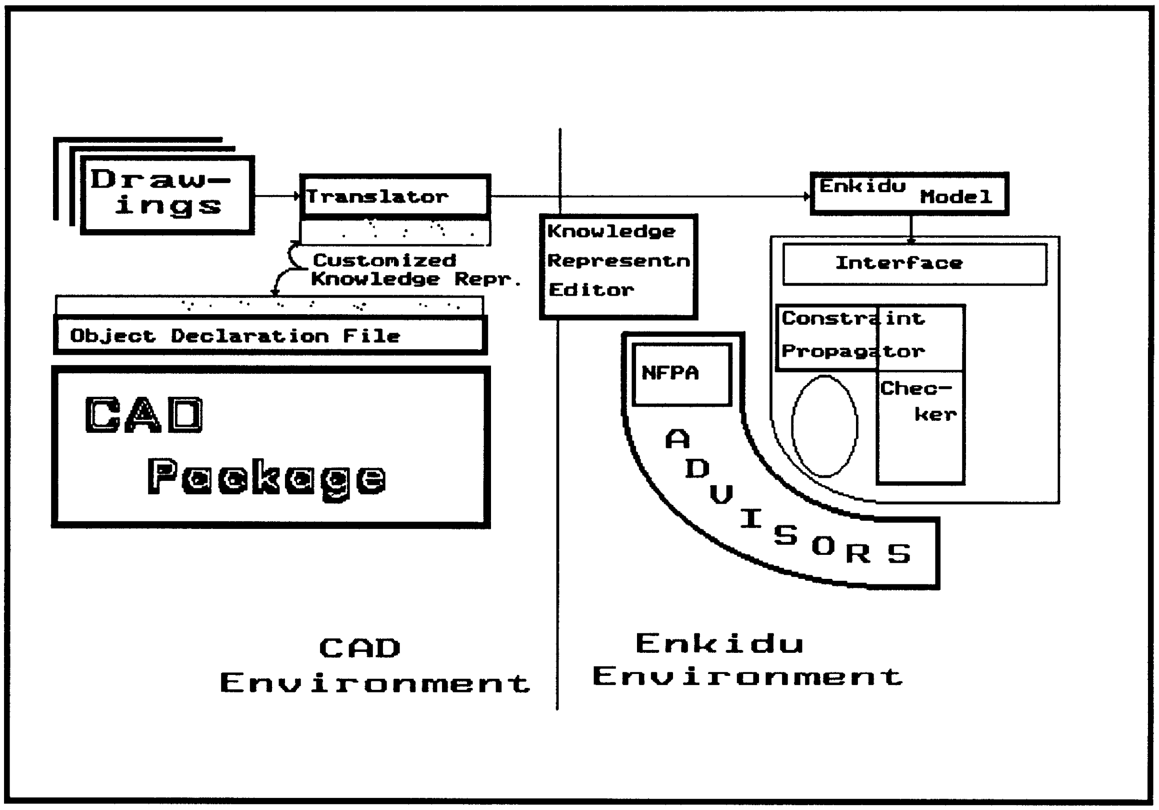


Figure (4.3): CAD - Enkidu Interaction

The Checker module consists of the following programs:

4.4.5.1 Knowledge Representation Editor

Every domain has its own knowledge representation. A knowledge representation is a description of the world of the application, its objects, and the methods of interactions between these objects. The system offers a tool for editing these representations.

The current implementation of the KR Editor is CAD package-specific, since it is written to handle the objects and attributes of Prime MEDUSA only. It allows the user to edit a network of objects and their attributes. The basic capabilities of the KR Editor are:

* Add an object
* Add an attribute
* Add method
* Edit object
* Edit method

Every CAD package has one, or more files for the declaration of objects and their attributes. The way methods are defined for these objects are quite different from one package to another. The procedure to enact the changed files is quite laborious and cumbersome.

The KR Editor relieves the user of worrying about these differences and it allows him to graphically parse a network of objects to define the new world of the application.

A smaller version of the KR Editor has been developed for CADDS 4X.

The final KR Editor should have the same interface on all the packages, using the *de facto* standard X.11 Windows.

4.4.5.2 Translator

It is a CAD package-specific translator. It translates the information from the CAD part database into system Specifications (Objects & Constraint network). Basically, the program duplicates the geometric and non-geometric information on a drawing into a format comprehensible by the system environment.

A translator has to be developed for every CAD package, until PDES[6] standardizes the exchange of data at all levels.

A FORTRAN translator has been developed between the system and Prime MEDUSA[7]. This Medusa translator fetches data out of the MEDUSA database, and the MIDAS + database for non-geometric attributes. For more description refer to Chapter Five (p. 114 .)

4.4.5.3 Gchecker

It is the file that declares the objects **Standard, Article, Violation** (as described in p. 71), and the related methods and procedures.

The Concept of Unification

The antecedent (screen) and the constraint parts of an article are written in a generic specification (constraint) format. Everytime an article is imposed upon an object, a copy of its constraint is made. In that copy, the variable &VARIABLE& is replaced by the name of the object being checked. Then the copy is evaluated into a constraint network to impose the constraint.

The major procedures, in the GCHECKER file, include:

VALIDATE-ARTICLE:

It is the top-level checking function that does two jobs:

- 1. if the article-antecedent is non nil, invoke the *check-article* function.
- 2. invoke all children articles, if any.

CHECK-ARTICLE:

It is a function that checks the validity of article's constraint and reports a violation if the constraint is not satisfied. This function does not care about the article's filter (ante), since this is the job of VALIDATE-ARTICLE.

6 PDES stands for Product Data Exchange Standards. It is a standard initiative launched by the DoD for the exchange of different levels of data. It has four levels of data exchange. The first deals with geometric information between two applications. The second level adds non-geometric information to the exchange. The third deals with non-transitive constraints. The fourth deals with full-fledged constraint propagation (Harrelson & Silvistri 1988). So far, this system offers the only language proposed for the fourth level.

7 A Computer-Aided Design (CAD) package for mechanical and AEC applications.

REPORT-VIOLATION:

It does the following 2 jobs:

- ° It reports the violation id in a diamond on a separate error layer on the drawing, and
- ° it also print a detailed description of the violation in the violations report.

ENFORCE

This is a function to check the compliance of an assembly, e.g., ward, to an article.

ENFORCE-STANDARD

This is a function to check the compliance of an object, possibly an assembly, to a whole body of standard.

4.4.5.4 Checker:

It is the domain-specific file that includes the additional objects, methods, and procedures.

An example of a domain-specific extension is the AEC extension built as part of the Code Orientation, Review, Enforcement, Commissioning and (CORECT) system described in chapter Five (p. 114).

4.5. The Knowledge Bases

Knowledge bases are files of domain knowledge, kept in three forms: (1) objects, (2) Specifications (need unification to become constraints) format, and (3) features. This knowledge is used to synthesize constraint types and save them into the catalog. For some domains, specifications may exist in a limited-vocabulary plain English (natural language) if parsers were developed for the domains.

A specification could be used in major ways: one way is to check the conformance of an existing design and the other way is to use the specification for design synthesis (from scratch).

4.5.1. Objects

Objects in the knowledge base has, at least, the following types of relationships:

* Causal relationships

* Taxonomical relationships

* Quantitative relationships, such as equations, inequalities, etc.

* Any other relationships required for specific applications, such as nosological relationships needed in heat transfer applications, and plate stress transfer applications, e.g., the design asphalt roads.

Once the semantic nets that represent the knowledge of a domain are established, one can apply to them some feedback-guided learning to improve the knowledge of an evolving domain, similar to the **harvest** mechanism (el-Shafei 86).

4.5.2. Polymorphic Specifications

Specificational knowledge in any domain has to have sufficient information on the three levels: functional level, geometric level, and graphic level, such that the system can utilize them to synthesizing two representations for each alternative available:

* **Abstract representation:** which has the functional (causal) description of the alternative.

* **Detailed-design representation:** which has the full description (functional, geometric, and graphical) of the alternative.

Specifications gets unified, as described in p. 80, to become specific constraints.

4.5.3. Features (States):

A feature is a group of geometric and functional constraints whose collective existance in an object is denoted by the name of one feature. Figure (4.4) shows an example of a feature frame in the domain of design of roads.

State/Feature:	Longitudinal-crack-lC
State-type	symptom
Class:	Structural
A-Kind-Of	Crack-C
Representative-Var:	LC
Variable-Dimension:	L
Severity-Threshold:	10 inches
Associated-symptoms:	nil
Necessary-symptoms:	nil
Contradicts-with	nil
Maybe-caused-by	(or (and block-crack-BC (or water-percolation-WP resilient-subgrade-RSG)) excessive-vertical-movement-of-underlayer-VDUL surface-fatigure-SF)
May-cause:	Potholes-PTH
Maybe-related-to:	(Surface-Stiffness-SS Base-Elasticity-Modulus-EB Subgrade-Modulus-ESG Freezing-Index-FI Equiv-thickness-HEQUIV Asphalt-Age-A Ave-Annual-Daily-Traffic-AADT Surface-Heat-Ts California-Bearing-Ratios-CBRs Heat-Capacity-C Accumulated-Load-per-year-X

Figure (4.4): Example of a Feature (State) Frame

4.5.4. Harvesting:

The harvesting mechanism as described in chapter Three on Methodology is implemented through the following modules: the Heuristic Pre-Processor (HPP), the Dimensional Analysis Unit (DAU), and the Stepwise Regression Analysis Unit (SRAU).

4.5.4.1 The Heuristic Pre-Processor (HPP)

This module collects all the maybe-related-to parameters and piles them into heaps according to some categorical classes. The rules of piling are provided by the user.

4.5.4.2 The Dimensional Analysis Unit (DAU)

This is the tool that takes every heap and creates dimensionally-homogeneous[8] groups. A group is a multiplication product of several parameters, each raised to some exponential power. The parameters in every group belong to the same class, e.g., temperature-class, stress-class, time-class, etc. DAU isolates the parameters under consideration in one group with a unary power of exponentiation. Through dimensional analysis, we make sure that the resulting relationship is physically-meaningful, and if there is a law or theory that may hold between these parameters, most probably it is the induced relationship. The DAU utilizes a sort of K-lines (Minsky 86) to keep in memory the possible combinations of parameters that form dimensionless group.

This mathematical tool is based on the formal P-theorem of dimensional analysis (Taylor 74), as described in appendix A.

4.5.4.3 The Stepwise Regression Unit (SRU)

This is the unit that takes the dimensionally-homogeneous groups, generated by the DAU, and build a relationship between them according to the occurance of these groups in all cases throughout the case library. The relationship is built incrementally, starting with the most statistically-influential groups until the relationship reaches the required correlation. The reason for stopping right after reaching the threshold correlation is because of the trade of between the added complexity of the new term versus the increase in correlation. This leverage on the complexity of the induced relationship was the reason of selecting the stepwise regression analysis, despite its fall into disuse because of its iterative nature (Bridgman 33). Example of the use of DAU and SRU is demonstrated in appendix A.

The harvesting capability is not fully integrated in this version of the system because the mechanism so far exists in a non-constraint version.

4.6. The Pre-/Post-Processors

These are external programs to be called by the system, like the Finite Element procedures to be used for analysis, and shading program used to enhance the presentability of the final design. Since the system is written in Lisp, then foreign function calls are possible to be embedded within lisp procedures.

For processors that are inherently uni-directional, iterative invocation from within the declaration of primitive constraint can simulate the reverse operation. This is how an FE processor can be embedded within the declaration of primitive constraint.

8 Dimensionally-homogeneous terms are terms that have the same dimensions. For ease of use, we build all terms to be dimensionless.

4.7. The Advisors

An advisor is a body of specifications (standards) that is used in design. Standards are collections of specifications that are frequently used in various designs. They are organized, and enforced, by code-permitting authority, whether national, local, or commercial. Standards are used during the engineering process in one of the following ways:

The system offers tools (procedures) for **defining standards, defining articles** (as shown in figure 5), **deactivating articles**, etc.

```
(gen-article 'OMH-SC&NC 1
   "Bedroom Requirements:"

   "Skilled Nursing Facility (SNF)shall be provided a minimum of
    125 square feet plus a 6 square foot closet in single bed-
    rooms and 100 square feet plus a 6 square foot closet for
    each patient in double bedrooms. No more than four will
    occupy a bedroom but, there shall be a minimum of 66.6%
    single & double bedrooms per ward."


   '(and (room-p object)
         (equal (room-type object) 'bedroom)
         (equal (ward-type (room-ward object)) 'SNF))

  '(and (cond3  result1
          ((= room-XX.capacity  1)
           (and (< =  125  room-XX.area)
                (< =  6  room-XX.closet-area )))
        ((= room-XX.capacity 2)
         (and (< = 100 (* room-XX.area % room-XX.capacity))
              (< = 6 (* room-XX.closet-area % room-XX.capacity)
         )))
        (truth (< room-XX.capacity 5)))
      ;;66.6%
```

Figure (4.5): Declaration of OMH-SC&NC Article.

4.8. The Case Library

The case library is where every design case is saved as an object. In addition to a pointer to the drawing file, the object should have the functionality of the product, constraint propagation network that represent the design, name of the designer, the reviewer, every modification/ correction, whether the design is constructed, or not yet, and if constructed, how many times this design has been used in successful analogies by the system.

Case description files are designated the extension "**.lib**" for conventional distinction.

4.8.1. Purpose of the case library

1. **Version control:** The case library can be perceived as a way to preserve a frozen status of the latest stage the design reached before work stopped on it. So, whenever a modification is needed to be added, the invocation of the case will assure re-creating a replica of the design environment (space) with all its constraints, dead ends, etc. The user can decide whether the modification should be added to the same version, or the modification will constitute a new version.

2. **Case-based Learning:** Design cases can be used for learning by analogy. as described in the methodology of the design synthesis in the previous chapter.

4.8.2. Agenda

The agenda is a mechanism that keeps track of every case till it gets deployed into operation. The agenda mechanism is responsible for collecting feedback on the design during the various stages of concurrent engineering. A feedback is a remark that did not turn into an Engineering Change Orders (ECO's), such as,

° comments during checking,

° Difficulties during manufacturing,

° design-imputed problems in maintenance.

The feedback is condensed[9] into a keyword, or key phrase. The condensed feedback is inserted as the tail of the bucket that represent the case-part in the instantiations list of the macro-constraint-type.

9 Not fully worked out yet, since it is left as a germ of inductive learning capability in the system.

4.8.3. Case

A design case is a data structure to allow *Case-Based Reasoning*. The case has to include the following three components:

* The resolved constraint-network, which include the macro-constraint that represent the artifact, in addition to all specifications imposed upon it.

* The final status of the queues right before deploying (manufacturing/constructing) the artifact.

* All feedback gathered after the deployment of the artifact.

Every time a case is being saved, an **after** method, called add-the-instantiation, appends a pair-list bucket to the instantiations' list of the (macro-)constraint-type. The bucket is a list of two items:

(a) the case's partname, which is also the name of the constraint that describes the part.

(b) feedback keyword, which is one of the following: unevaluated, successful, failure.
In case of failure, an intricate mechanism for inductive learning (from failures) needs to be developed.

The case data structure has the following attributes:

name: a unique name of the case.

partname: the unique name of the part/assembly whose design is described throughout the case. The same partname is used to define the (macro-)constraint.

constraint-filepath: the file pathname of the constraint model description of the case. It includes:
(a) the constraint network.
(b) queue image with all the nogood sets, unresolved contradictions (if any), etc. Next time the network is loaded, it is loaded with the same circumstances that concluded the design.

graphics-filepath: a pair list of the following:
(a) the name of the CAD package, e.g., CADDS4X-rev.5, or AutoCAD-rev.10, to invoke the right *Data Access Routines (DARs)* to fetch data from the CAD part databases;
(b) the file pathname of the graphical part/assembly that represents the case.

project: Name of the project (as described in) to which the part/assembly belongs.

functionality: keyword or phrase selected from finite lexicon.

specifications: A list of the specifications imposed upon the design, other than the functionality.

queue-status: The status of queues by the end of the design.

feedback-specs: These are feedbacks (specifications) gathered after the deployment of the artifact. It is a list of specifications that in case of selecting the design case as an analog for a new design problem, this list of specifications needs to be imposed upon the copy of the design case to avoid the problems of the old case that necessitated the feedback.

maintenance-log: a list of pentaplets. Each pentaplet represents a problem by five aspects:

1. *list of pairs of symptoms and severity values*: symptoms are compared against the
frames of the relevant features/states.
2. *date-of-reporting*
3. *diagnosis*
4. *remedial-action*
5. *feedback*: it is a scalar value between -1 and + 1 to be collected later on.

4.9. The Interactive Environment:

An interactive environment was partially-developed to offer a user-friendly interface. The user can iconically load different constraint libraries that represent the specifications he wants to use. The user can also edit a constraint network either textually or schematically. The environment, shown in figure (6.4), allows the user to do the following:[10]

- Load different constraints catalogs (or specification libraries). Once a catalog is loaded, every constraint-type is represented in the Catalog pane by a mouse-sensitive icons. When the user wants to add a constraint to a constraint network, in the constraint network pane, all he needs to do is to click upon the constraint-type icon in the catalog pane.
- Instantiate a constraint-type, i.e., create a constraint of the class described by the icon.
- Dissolve a constraint, i.e., disconnect it from all the surroundings.
- Connect two pins.

10 Not all of these capabilities have been fully developed.

- Encapsulate a network into a macro constraint.
- Graphically simulate the propagation of values through a constraint network.
- Change the scale
- Graphically display the status of the system (queues and statistics).
- Switch to textual editing of the constraints.

4.10.Modus Operandi: How the System Works, and What We Get Out Of It

While the afore-mentioned system architecture should work in any design/engineering task, it had to be tested in two major modes of operation:

4.10.1. (A). Design Synthesis

The user posts his specifications, which guides the system in selecting the most appropriate alternative (through the initial selection network, described in the chapter on Methodo   logy), whose parametric model is detailed and graphically displayed. The user may impose additional constraints, which are propagated through the design. The attempt to satisfy the additional constraints may cause the entire design to be revised. The user can query the system about the way (reasoning) any parameter got calculated.

The use of the system in design synthesis is demonstrated in the first example in the chapter on *Examples*. The example deals with the design of reinforced concrete exhibition halls.

4.10.2. (B). Design Checking

The user can post any body of standards, as specifications, and enforce them upon a CAD file. The system will flag all the violations in the design.

The use of the system in design synthesis is demonstrated in the second example in the chapter on Examples. The example checks the compliance of a floor plan with two chapters of the code (NFPA and NYS-OMH Life Safety code).

4.10.3. Other Uses

(C). Hybridizing Two Designs into a New One.
For example, we can use the main structural system of design (A) with the Heating, ventilation, Air Conditioning (HVAC) of design (B) to create design (C).

(D). You can **make use of the case library (experience) of another colleague/designer.**

(E). With the **cost estimate** function automatically translated into a constraint, the system can be used as an engineering spreadsheet to evaluate cost/specification relationship.

4.11. Implementation

Initially, the purpose of this thesis was design synthesis, but since code/specification compliance is expected in every design, it was obvious that the work had to be done within the framework of design automation.

Programs needed both to conduct the both design synthesis and checking were developed. These programs represent most of the system environment.
Working with a CAD vendor, Prime Computer was a chance to apply these ideas, and test them against real-life problems of real CAD/CAM customers.

Systems were written in Common Lisp. Interfaces to the CAD packages whenever needed, were written in FORTRAN & C.

The environment is a portable one, and that was proven by integrating it with several CAD systems, including CADDS 4X, MEDUSA, and PrimeDesign.

Description of the applications are provided in the chapter on *Examples*.

5. Examples

5.1. Structural Design Synthesis of Reinforced Concrete Exhibition Halls

This is an example for design synthesis of reinforced concrete exhibition halls. The example is an application of the system to explore the relationship between dimensioning, analysis, and conceptual design, as illustrated in figure (5-1).

5.1.1. Motivation

Conceptual design is the least understood stage of engineering design. Assuming that the designer made the right selection in the conceptual design stage as the design process continues several parameters get changed from their initial values. One would expect, after some changes, that at some point the designer will decide that his conceptual design is no longer the most appropriate choice. This almost never happens. What happens is some patching up of the design so it will (*barely*) meet the specifications, rather than being the optimal design. The major reason for choosing the patching route is the amount of effort involved in any fundamental (conceptual) change. With the automation of the design process, worries about major changes should be alleviated since computers help us override this taxing barrier.

- ° A good illustrating example is the area of structural design of exhibition halls. In that domain, the selection of a main structural system is made between several systems based on the outer dimensions and the architectural requirement.
- ° After the system makes its selection, it goes onto fully designing, i.e., dimensioning, the system.
- ° After the design is made, the user can say, for whatever reasons, that he cannot accept the depth of the beam. He subsequently imposes a constraint.
- ° The system will try to satisfy the new constraint with the least possible changes.
- ° The system finds that nothing short of a change of the main structural system (conceptual design) will solve the problem.
- ° So, a new conceptual design is reached, and it gets completed into a full system.

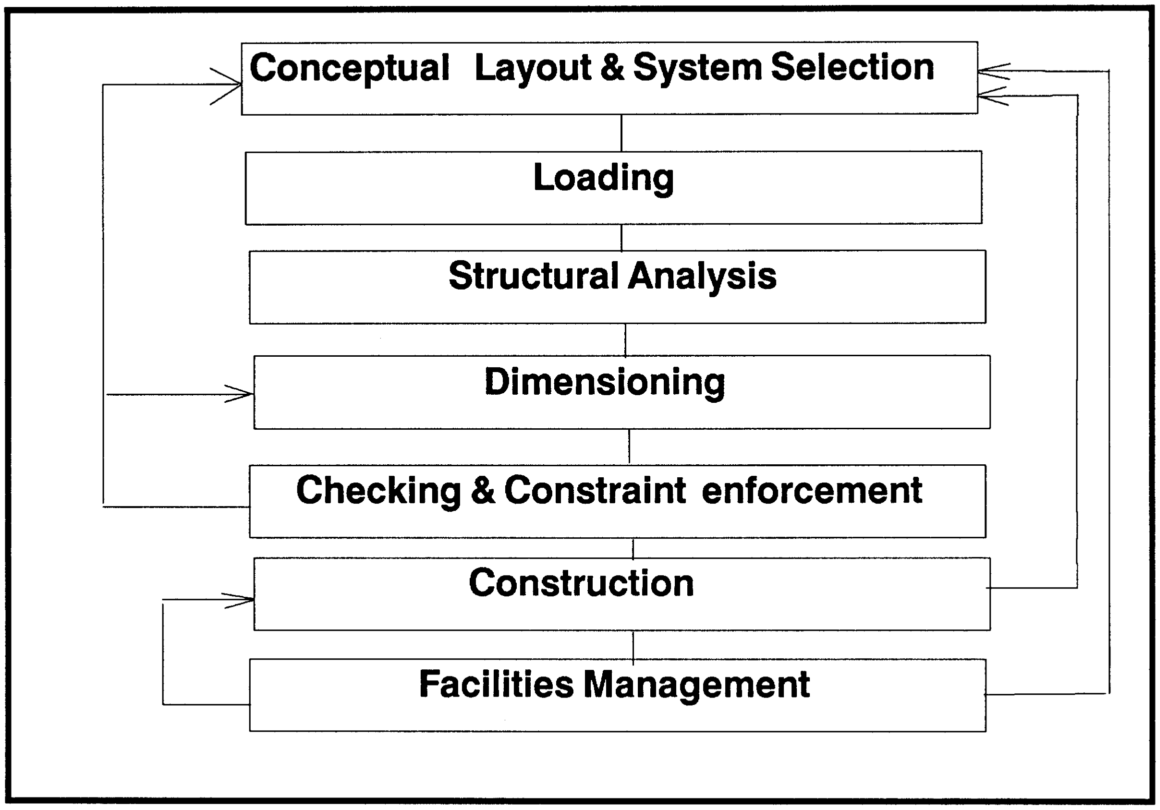


Figure (5.1): Stages of the Structural Engineering Lifecycle

5.1.2. Main Structural Systems

The main strucutral system (MSS) is the skeleton that holds the general layout of the structure in a way that meets the required functionality and that allows the transfer of the various loads to the ground (through foundations, if it is above the ground). Loads get transferred in three directions (x, y and z). Therefore a secondary MSS may exist to transfer the load in one direction, x, to a primary MSS that transfers the load in the perpendicular direction, y. The loads are transferred vertically to the foundation through columns or vertical legs of structures. An MSS consists of all these components, seconday MSS, primary MSS, and columns/legs. Sometimes the functionality of two components are achieved by one subsystem, such as portal frames which carry the load both in the y direction and the z direction. Other systems may carry the load in the three directions simultaneously, as in double-curved shells.

Theoretically, there are no limitations on span spacing between beams. The only solid constraints in structural design are maximum allowable stresses and maximum allowable deflection (strain) at the critical points. In the case of concrete structures an additional constraint on the maximum allowable crack width is added.[1]

Various main structural systems lead to various straining actions, which in turn lead to different stresses and strains. For the same land plot, different main structural systems produce different stresses and strains.

Instead of trying elaborate structural analysis and design of all main structural systems to cover a piece of land, conventional wisdom provides us with some guidelines for the initial selection of the appropriate main structural system. Unlike expert systems, initial selections are susceptible to change due to elaborate analysis or due to a change of design input.

General Assumptions:

1. These are the assumed values for the general parameters:
   Average floor thickness of 0.175 ms (7"),
   Concrete density = 2.35 ton/m^3
   Dead Load Intensity = 1.0 * 1.0 * 0.175 * 2.35 = 0.4 ton/m^2
   * (2200/1600) = 0.56 psi
   Live load intensity = 1.5 ton/m^2 * 1.375 = 2.1 psi

2. According to the American Concrete Institute (ACI) 1981 standards (Winter & Nilson 85, Thornton & Lew 83), the total design load should be calculated as following:
   Total load intensity = 1.4 D.L. + 1.7 L.L.
   = 1.4 * 0.4 + 1.7 * 1.5 = 3.11 ton/m^2 = 4.28 psi
   = 616.32 psf
   equiv-load = average-spacing * total load intensity
   = 5 * 3.11 = 15.55 t/m

3. General assumptions are listed in the "Design of Sections" section (p. 5-105).

4. Relationships for the design (maximum) values of straining actions are given for every main structural system (MSS). Using standard design of sections constraints, these straining actions are used for dimensioning the instantiations of any MSS.

1 Specifying and setting values for these limits is the job of the various national standards using various theories for failure mechanics, such as **Ultimate Strength** approach, **Working Stresses** approach, etc.

General Remarks:

A more precise approach to functional parametric modeling of structural systems will be through the modeling of the Loading Diagram (LD) by a binomial series. The Shearing Force Diagram (SFD) can be derived from the integration of the Loading Diagram. The Bending Moment Distribution (BMD) could be reached by the integration of the SFD.

$\textit{Deflection Line} = \int BMD = \int\int SFD = \int\int\int \textit{Loading Diagram}$

Limitation: To develop the model this way, constraints have to handle calculus first.

System of Simple Beams & Columns

The slab is carried directly by the simple beams, which transfer the load to the columns. The slab, the simple beams and the columns make the main structural system. To stay within the allowable stresses and strains, conventional wisdom suggests the use of simple beams & columns when the span is less than 5 meters, and spacing is less than 4.9 meters to ensure slabbing action[2]. The system is also suitable for pre-fabricated construction.

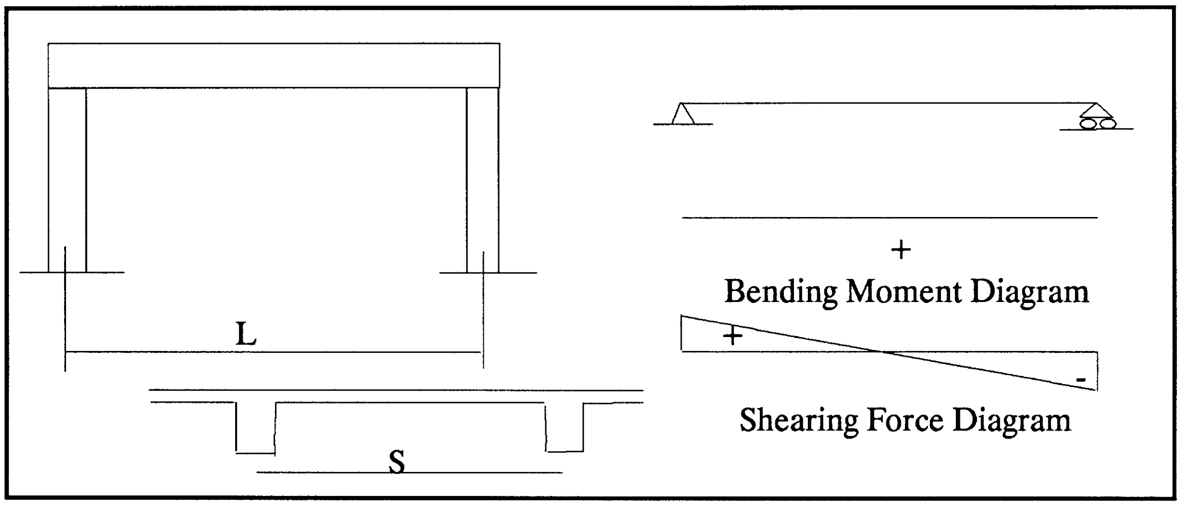


Figure (5.2): Simple Beam & Column System

Assumptions:

1. The slab load is assumed to be uniformly distributed along the span (L) of the beam.

2. Every beam is assumed to be supported by a hinge support on one end and a roller support on the other. This is a safe assumption. Monolithic casting of concrete, in case of in-situ construction, will reduce the degrees of freedom at supports, i.e., it possibly produces clamping (negative) moments that reduce the critical positive bending moment at the mid-span.[3]

Relationships:

$$Max.B.M. = \frac{wL^2}{8}$$

2 Load transfer in the shorter direction of a panel.

3 The introduction of negative moments at the ends of a beam will hang the same Bending Moment Diagram from a negative datum rather than a zero datum. Consequently, the maximum positive bending moment will be of smaller value. Maximum positive BM is the major factor in dimensioning the cross section of the beam (concrete and reinforcing steel).

$$Max.\,S.F. = \frac{w\,L}{2}$$

System of Continuous Beams on Columns

The slab is carried by secondary beams, which are carried by continuous beams. Continuous beams are supported by columns. The span of such a system is preferred to be less than 8 meters. A system of contiuous beams and columns takes advantage of the monolithic casting of concrete in developing negative clamping moments to offset the positive bending moments produced by loads. Lower design moments need less cross-sectional moment of inertia to withstand it, i.e., smaller sections. Continuous structures usually need less material and offer more space, which make them desirable. The price that has to be paid is the large-scale in-situ casting of concrete to ensure monolithic action, which sometimes becomes expensive and time-consuming.

Relationships:[4]

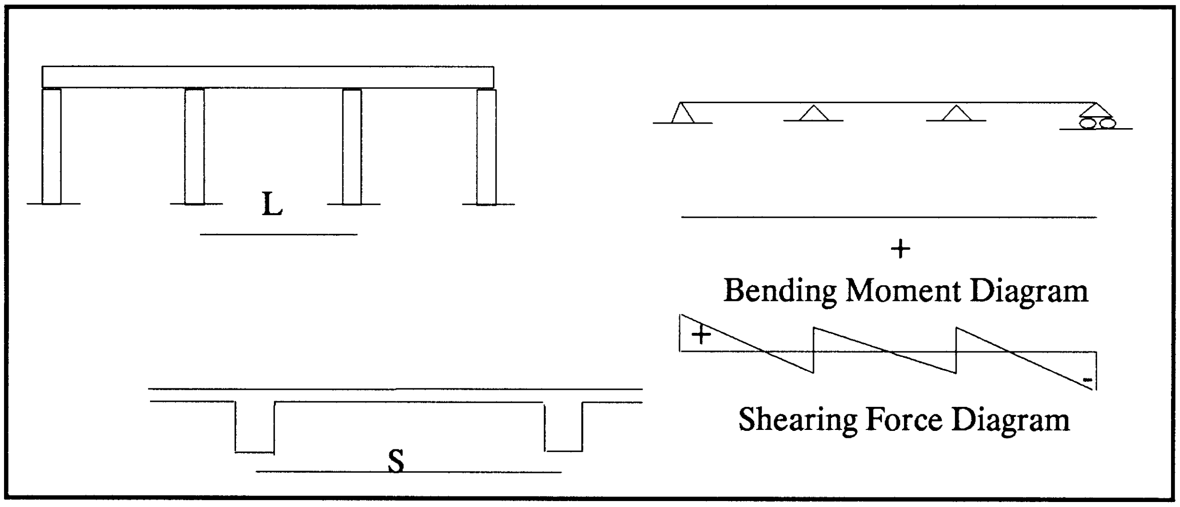


Figure (5.3): Continuous Beam & Column System

$$Max.\,+ive\,B.M. = \frac{w\,L^2}{f}$$:*where* $f =$ 10 for *outer spans*, 12 for *inner spans*.

$$Max.\,S.F. = 0.6\,w\,L$$

4 More relationships, of the same form, could have been added to express straining actions at the various locations, but that was not needed for the demonstrative purposes of the approach.

System of Girders on Columns

The slab is carried by cross girders (secondary beams), which are carried by the main girders. The secondary beam could be defined as a simple beam or a continuous beam, depending on the construction method. Treating cross girders as simple beams is a safe approximation. It is conventional wisdom that the span of beam and girder system should be less than 7. The system does not depend on monolithic action. Therefore, it is appropriate for pre-fabricated construction and flat slabs. It also is appropriate for heavy distributed loads (dead and live).

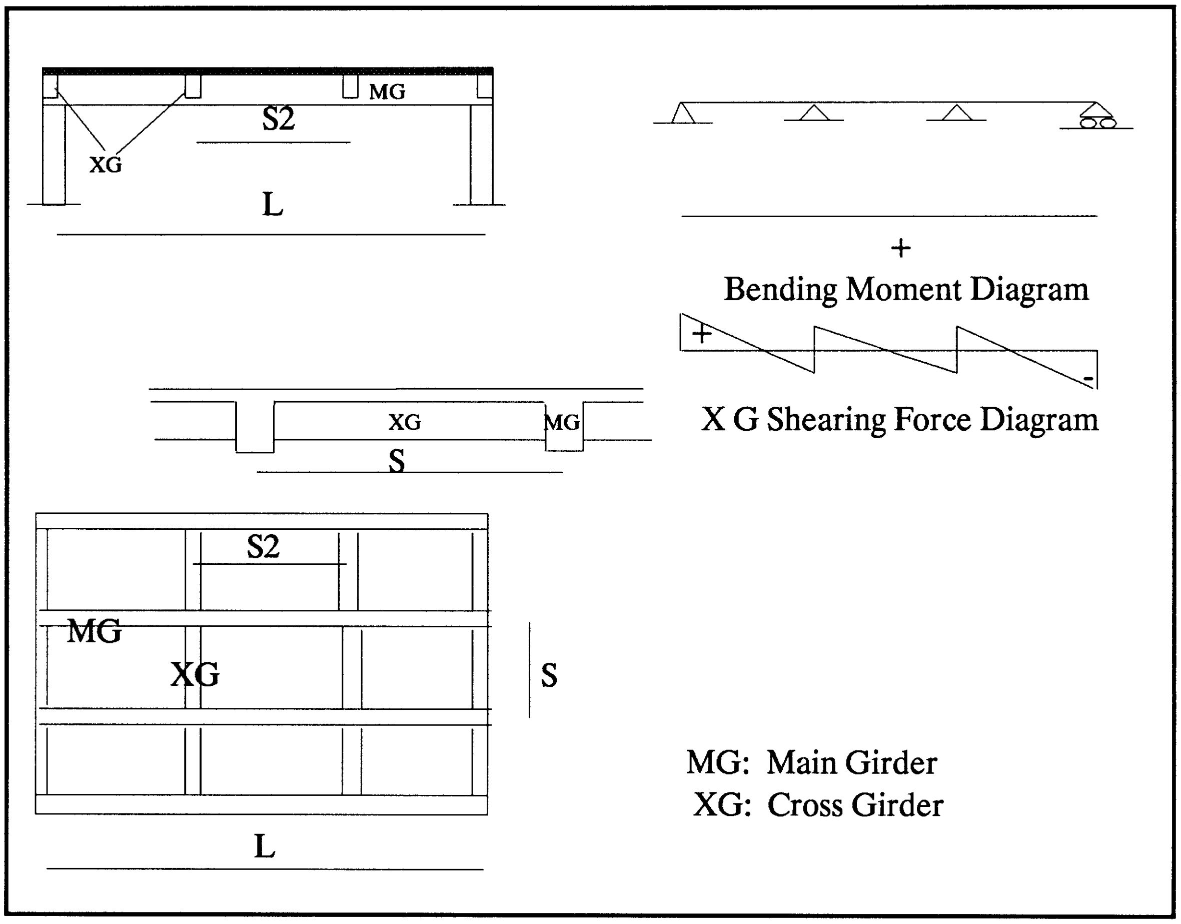


Figure (5.4): Girders & Columns System

Relationships:

Main girders are treated as simple beams with span of (L), with the approximation of considering floor distributed loads transfered directly through slabbing action, this skirts the problem of dealing with concentrated loads (from the cross girders).
Cross girders are designed as simple beams with span of (S).

System of Portal Frames

Portal frame is a structure in which the girder and the columns are made all one piece, that looks like a "gate"; hence the name. Load is transferred from the secondary MSS, which is either a slab or a slab supported by cross girders, to the portal frame, the primary MSS. Portal frames depend on monolithic action to develop clamping bending moments at the joints. Therefore, in-situ construction is needed for this main structural system. Conventional wisdom suggests that the span of a plain portal frame should be less than 12 meters. The secondary roofing structure (slab & beams) is treated as a virtual slab.

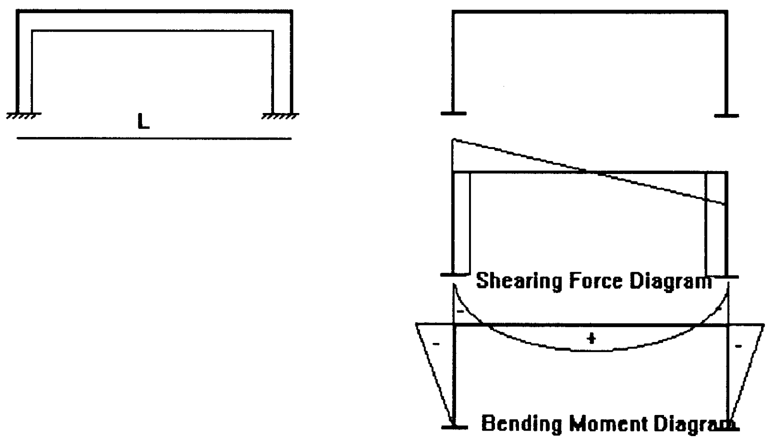


Figure (5-5): Illustration of Portal Frame System

Relationships:

The following calculations are the rough first-order analysis of the simplified case considered.

$$Max.\ +ive\ B.M. = \frac{wL^2}{12}$$

$$Max.\ -ive\ B.M. = \frac{wL^2}{16}$$

$$Max.\ S.F. = \frac{wL}{2}$$

See figure (7) for the constraint representation of these relationships.

System of Trusses

The following points demonstrate the structural need for trusses:

° 1. A long-span girder, supported from both ends, will develop high bending moments in the middle of it.

° 2. To withstand high moments, a cross section has to have a high moment of inertia. Depth of section is vital in determining the moment of inertia, I, of a section, e.g., $I = \frac{b\,d^3}{12}$ for a rectangular section.

° 3. These positive bending moments produce flexural tensile stresses in the lowermost fibers of its midspan, and flexural compressive stresses in the topmost fibers of its midspan.

° 4. Concrete is a brittle material that withstands practically no tension, reinforcing steel is introduced for this purpose.

° 5. Considering 3 & 4, large portions of concrete cross section are inactive, and could be carved into voids instead

Various shapes of voids are introduced to form different kinds of trusses that can span longer distances, such as 20 to 30 meters. Trusses have to be cast all in one piece, but they can possibly be fabricated away from their columns. Trusses are good for roofing an area that needs a lot of elevated duct work, piping, or ventilation.

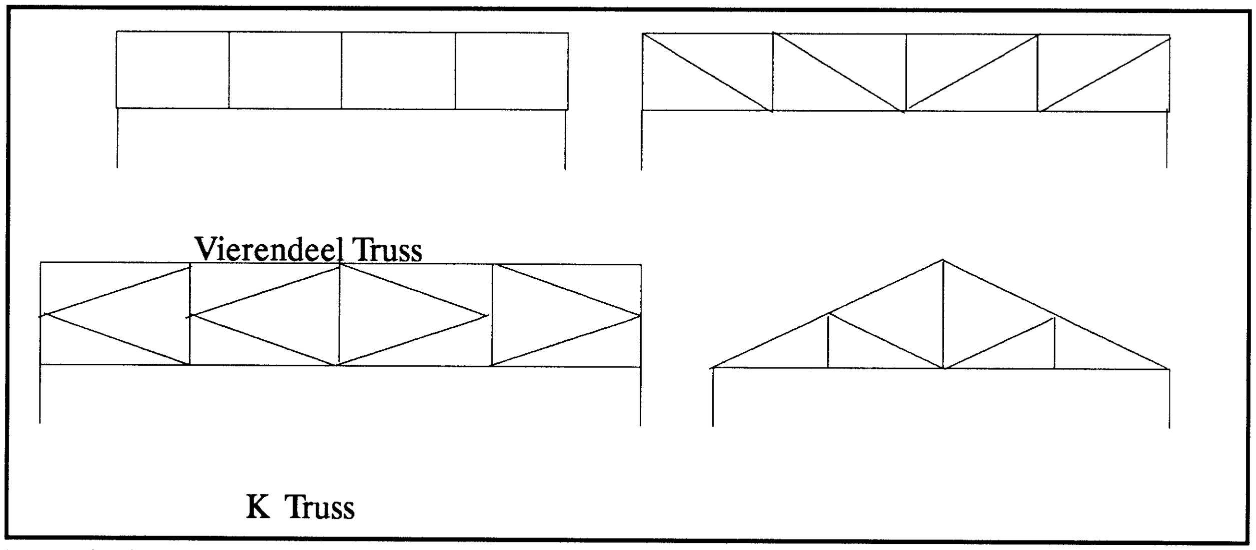


Figure (5-6): Illustrations of Various Trusses.

Relationships:

Forces in the truss members are to be determined by a truss analysis. Nevertheless, for the demonstrative purposes of the application, a Vierendeel truss is being used. In that truss the member design force equals the vertical reaction at the support.

System of Cylindrical Shells & Diaphragms

Cylindrical shells (vaults) are good for long spans (14-20 meters). To preserve the delicate cross section from deformation or side sway, diaphragms are introduced every 30 meters or so. The major disadvantage of cylindrical shells is the expensive construction. Cylindrical shells could be either pre-fabricated, or cast in-situ.

System of Double-curved shell:

The most efficient roofing structure is the egg, with no main and secondary systems for load transfer. Only a thin shell that transfers the load in three dimensions. Double-curved shells are the solution for spans of more than 30 meters. The problem is the expensive construction techniques needed for it. Double-curved shells include spherical domes, Hyperbolic Paraboloid shells, etc.

The system uses a spherical dome whenever it decides to use a double-curved shell.

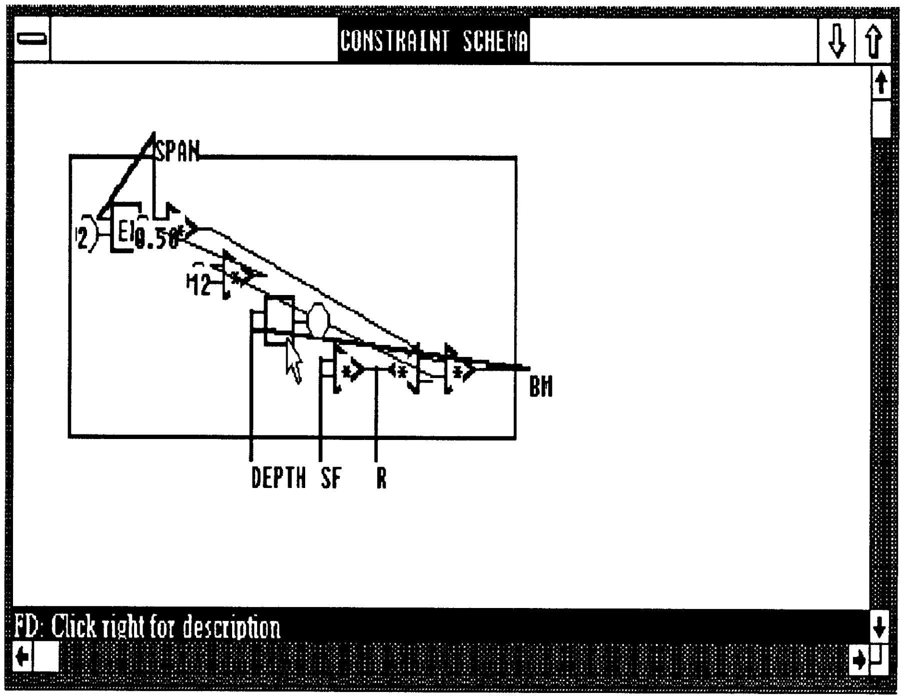


Figure (5.7): Constraint Represent. of a Portal Frame

5.1.3. Structural Design & Engineering Stages

The following are the steps of structural design & engineering, as illustrated in figure (5.1).

* The initial (conceptual) design of structural systems is made according to some heuristic rules (Helal 1980). System topography and dimensioning[5] of cross sections are needed for assessing the loads on the structural system. System topology is the selection of the main system and the selection of the secondary system.

* The loading combinations are used in the structural analysis. The structural analysis provides us with the distribution of straining actions, such as Normal forces, Shearing forces, and Bending moments, all over the system.

* Critical sections, with critical straining actions extracted from the analysis, are used for the design of the exact dimensioning of the cross sections. Design (dimensioning) of cross sections may reveal one of the following:

  ° No possible dimensioning could be made at a specific cross section. This is because, in order to keep the stresses and strains generated by the straining actions at that section within allowable margins, the dimensioning will be impossible (i.e., unrealistic[6]) because of **constructibility** considerations, inertia differentials, etc.

  ° The exact dimensioning is far from the initial heuristic dimensions. In this case, a second run of design has to be carried out using the exact dimensions of the first run as the initial dimensions for the second run.

* New specifications/constraints could be introduced to the system, which may necessitate a change in one parameter. That change will propagate throughout the system to reach the stable status again with a minimum[7] amount of changes, <u>even if it requires a change in the main structural system selected in</u>

---

5 Dimensioning of a cross section is the determination of both the (outer) dimensions of the reinforced concrete section, and the amount and arrangement of reinforcing steel in the section such that the stresses and strains in the sections are kept within the allowable limits.

6 Realistic dimensioning in itself is another field for heuristics. If the distribution of straining actions along the element is violently varying, we cannot design a structural element with too much sudden variations in the dimensions, or in the reinforcement to cope with force distribution. Instead, we design the element to accomodate the maximum straining actions happening all over the element. Such safe dimensioning is far from being parsimonious.

7 Minimum change is dictated by the design criteria (such as minimization of depth, maximization of stresses, etc.), and the conflict resolution rules.

the conceptual design stage. More description of propagation of changes is offered in "*Different Uses*" *section, p.* 109 .

* Problems that arise during construction should be used, later on, as corrective feedback to the conceptual design stage for similar projects.

* Problems that arise during the lifecycle of the structure should be fedback to both the construction stage and the conceptual design stage to be mitigated in similar projects later on.

The transition from structural analysis to dimensioning is a tricky one, as explained in the subsection on limitations, limitation 1 (p. 113).

5.1.4. Selection of Main Structural Systems

The selection process of a main structural system (MSS) is an example of the *selection methodology* outlined in chapter Three.
The system starts with an initial selection of an MSS. Initial selection is based on conventional wisdom among designers (Helal 80, Winter & Nilson 83). This pre-compiled knowledge is based on allowable stresses and strains of conventional loadings of plain structural systems. So, initial selection provides us with a good starting point in design. Detailed analysis on the same plain systems using the same conventional loadings will lead to the same results. Specific designs with different allowable stresses, strains, or loadings may lead to the dropping of the initially selected system, in favor of another system.

(0) Selection Criteria

The system uses the following criteria in making its initial selection of a main structural system:

Span: Information in this criterion is based on conventional wisdom among designers (Helal 80, Winter & Nilson 83). This pre-compiled knowledge is based on allowable stresses and strains of conventional loadings of plain structural systems.

Construction: Type of construction influence the selection of the main structural system. The major two types of construction are *in-situ casting* and *pre-fabricated construction*.

```
(defcon selector ()
 ;there is no advantage in packaging it into an mcon, so you can start from the
next line
    (cond4 cc) (global selected-system)
            (zand ( <! (global span) 5)
                  ( <! (global spacing) 4.9))
          (global SIMPLE-BEAM-token)
            (zor  ( <! (global span) 7)
                 ( =  (global construction)  (global pre-fab-token)))
            (global BEAM&GIRDER-token)
            (zand ( <! (global span) 8)
                  (zand    ( = (global construction)  (global IN-SITU-TOKEN))
                           ( <! (global spacing) 5.1)))
             (global CONTINUOUS-BEAM-token)
             (zand ( <! (global span) 12)
                   ( = (global construction)  (global IN-SITU-TOKEN)))
             (global FRAME-token)

            ...  ;; the rest of the main structural systems

             )

((selector xs1))
```
Figure (5-8): A Simplified Version of the Selector

Use: The purpose of the hall influence the selection of the MSS. Some purposes need ductwork, others need elegant image, a third may need both. The default use is ordinary use that has no special requirements.

(1) Initial Selection of Tokens:

* The selection criteria are composed into a constraint network, similar to the simplified one shown in figure (8).

```
((case4 c41) (global y) (global selected-system)
   (global simple-beam-token)
   ((simple-beam sb4) (global span) % Rsb SFsb BMsb)
   (global beam&girder-token)
   ((beam&girder bg4) (global span) % Rbg SFbg BMbg)
   (global continuous-beam-token)
   ((continuous-beam cb4) (global span) % Rcb SFcb BMcb)
   (global frame-token)
   ((frame fr4) (global span) % Rf SFf BMf)

   ...   ;; the rest of the parametric models

   )
```
Figure (5.9): Instantiating a Parametric Model

* Then the functional specifications, acquired through a pop-up window, are propagated through this initial selection network.

* The propagation of specifications leads to the selection of (equating the global variable *SELECTED-SYSTEM* to) a token that has the name of an MSS.

(II) Instantiating the Parametric Models:

* Instantiate the parametric models[8] of the various alternatives.

* Graphically display only the instantiation of the selected MSS.

(III) Severing link to Initial Selection

Disequate the global variable SELECTED-SYSTEM with the initial selection network, in preparation of propagating any functional specifications. Severing of the link is made to avoid collision between specifications and heuristics, since heuristics' role had expired.

8 Unfortunately, all alternatives have to be instantiated a priori, since the propagation mechanism, so far, has no notion of delayed or *conditional propagation*. More discussion of the topic is found in the last chapter on *Limitations*.

(disequate selected-system (the result cc))

(IV) Propagation of Specifications:

* Functional specifications, given by the user, are propagated through the parametric models of the various MSS's.

* Similarly, propagation is made for any later constraints imposed by the user, as shown in the example use given in subsection on *Different uses*.

5.1.5. Design of Sections

For the dimensioning stage, the system also knows how to design RC sections in two ways: under flexural moments, and under compression (Winter & Nilson 1985):

<u>**Design of Sections under Flexural Moment:**</u>

All the design equations are made using the Ultimate Strength Method and they are derived from the ACI code 1983 (Winter & Nilson 1985, Ch.4).

$$d^2 = \frac{M}{\varphi \rho f_y b \ (1 - (0.59 \rho \frac{f_y}{f_c}))}$$

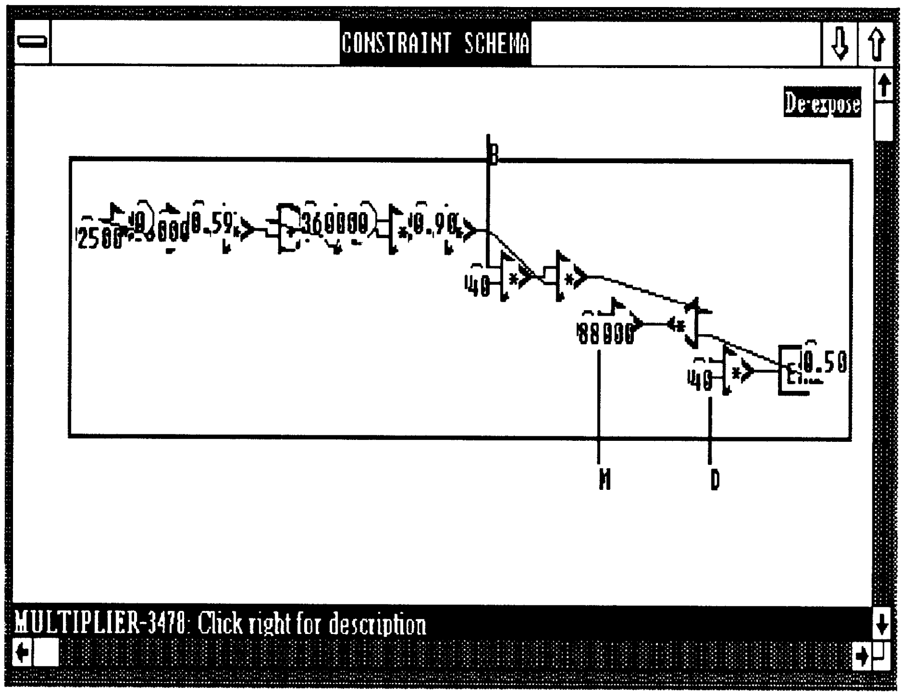
where:

b: is the breadth of the section, in inches.
d: is the (active) depth of the cross section, in inches.
M: is the flexural moment working perpendicular to the section, lb.in
f_c: 28-days Compression Strength of Concrete, psi.
(defaults to 2,500 t/m2 = 3,500 psi)
f_y:Tensile strength of Steel, psi.
(defaults to 36,000 t/m2 = 50,000 psi)
φ: Bending reduction factor, defaults to 0.9.
ρ: Minimum reinforcement ratio ($= \frac{200}{f_y}$ for flexural sections)
$= 0.05$ for compression sections such as columns
fs_{column} : reduction factor for tied columns, defaults to 0.7

The constraint representation of the procedure is demonstrated in figure (11), while the schematic illustration of the same procedure is shown in figure (10).

```
;;; Flexural design of sections:
;;;  Units: b (m), d (m), M (ton.m)
(evaluate-input
 '(defcon  design  (b d M)
    ((exp exp1) (* d 40)                          ; 1 m    =   40  in
        ((* mult1) (* M 88000) %                   ; 1 t.m = 88000 lb.in
              (* (*  b  40) (* (global fi)
                          (*  (global  ro)
                           (*   (global   fy)
                              (+ 1 %
                                (*  0.59
                                  (* (global ro)
                                   (*   (global  fy)
                                     %   (global   fc))) )))))))
       0.5)))
```

Figure (5.11): Flexural Design Macro Constraint.



MULTIPLIER-3478: Click right for description

Figure (5.10): Representation of Flexural Design of Section

To dimension a section, called dd1, that has a breadth of 0.25 meters, and is subject to bending moment of 100 ton.meter, we need to invoke the flexural design macro constraint that will calculate the depth of the section and the tension steel reinforcement.

```
((design dd1)  0.25  dx1  1e5)
```

Design of Sections under Compression:

(Winter & Nilson 1985, Ch.6)

A tied column fails at the load Pn

$P_n = 0.85 f_c A_c + f_y A_s$;immutable because **d** shows up in both terms

$= A_c (0.85 f_c + f_y \rho_{column})$

The design load Pd is:

$P_d = f_s P_n$

where:

fs : is the factor of safety
= 0.70 for tied members
0.75 for spirally reinforced members.

The constraint that represents the procedure is illustrated in figure (12).

```
'(defcon compression-design (b d N)
  ((* PDMULT) N  (global fs-column)
              ((* PNMULT)  ((*  m1)  b  d)
                                         ((+ a1)  ((* m2)  0.85  (global fc))
                                         ((* m3) (global  fy)
                                              (global ro-column))))))
```

Figure (5.12): Design Constraint of Compression Sections.Z_LABEL FIG

Design of section under compression with eccentricity:

It is done through the simultaneous fulfillment of the two previous cases. These two cases cover most of the general design cases of reinforced concrete sections.

5.1.6. Knowledge representation

In this section, we discuss the objects that were needed for the customization of the system to be applied in the area of conceptual design of main structural systems of R.C. exhibition halls.

Hall:

It is a macro constraint created to be a wrap around the instantiation of the macro constraint that represents the selected main structural system. *The conceptof a wrap-around* was needed so the functional specifications are applied only once to the structure at general, without worrying about what alternative MSS is selected at what time.

The pins of the macro constraint are divided into three categories:

- ° **Specification pins:** width, depth, height, breadth, max-bay
- ° **Variable pins:** span, spacing
- ° **Internal constraints:** main-system, secondary-system.

Project:

It is the data structure that holds all the information needed to be known about a project to ba saved as a case. The **project** data structure is affiliated to the **HALL** macro constraint that represents the topmost specifications of the project, such as outer dimensions, girder depths, column depth, etc. This affiliation is made to attach book-keeping administrative information as shown in the attributes later on.

The object inherits the **case** data structure defined in chapter 4 on *Implementation*.

The following are the attributes of a Project:

name: unique name of the project

mcon: the **HALL** macro constraint that represents the project. Its default name is the project name prefixed by "hall-".

elevation: name of the window pane where the elevation view will be displayed.

plan: name of the window pane where the plan view will be displayed.

mouse-line: name of the window pane where the mouse-sensitive messages will be displayed.

dialog: name of the pop-up user interaction window.

icon-window: name of the window pane where icons representing the major system functions are displayed.

scale: scale of the drawing, pixels/meter

structures: The top-level structures that comprise the project.

__structure:__

It is the data structure the represents the structure to be designed. It is affiliated to the macro constraint of the selected MSS for some book-keeping purposes.

The data structure has the following attributes:

name: The unique name of the structure.

mcon: the macro constraint that represent the structure. Its default name is the structure name prefixed by the type of the structural PORTAL-FRAME-CCRE.

width: The (facade) width of the plot of land to be roofed, in meters.

depth: The other (secondary) dimension of the plot of land to be roofed, in meters.

height: The clear (internal) height of the bottom of the roof, in meters.

min-required-bay: The minimum span[9] in the major (facade) dimension, in meters.

x0, y0, z0: The pixel coordinates of the upper left corner of the drawing.

5.1.7. Different Uses

Figures (13) through (14) demonstrate one possible way for using the system.

* First, the system query the user about the functional specifications of the hall, through a pop-up window. Specifications include dimensions of the hall, the use of the hall, and construction method.

* The system creates the selection constraint network, similar to the one in figure (8). The network is used for selecting the appropriate main structural system.

* The parametric model of the selected main structural system is instantiated according to a detailed-design selector network similar to the one in figure (9).

* The propagation of the user's functional specifications through the parametric model, leads to a detailed design, that gets graphically displayed as in figure (13).

9 Span is the distance between to adjacent columns, centerline to centerline.

* The user can make a modification to the design, either textually by typing in a specification (constraint) to the constraint listener pane, or he can graphically do that by clicking upon the icon that represents a parameter. Figure (15) shows what happens when the user clicked upon one dimension, the *minimum-bay-width* in this case. A pop-up window prompts the user to enter the new value, the user in this case asks for column-free facade.
The system synthesizes a constraint **(= = minimum-bay-width project.width)**, that represent the modification, and propagates it.

* The propagation of a modification may trigger a conflict. The system was designed such that all conflicts will be brought up to the attention of the user so he can decide on the way to resolve it.

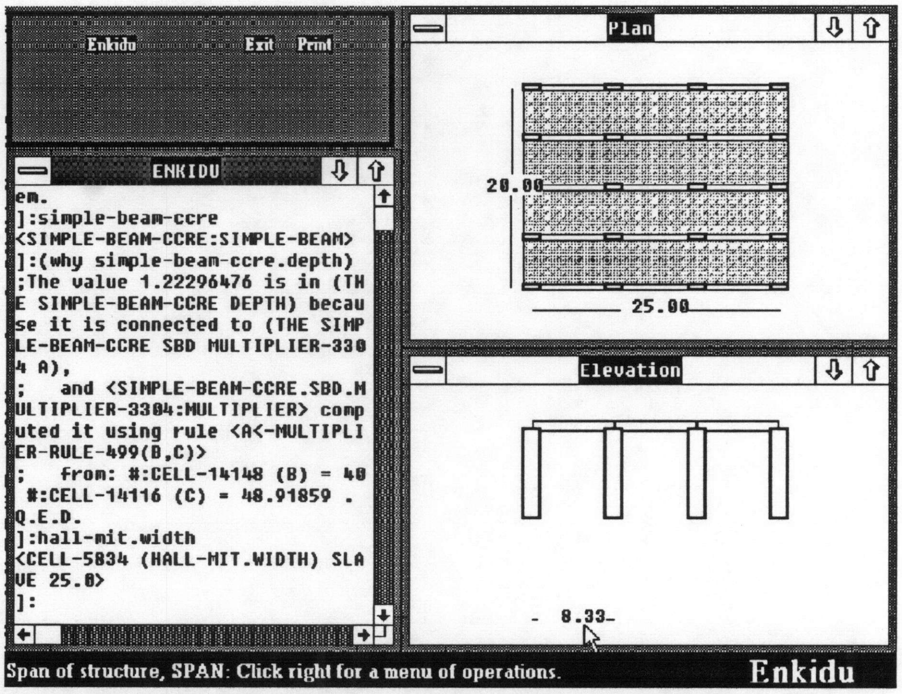
* After resolving any conflicts, and completing propagation, the graphical representation of the new design is displayed, as in figure (14). That figure shows that the modification to the functional specification prompted drastic change in the design, that amounted to selecting a different main structural system.

5.1.7.1 Examples of Other Questions that the System Can Answer

* **Change the kind of reinforcing steel used.** The system then uses the new yielding stress of that steel all over the design. This approach is frequently used by designers to take more flexural stresses in shallower beam depths. Traditionally they have to either recalculate the whole design, or upgrade the steel all over the structure to meet the critical stresses at few sections, which means a lot of waste of materials.

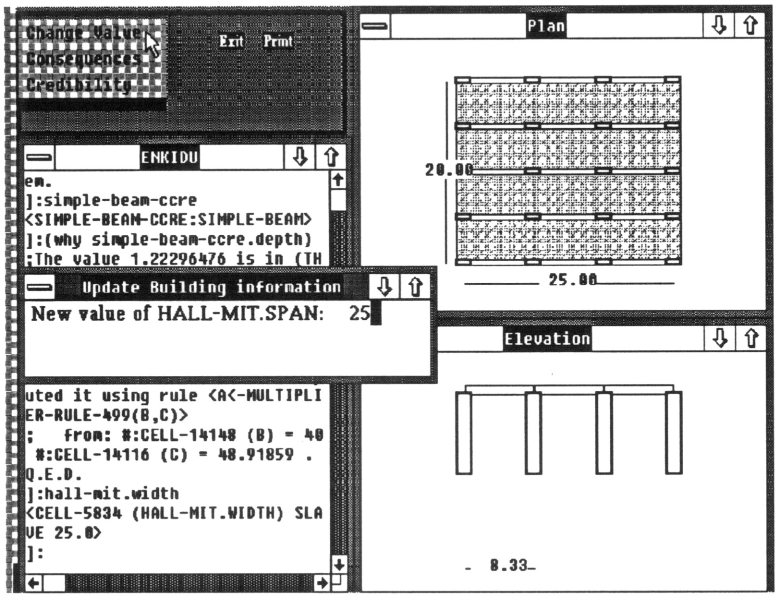
* **Change the maximum allowed width of a crack:** In a water-tight design, say of a tank, it is a frequently-discussed issue to question the maximum allowed width of a crack, its rationale, and its impact upon the design. To scrutinize the various views, the system allows the user to change the maximum allowed width of a crack, and the system propagates it all over the design. The propagation may trigger drastic changes in the design.

* **Assign a cost function for every main structural system.** The user can post whatever function to represent anything, such as cost or time. Every time a change is made in the design, the function gets recalculated automatically. Of course, threshoulds, blows and whistles can be attached to monitor/control these functions.


Enkidu
Exit
Print
ENKIDU
Plan
20.00
25.00
Elevation
8.33
Span of structure, SPAN: Click right for a menu of operations.
Enkidu

```
em.
]:simple-beam-ccre
<SIMPLE-BEAM-CCRE:SIMPLE-BEAM>
]:(why simple-beam-ccre.depth)
;The value 1.22296476 is in (TH
E SIMPLE-BEAM-CCRE DEPTH) becau
se it is connected to (THE SIMP
LE-BEAM-CCRE SBD MULTIPLIER-330
4 A),
;   and <SIMPLE-BEAM-CCRE.SBD.M
ULTIPLIER-3304:MULTIPLIER> comp
uted it using rule <A<-MULTIPLI
ER-RULE-499(B,C)>
;   from: #:CELL-14148 (B) = 40
 #:CELL-14116 (C) = 48.91859 .
Q.E.D.
]:hall-mit.width
<CELL-5834 (HALL-MIT.WIDTH) SLA
VE 25.0>
]:
```

Figure (5.13): The design made by System, from Textual Input

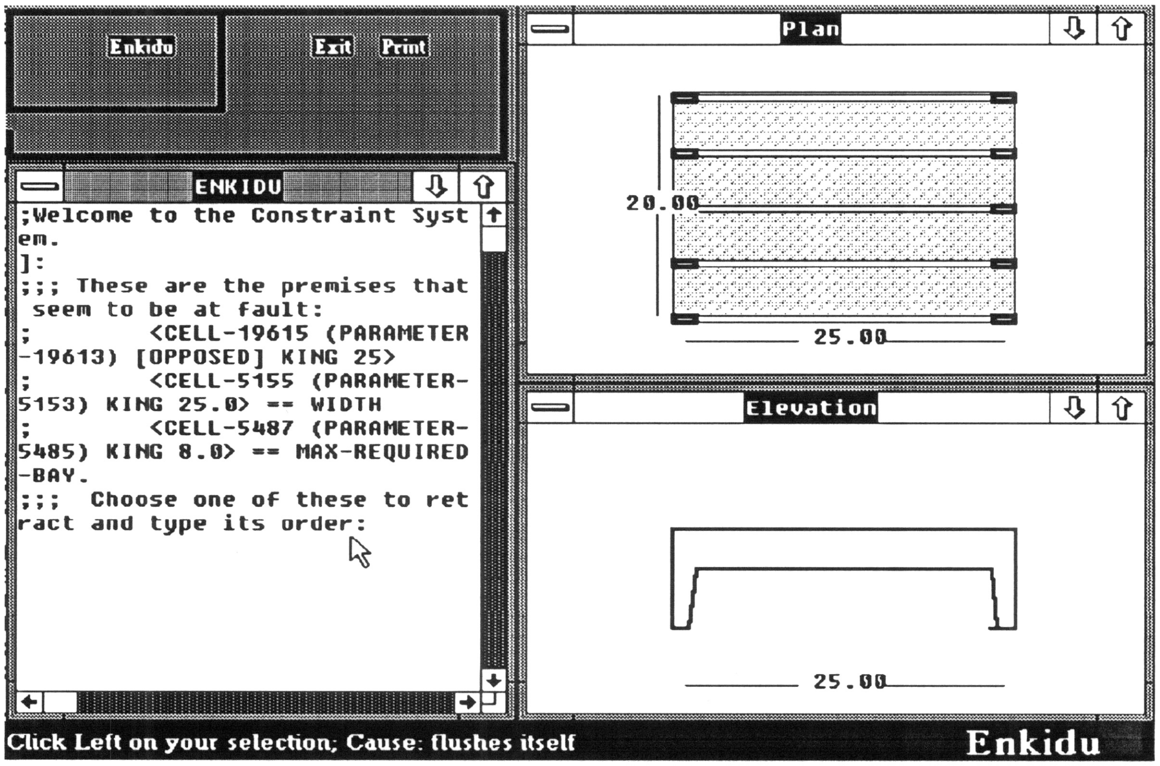


```
em.
]:simple-beam-ccre
<SIMPLE-BEAM-CCRE:SIMPLE-BEAM>
]:(why simple-beam-ccre.depth)
;The value 1.22296476 is in (TH
```

New value of HALL-MIT.SPAN: 25

```
uted it using rule <A<-MULTIPLI
ER-RULE-499(B,C)>
;   from: #:CELL-14148 (B) = 40
 #:CELL-14116 (C) = 48.91859 .
Q.E.D.
]:hall-mit.width
<CELL-5834 (HALL-MIT.WIDTH) SLA
VE 25.0>
]:
```

Figure (5.15): The User Clicks upon a Dims. to Change.



```
;Welcome to the Constraint Syst
em.
]:
;;; These are the premises that
 seem to be at fault:
;       <CELL-19615 (PARAMETER
-19613) [OPPOSED] KING 25>
;       <CELL-5155 (PARAMETER-
5153) KING 25.0> == WIDTH
;       <CELL-5487 (PARAMETER-
5485) KING 8.0> == MAX-REQUIRED
-BAY.
;;;  Choose one of these to ret
ract and type its order:
```

Click Left on your selection; Cause: flushes itself

Figure (5.14): System Propagates the Change, Reaches New Design

5.1.8. Limitations

1. The current models do not allow for imposing concentrated loads and moments at random locations.
   A possible solution could be made by devising a list of straining actions (loads & moments) that work on the structure. Then a different kind of constraint, that takes into consideration load value, coordinates, and type of straining action, needs to be devised. Such a constraint could be an empty one that only has pins extending out of it, without a processing body. The constraint is made just for clumping values together.
   (defprim (concentrated-load P) (val direction x y z))
   :where direction is the angle, in centigrades, measured clockwise, from gravitational downward.

   (defprim (concentrated-moment M) (value direction x y z))
   :where direction is either 0 for clockwise, or 1 for counterclockwise.

2. Due to the infinite combinations of loads the sections with critical straining actions may appear anywhere all over the structure. While we can make reasonable guesses for critical sections in traditional loading cases, there is no way to make such a guess for a generalized loading case.
   A possible solution could be in saving a symbolic function, e.g., binomial series, as the value for every kind of straining action for every structural element. Then a **differential** constraint will take the fuction and give the maximum and minimum locations. Another constraint should determine all straining actions at a specific section.

3. This application does not give the user the freedom to select support types, e.g., roller, hinge, fixed, etc. The system instead selects supports such that the system will be statically determinate. The application had to be made like that to avoid worrying about analysis of structurally indeterminate systems. Such analysis should be made by one of the standard Finite Element Analysis systems to which the application will be hooked up.

4. This application assumes the secondary structural system to consist always of simple beams.
   The **solution** is to repeat the steps for selecting the Main Structural System (MSS), both initially and finally, for the selection of the Secondary Structural System (SSS).
   This solution was not implemented since it demonstrates no more capabilities of the system , while it slows down the performance, especially on a machine with limited memory.

5.2. Design Checking & Optimization (Correction) of Floor Plans[10]

This example demonstrates the application of the system to the are of checking the compliance of a floor plan of a hospital to a body of standards. The standards included chapter 5 of the National Fire Prevention Association (NFPA) code, and New York State - Office of Mental Health's Life safety code.

Basically, what the system does is taking a floor plan (on a CAD file), and generating both a copy of the same drawing with all the violations flagged out on it, and a detailed violation report. Current work is being done to utilize the violations in suggesting improvements (global solutions) for the initial design.

The formal name of the project is Code Orientation, Review, Enforcement, Commissioning, and Testing (CORECT).

5.2.0.1 Evolution:

° The design synthesis experiment, which demonstrated the inter-relationship between dimensioning & conceptual design, showed the need to propagate changes, i.e., to be **parametric**.

° Every change, made to the hall design, is perceived as an additional specification.

° That led to the thought "*what if we need to know the compatibility of a specification , without enforcing it?*" In other words, there was a need to have a mild conflict that can be noticed and carry on checking other things.

° That brought up the idea of the checking stage in the engineering life cycle, discussed in fig.(1).

° New York State expressed their need for automating the checking stage of the lifecycle. especially for the **National Fire Prevention Association code (NFPA)**.

° It was a good chance to implement/test an additional module of the system environment , that led to look, again at the engineering lifecycle and to try to come up with one environment for design problem solving. The process contributed positively to the design of the Engineering Knowledge-integrated Design Utilities.

° After winning the national award, NYS code enforcement agencies said "It's nice of the program to tell us about the violations; but wouldn't it be nicer if it can suggest to us how to fix these violations?" Therefore Phase (II) of the project was started to do that.

° So, design synthesis evolved into design checking, which in turn, evolved into design synthesis again. That proves the concurrent engineering nature of the whole domain of design.

10 This system was selected by the National Office of Government Technology, Washington, DC. as the **Innovative Solution of the Year 1989.**

5.2.0.2 Input:

The chief engineer has to input the standards (code) to be enforced only once, in the syntax of the system Specifications, in a text file, as shown in figure 5-16 .
Everytime a design is submitted for approval, a CAD file of it is passed to the design checker.

5.2.0.3 Output:

1. A copy of the input drawing, with all the violations flagged out, and
2. A detailed violations report that describe every violation flag in the drawing.

```
(gen-article  'OMH-SC&NC 1
    "Bedroom Requirements:"

    "Skilled Nursing Facility (SNF)shall be provided a minimum of 125
     square  feet plus a 6 square foot closet in single bedrooms and 100
     square feet   plus a 6 square foot closet for each patient in double
     bedrooms.  No more than four will occupy a bedroom but, there
     shall be a minimum of   66.6% single & double bedrooms per ward."


    '(and (room-p object)
            (equal (room-type object) 'bedroom)
            (equal (ward-type (room-ward object)) 'SNF))

   '(and (cond ((= (room-capacity object)  1)
                    (= (room-area object)    125)
                    (= (room-closet-area object)   6))
                   ((= (room-capacity object)   2)
                    (= (/ (room-area object) (room-capacity object))  100)
                   (= (/ (room-closet-area object) (room-capacity object))
                        6)))
           (< (room-capacity object) 4)
        ;;66.6%          .
        ..
        ))
```

Figure (5.16): An Article of NYS-OMH Space Conditioning Code

5.2.1. Implementation

The system is written in system specifications, and Lisp. The interface to the CAD package is written in Fortran. Checking is made through unification of the specification with every suitable object in the part database of the CAD file, as described in 80. Suitability is determined through a *filter predicate* as shown in figure 5- 16.

Figure 5- 16 demonstrates an article (specification) of the code. The constraint is a plain Lisp code, while Figure 5-17 demonstrates the same article declared as a Constraint.

```
(gen-article  'OMH-SC&NC 1
   "Bedroom Requirements:"

   "Skilled Nursing Facility (SNF)shall be provided a minimum of 125
    square  feet plus a 6 square foot closet in single bedrooms and 100
    square feet   plus a 6 square foot closet for each patient in double
    bedrooms.  No more than four will occupy a bedroom but, there
    shall be a minimum of   66.6% single & double bedrooms per ward."


   '(and (room-p object)
         (equal (room-type object) 'bedroom)
         (equal (ward-type (room-ward object)) 'SNF))

  '(and (cond3   result1
              ((= room-XX.capacity 1)
               (and (< =  125  room-XX.area)
                    (< =  6   room-XX.closet-area )))
              ((= room-XX.capacity 2)
               (and (< =  100  (* room-XX.area  % room-XX.capacity))
                   (< = 6 (* room-XX.closet-area % room-XX.capacity)
                    )))
               (truth (< room-XX.capacity 5))).
    ;;66.6%
    ..
```

Figure (5.17): The Same Article of NYS-OMH as a Constraint.

5.2.2. Knowledge Representation

The following is a description of the objects of the domain of AEC design of hospitals.

Architectural-Closure:
This object is created merely for inheritance purposes.
arch-elements

Architectural-Element:
Yet another object created for inheritance purposes.

rating:

finish: It is a list of lists. Each of the inner lists has a finishing material and the specification needed to quantify its contribution to the fire rating. The cumulative fire rating of all inner lists is the value to be stored in the abovementioned fire rating attribute of the element.

Building:
It inherits the object structure, in chapter 3, p. 56.
This conceptually is the largest object in this customization.

name	existing?
type	components

Floor:

```
id                          name
building
(type 'patient-floor)     ;repititive/patient, non-patient, mechanical
(Components '()))
```

Bexit:[11]
Building exit.

name	building
floors	doors
corners	

11 Some implementations of Lisp and C++ will not allow defining an object with the name **exit**.

Ward:

```
id                              name
class                           (capacity 0)
floor
(type 'general)       ;secure, administration, general/psychiatric, SNF
(number-of-bathrooms 0)         fixtures
(components '())
```

Corridor:
It inherits **ARCHITECTURAL-CLOSURE.**

```
name                            id
width                           exits
smoke-barriers
height                          (rooms-opening-to-it '())
(doors-opening-into-it '())     windows
finishes  (for wall, ceiling and  floor)
ward                            corners
```

Door:
It inherits **ARCHITECTURAL-ELEMENT.**

```
id                              name
room                            corridor
rating                          swinging-direction
(self-closing?  nil)            (tight-fitting? nil)
glazing-type                    side-clearance
mid-coordinates                 closer
latch                           hinge
hinge-coordinates               composition
type     ;swing-, revolving-, turnsile-, balance-, ...
usage    ;'room-, 'fire-stop-, 'exit-door
exit                            smoke-barrier
size
```

Room:
It inherits **ARCHITECTURAL-CLOSURE.**

```
name                            id
(capacity 0)                    area
type     ;'patient-, day-, quiet-, program-, recreation-, visitors-, dining-room
(window '())                    corridor
doors                           (closet-area 0)
furniture                       finishes
```

ward	(corners '())
(attached-bathroom-fixtures nil)	

Bathroom:
It inherits **ROOM.**

name	id
area	(number-of-showers 0)
(number-of-washing-basins 0)	(number-of-toilets 0)
(number-of-lavatories 0)	
(number-of-tubs 0)	ward

Window:
It inherits **ARCHITECTURAL-ELEMENT**.

id	room
area	glazing-type

Fixture:
It inherits **ARCHITECTURAL-ELEMENT**.

id	type
room	coordinates

Wall:
It inherits **ARCHITECTURAL-ELEMENT**.

name
closure

Ceiling:
It inherits **ARCHITECTURAL-ELEMENT**.

name	closure
height	projection

Material:
The data structure for building material, e.g., finishing (painting, tiling, etc.). The only attribute we care to know about, for the purpose of checking the compliance to the NFPA code, are:

name	flame-spread

5.2.3. Results

The system developed in this section was used to check the compliance of hospital floor plans to both chapter Five, titled *egress*, of the National Fire Prevention Association (NFPA) code 101, and the New York State OMH Space Conditioning & New Construction program (NYS-OMH-SC&NC). Figure (5.18), in the following page, demonstrates a part of the output drawing generated by the system integrated with Medusa CAD package as a result of imposing NYS-OMH-SC&NC. The input drawing looked exactly the same, without the violation icons which are plotted on a separate sheet.

Figure (5.19), in p. 122 , demonstrates parts of the violation report generated when checking the same design.

5.2.4. Impact on the Checking Process

1. The system allows the code permitting authorities to be more even-handed with various designs, competing or not.

2. The system encourages the code permitting authorities to be more strict in enforcing the standards. To avoid the heavy-handedness of the system, two measures were added to it:
   - ° The system allows the user to turn off any number of articles of the code. The output, though, will show the article numbers in the corner of the drawing.
   - ° The user can set the equality threshold, such as 1% or 43%, after which a violation is triggered.

3. The encoding process into the system encourages the code permitting authorities to rewrite the standards in a more precise unequivocal language.

4. The system can be used in monitoring the performance of the various designers, and to guide them in overriding their weaknesses.

5. Since the motivation of such a system is reaching better designs rather than punishing designers, the system offers an interactive version of the standards, which can be used by the designers to check their designs before submission to the owner.

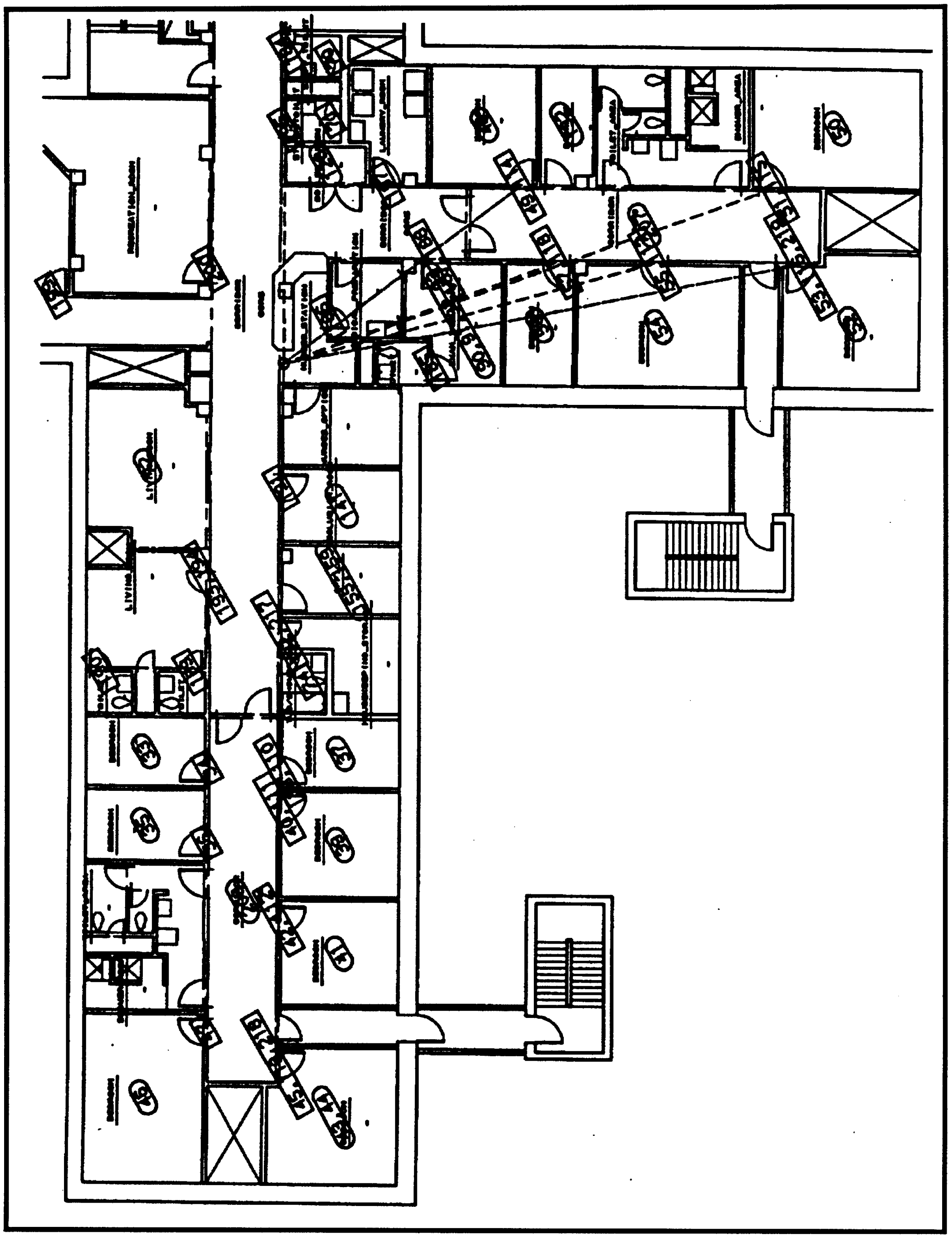


Figure (5.18): Output of Checker on a Floor Plan of Hospital

```
Reference  49:
ward class: PSYCHIATRIC       type: LONG-TERM-CARE     name: West
room type: BEDROOM                              number: br-3124
door id: 256
  door is too narrow for Psych. ward.  REQUIRED: 40.00 inches.  ACTUAL: 36.00 inches.

Reference  52:
ward class: PSYCHIATRIC       type: LONG-TERM-CARE     name: West
room type: BEDROOM                              number: br:3203
 Insufficient number of CLOSETS in room.  REQUIRED: 3, ACTUAL: 0.

**** Article OMH-NC&SC 11:  Line of Sight ********************************
Reference  114:
ward class: PSYCHIATRIC       type: LONG-TERM-CARE     name: West
room type: BEDROOM                              number: br-3085
door id: 256
  Door is not visible from NURSES' STATION.

**** Article OMH-NC&SC 13: Seclusion Rooms ********************************
Reference  127:
ward class: PSYCHIATRIC       type: ADMISSION     name: B
room type: SECLUSION-ROOM  number: 3012
  Seclusion-room's door opens to corridor COR1.

Reference  143:
ward class: PSYCHIATRIC       type:  LONG-TERM-CARE     name: West
room type: OFFICE                               number: 2157
  Office is not adjacent to a bedroom, as required.

**** Article OMH-NC&SC 17:  Latch Side Clearance of Doors **************************
Reference  185:
ward class: PSYCHIATRIC       type: LONG-TERM-CARE     name: West
room type: TOILET                               number: 3102
door id: 270
 Door's latch side clearance is too narrow on pull side. REQUIRED: 18.0 inches, ACTUAL:
17.28
```

Figure (5.19): Part of Violation Report of Demonstrated Floor Plan

5.2.5. A Specification-driven Approach to Facilities Management

The example of checking floor plans demonstrates the suitability of the system for the domain of facilities management. Similar to other domains, the validity of the system in checking the compliance of existing AEC designs to standards is demonstrated in the first phase of the project, and its validity in design synthesis is demonstrated in the second phase which is currently experimental. Furthermore, the system can be applied the following areas of facilities management:

5.2.5.1 Condition Survey of an Existing Facility

Condition Survey of an existing facility in preparation for rehabilitation. Causal (functional) relationships, in the knowledge base, are vital for for establishing a model-based reasoning capability that can guide surveyor to more precise, efficient survey of the condition of the facility. Stacking of cases can lead to finessing the causal model (el-Shafei 86).

5.2.5.2 Automatic Generation of Work Orders

Upon telling the system that employee X will move from room A to room B, the system will generate all the required work orders to move his telephone line, terminal/workstation, personal printer, personal fax, change the ethernet, update the local area network and consequently modify the circuitry of that part of the floor, if needed, to accomodate the change in power load. The same thing should happen to the HVAC controls to accomodate the change in HVAC load.

5.2.5.3 Survivability System

An important facility, such as an airport, highway, power plant, etc., should have a *survivability system*, that will enable the its management to perform under adversary conditions.

The major goal of such a system is minimizing the time needed to bring the facility back to functioning. To meet that goal, the system should integrate Geographic Information System (GIS), Facilities management, Control system, Scheduling/Planning, and a decision support system. The system should allow the facility survivability center to monitor on, almost, a real-time basis the status of all facilities and capabilities around the facility, such as runways, hangars, residential quarters, personnel, fire protection systems, Petroleum Oil Lubricants (POL) lines if we are dealing with an air base. As a facility gets damaged, the system will suggest a possible way of fixing the damage given the available resources. The user can decide an overriding solution though.
Information is to be input to the system through icons and menus. The system should take care of preventive maintenance. If a facility is damaged, not only that

the system will suggest which combination of personnel and machinery should fix it, it will also assign a damage assessment team to survey the other facilities that might have been affected because of adjacency (like POL lines), or structural inter-dependence (like the sub-base of pavement).

5.2.5.4 Knowledge-Based Diagnostic System

The system can have a diagnostic system which will offer both diagnosis and remedial action to every possible malfunction that may happen. No such system can ever be fool-proof, because of the ever changing technology among others. Therefore, the system has to be able to learn from the human maintenance engineer, using both previous cases and failure scenarios/mechanisms.

5.3. Other Applications to Complete the SDD Environment

Over the last two years, several applications has been developed, that made the system more robust and versatile. This versatility makes the system candidate to be a *Concurrent engineering* environment. Concurrent engineering is an approach that perceives the engineering of a domain as a continuum. In that continuum, every stage of the engineering process influence the other stages, in all directions, i.e., in a non-directional way. Therefore, the best representation for the problem is constraint propagation paradigm. So far concurrent engineering is more of a creed rather than a real implementation. The system, presented in this dessertation, is a serious trial to flesh out the creed.

In the rest of the section, several applications will be mentioned briefly, and how did they contribute to the development of a more robust environment.

5.3.1. Design of RF Connectors

There are three distinct teams who work on the design of RF connectors: Electrical analysts, electrical designers, and prototype modelers. Each of these teams has its own language for desin:

Electrical analysts: they deal with mathematical models. Their work ends by the development of a mathematical model.

Electrical designers: they take a mathematical model and flesh it out into a geometry, according to their experience, and some standards. So, their language is CAD systems.

Prototype modelers: they take a CAD file of an RF connector, and they develop worst-case logic diagram models to test the new connector.

So, the three teams speak three different languages, mathematical equations, CAD, and schematics. Indeed, constraints and backgrounds of each team are different. To develop good designs, the three teams have to communicate simultaneously rather than handling design sequentially, one team after another.

The solution was to perceive the product (RF connector) in a trimorphic representation: mathematical representation, CAD representation, and schematic representation. Every team can introduce a change, or a constraint in one language, the other two representations (languages) get generated automatically. Such representation was possible through a constraint modeling of the problem.

5.3.2. Design of Wheel Assembly of Industrial Fans

The problem: the *design-and-Quote* process for industrial fans. Industrial fans are giant ventilation devices of a discharge in the order of 1 million cubic foot per minute used in ventilating power plants, mines, etc. The manufacturer has 29 different families of industrial fans. To participate in a bid, the manufacturer has to fairly design the needed fan according to the functional specifications to be able to give a realistic quotation. This process is called design-and-quote. It is usual for the manufacturer after he turns in the quotation to find out that a new change in the functional specifications was made, and that design has to be made from scratch again. It used to take about 2 months for the manufacturer to go through this process. With the introduction of CAD systems the design-and-quote process was reduced down to 2 weeks, but that is not sufficient anymore with the fierce competition, and the more demanding specifications.

Solution: The solution was made of two parts:

First part is a selector part, similar to a production rule-based expert system that selects the appropriate family of fans to be used.

Second part was developing a parametric model for every family of fans. Once a family of fans was selected, its parametric model is instantiated (dimensioned). Whenever a change (constraint) is introduced to the system (textually, or graphically), it is propagated all over the design. The change may trigger just small changes around the fan, or it may invoke a drastic change such as adapting different family of fans. Engineering drawings are automatically generated by the end of design. Quotation can be automatically generated as function of the parametric model.
The system was integrated with Computervision's CADDS 4X, so the user can invoke the designer application either as a stand-alone application, or from within CADDS 4X.

Result: The solution reduced the design-and-quote process to less than 2 days.

5.3.3. Intelligent Fastening Joint Designer

Problem: One item on the highly publicized government procurement list, in 1988, was the $5,000 bolt. The bolt was manufactured by a major contractor, that manufactures a major gadget. The manufacturer clarified that a bolt may cost that much if it was one-of-a-kind bolt. Both the government and the contractor have their rules and inventories that should be followed. The problem is how to make sure that the mechanical designer will follow these rules. Instead of using the inventory catalog, a designer may opt to design a custom bolt for a specific task. The custom manufacturing of such a bolt may exceed $5000. More important is the cost of maintaining this new class of bolts. It costs about $3,000 to maintain (refasten, grease, etc.) every category of bolts per year, in everyone of the gadgets. So, in a product with 15000 bolts in it, if the designers use, say 1000 types of bolts, it will cost the owner about $3 million to refasten the bolts on every gadget every year. Of course the situation is not that bad.

Solution: Since design of fastening joints (bolts, rivets, and welds) is not that complex, automation of the whole process was suggested. An intelligent Fastening Joint designer was developed on top of CADDS system.
Whenever a designer wants to fasten two, or more, parts together, all he needs to do is to click upon a menu icon inside CADDS. CADDS will ask the user to digitize the parts needed to be fastened.
CADDS passes the information to the our system that accesses the part database and it extracts the kinematic and physical information about the parts. Consequently the system decides the type of fastening (bolt, rivet, or weld), then it fully analyzes, then designs the fastening joint according to all the rules of the company, and using the inventories of the company.
The system updates the CADDS part database to reflect the new design. The whole design process takes less than 3 minutss.

5.3.4. Parametrized Design of an Office Partition

Problem: Discrete parametric modeling of office partitions. A major manufacturer of office furniture wanted to develop a system that takes the functional specifications from customer, and the system should design a product made of the manufacturer's standard components. A family of components may be represented by a parametric model, but discrete domain for some of its parameters. In other words, not every consistent set of values will constitute a solution.
Solution: The various families of office partitions were parametrically modeled. Each components was also parametrically modeled. Unlike systems based on simultaneous equation solvers, the system allowed discrete parameters, which was vital for this application. The application generates a bill of material (BoM), and an engineering drawing ready for manufacturing.

5.3.5. Synthesis of the Front Panel of a Car Stereo

The model was developed for Philips - Consumer Electronics (Eindhoven, Netherland).

Various automotive manufacturers use Philips' car stereos. The product has to comply to the standards of the car manufacturer, to the standards of the market country, and to the shopfloor capabilities at the manufacturing plant. Therefore, for every model, several versions may exist. This adaptation process amounts to almost complete redesign.

Philips needed to parametrically model the front panel of a car stereo so it can impose a new constraint with minimal effort.

To do that, we used one chapter of the German DIN standards and one chapter of

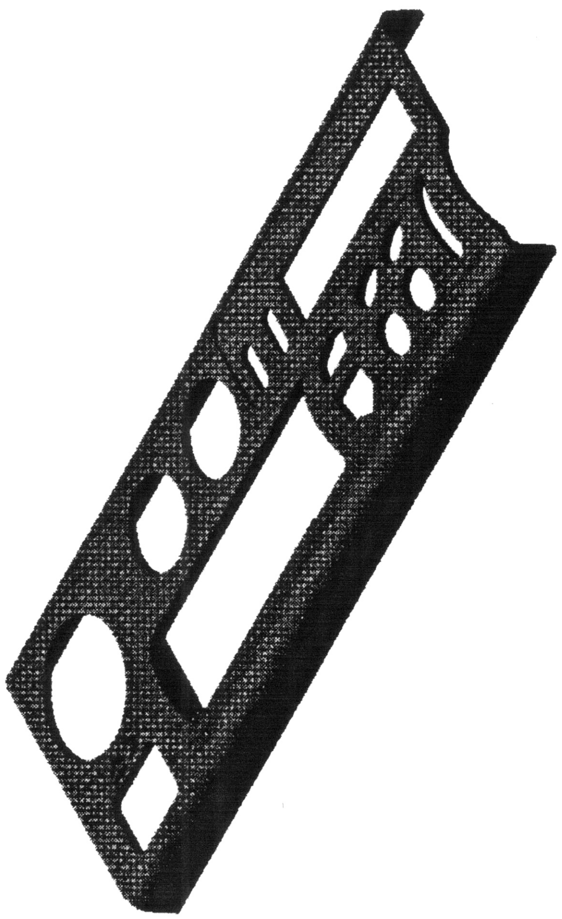


Figure (5.20): Front Panel of Car Stereo - Courtesy of Philips

the European ECE code that deal with front panels of car radios.

The specifications deal with the geometric representation of some functionalities. Therefore a functional model of the front panel had to be established.
Every front panel of the car stereo consists of four functional groups:

1. Amplifier Group

2. Tuning Group

3. Cassette Group

4. Display Group.

A functional/geometric parametric model is established for every model. Specifications are enforced upon this model.

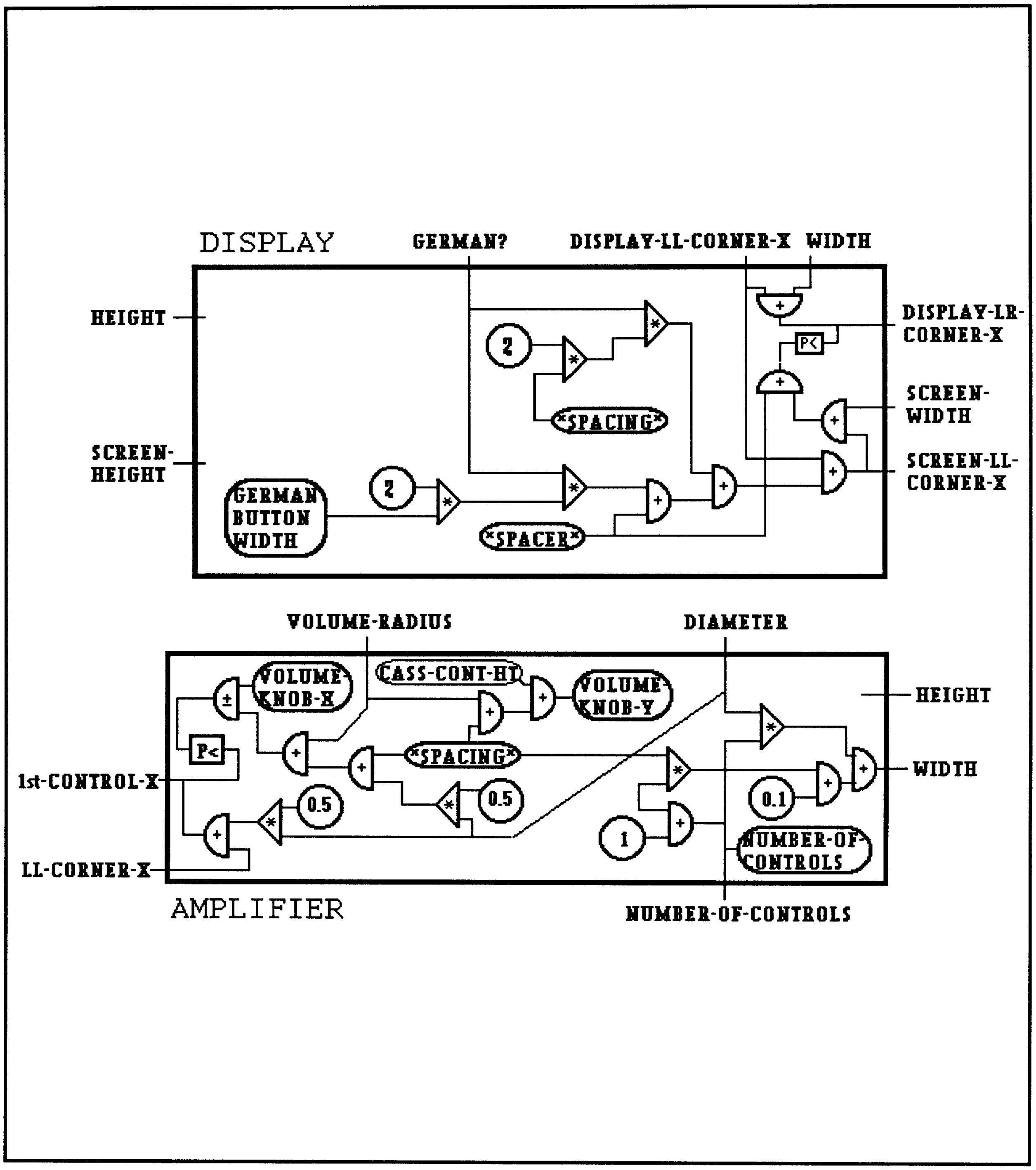


Figure (5.21): Schematic Representation of a Front Panel

'There is no permanence.
Do we build a house to stand forever, do we seal a contract to hold for all time?
Do brothers divide an inheritance to keep for ever? Does the flood-time of rivers endure?
It is only the nymph of the dragon-fly who sheds her larva and sees the sun in his glory.
From the days of old there is no permanence.'

Utnapishtim, the faraway
(Noah in Epic of Gilgamesh, circa 3500 BC)

Summary & Conclusion

This chapter first summarizes the system, presented throughout this dissertation, then it compares its approach to traditional approaches to design. Then the contribution of the thesis is offered. The third section discusses the limitations of the system, and the future work needed for further development. Finally, the conclusion describes how the implemented system fits the hypothesis described in chapter one.

6.1. Summary

This thesis casts the engineering design process mainly as the propagation of a body of specifications (constraints). The propagation of these constraints leads to a series of conflicts between various, occasionally-competing points of view. The resolution of these conflicts is the embodiment of the final design.

To test this perception, a system was developed for design automation. The system captures design as a network of specifications, at some state of conflict resolution, that propagated a certain set of values for the design parameters. The system used a mechanism for *constraint local propagation*. Similar to traditional design approaches, the system documents a design both in calculations and illustrative graphics. To be useful for other designers, the design, moreover, includes the alternatives that were available for the designer, and the dead-ends he tried out. All these aspects of a design constitute a design case to be added to the memory (experience) of a designer.

The philosophy was translated into a system (environment) of computer programs. The system covers the whole engineering process, starting from conceptual design, through analysis, dimensioning, drafting, and checking to manufacturing and maintenance. This environ- ment was applied to various stages of that engineering process in several domains. The versatility of the applications contributed positively to the refinement of the implementation of the system. The various applications of the environment in different domains prove both the validity of the philosophy, and the robustness of the implementation. Applications, nonetheless, illuminated some limitations of both the approach and the implementation. These limitations were discussed at the beginning of this chapter.

The system uses two representations for each artifact, an abstract (mainly-functional) representation, and a detailed representation. This dimorphism was suggested by several researchers in cognitive modeling. On the other hand, to assure portability and learning, modeling in the system is divided into three classes: functional representation, geo- metric representation, and graphical representation. The knowledge representation used in the system allows for the extension of the system to handle innovative design through case-based learning and reasoning.

The system architecture proposed in this dissertation for design automation is an attempt to rationalize the design process. In addition to the rationalization and the development of clear algorithms for the major design tasks (utitlities), the system has further advantages when compared with the traditional design programs, including the following attributes:

* The system allows the user to impose constraints at any point of the design. A manufacturing constraint, imposed late in the process, may propagate backward to change the conceptual design selected earlier.

* A design made by this system includes, in addition to calculations and illustrative graphics, the alternatives that were available for the designer, and the dead-ends that were tried out.

* Because of the previous capability, the system can utilize previous designs to develop better designs, with less computational resources.

* The system allows a design to be either under-constrained or over-constrained. Over-constrained designs can be resolved by the conflict resolution mechanism of the system. Under-constrained designs can be completed by mechanisms for posting default and assumption values for unknown parameters. Default and assumption values are of lower merits than the values propagated by the user. Therefore, they succumb to values of higher merits in conflict resolution.

6.2. Contribution of the Thesis

There is one major contribution of the thesis. In the following paragraphs, a description of the various facets of that single contribution, namely the environment for specification-driven design is given.

6.2.1. (I) An Architecture for Design Problem Solving

The major contribution of the thesis is the design of an environment for design utilities. Most of the elements of this environment are developed (for concept proving purposes).

The major features of this environment are the following:

Specification-driven Design:

The design process is perceived to be controlled by one type of information, which is specification. A specification could be used in at least three major ways:

- ° To **check** the compliance of an existing design.
- ° To **synthesize** a new design from scratch based.
- ° To **be manipulated** by a specs builder that goes through a body of standards, selects the articles relevant for the problem at hand, then creates a body of specs.

Checking:

Checking is made through **unification** between a specification (constraint type) and every legible object in the design (Kowalski 79). The result of the unification process is the creation of an instantiation of the constraint pinned to the parameters (attributes) relevant to that object.

Learning:

With the invention of CAD systems, instead of burying our designs in dark cold vaults, we now bury them on magnetic tapes. Once a design is over, no use is made of it in subsequent projects. This is not the case with human designers, where every additional project adds to the experience, and hence to the performance, of the designer. Abstractions of all previous designs are lurking in the back of the designer's mind whenever he is faced with a new project. Through some analogy, the designer decides the similarity between the requirements of an old design and those needed for the current project.

The system architecture captures a complete design as a constraint propagation network. This representation is used as an abstraction of the system ever lurking in the back of the system's mind when encountering a new design project.

Synthesis:

The same representation allows for generic design synthesis for non-electronic products, which are versatile and infinite. All you need to have is a body of specifications of the required product and the existence of a mapping between functionality and geometry for that category of devices (After all, the scope of the thesis is traditional design rather than innovative design).

6.2.2. (II) User-Defined Conflict Resolution Rules

Unlike other constraint propagation programs, that have pre-defined scalar merit values, this system allows the user to define whatever merit values he may need, and the the behavior that reflects real-life environment. *This capability allows the construction of semantically-sound design environment that resembles the real environment.*

The merits could be structural-engineer, architect, senior architect, owner, ASHRAE-standard, etc. Then the user can define how to prioritize these merits. The conflict resolution rules could be plain scale of merits and it could be any behavior modeled for the system to follow. Such merits will represent the real proponents of every specification, such that at a time of conflict between specifications, the system can follow the user's instructions to resolve the contradiction.

Every parameter in a constraint propagation system has, in addition to its value, a merit level that will be used in conflict resolution in case of a contradiction.

In most of the classical CP systems (Steele 80, Gosling 84), the merit scale is made of a finite set of pre-defined levels of merits, e.g., Steele & Gosling offer three-level linear scale (in decreasing importance): Constant, Default, and Parameter. So if a conflict is detected, the culprits are inspected for their merit values. The one with the lowest merit will be overwritten. Merits in this system are not on a scalar range. A merit may possess quite a complex1 behavior that drives it differently against other merits. When a conflict is detected, each of the merits of the initial premises is compared against the other merits according to the defined behavior.

6.2.3. (III) Harvesting Mechanism for Adaptive Learning in Evolving Domains:

Harvesting is a technique for improving the model of an evolving domain using the feedback on previous cases. In other words, harvesting uses *case-based reasoning*, and feedback, to improve its *model-based reasoning* capabilities.

1 The merit behavior can be as complex as a Lisp function can be.

An *evolving domain* is a one that is not governed by a well-established model (theory). Instead, starting from one set of premises, several competing theories can lead to various results. These theories are incomplete, in the sense that they cannot cover the whole domain, and/or unreliable, in the sense that they not always lead to the right answer.

An *explicit artifact* is an artifact whose shape is a direct function of its functionality. Consequently, a single parametric model represents that explicit artifact. In other words, the geometry of the artifact can be represented as terminal nodes in the causality network.

The design of explicit artifacts in evolving domains is a subject of a a great deal of heuristics and empirical formulae, e.g., shear design of metal formed artifacts, design of asphalt roads.

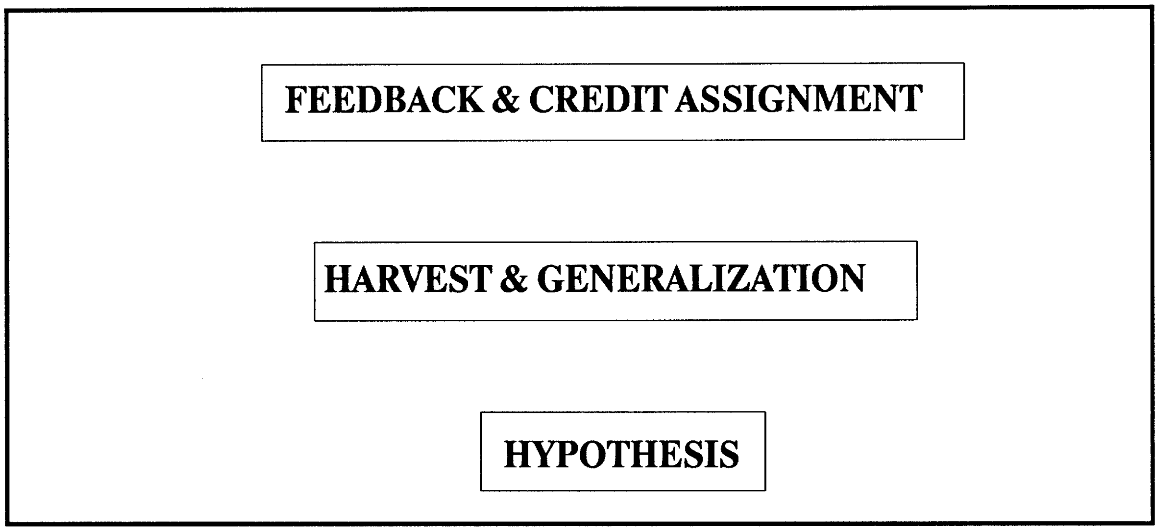


Figure (6.1): The Harvest Mechanism

The harvest mechanism has three parts: *the feedback & credit assignment part* for discarding the unreliable pieces of knowledge, the harvesting part to collect knowledge from different sites, and the *hypothesis part* for postulating new relationships in the domain.

6.2.3.1 The Feedback & Credit Assignment Part

This is the part that collects feedback on every design case according to an agenda mechanism in the case library. The system backpropagate credit all over the reasoning path used in reaching the design.

Through the feedback feature of the architecture, every design case is augmented by a follow-up result, which is a credit value ranging from -1 to 1. The system uses the reasoning capability to backpropagate the credit through the causality network using a credit assignment mechanism. This credit is assigned to the constraint that represents a causal relationship. As time goes on, the system accumulates credit on the various causal links (constraints). The user can set a credibility threshold. The system will not propagate through a causal constraint whose credibility is less than the set threshold. Periodically, the system will report all the causal links (constraints) that accumulated a sufficiently negative credit to the user so he can replace them or fix them.

6.2.3.2 Harvesting & Generalization

This is the part that harvest (collects) the previously-identical copies of the knowledge bases released to the various sites. The system runs a super generalization session on them to improve the qualitative knowledge of the evolving domain.

Because of the adaptiveness of the system, the identical copies of the knowledge base released to various users, or sites, will evolve in different directions according to the circumstances and cases encountered in every site. To utilize this gain of knowledge toward the finessing of a theory for the evolving domain, and to make sure that the gained knowledge is not just local perturbance, all the previously-identical copies of the knowledge base, with all the cases encountered by each copy, are gathered every year for a super generalization session.

Super Generalization Session applies several generalization techniques, as described in (Michalski 84), to the causal and taxonomical knowledges.

The new version of the knowledge base with all the generalizations added, and all the low-credibility links removed, is broadcast to all sites again.

6.2.3.3 Relationship Hypothesis Part:

This is the part that suggests new relationships (equations) between parameters to modify the knowledge base of evolving domains.

Every parameter is connected through constraint networks to all related parameters. In addition to the definite relationships represented by the constraint networks, every parameter has a list of possible influences, i.e., other parameters that may have influence on the parameter under consideration. Periodically, the system goes through the values of the *possible-influences* parameters in the accumulated cases, and it runs *Stepwise Regression Analysis*, in conjunction with *Dimensional Analysis* to develop new relationships with a significance level above a user-set level. The new hypothesis (relationship) will not be translated into a constraint until it is approved by the user. This capability makes the system continuously adaptive.

6.2.4. (IV) Constraint building environment:

1. **The user can edit/construct a constraint (network) either textually or schematically.** Figure (2) shows the textual description of a macro-constraint representing a simple Portal Frame. Figure (3) shows the schematic representation of the same constraint. The user can build a constraint through either way and the other representations will be generated automatically.

   Currently, work is under way on making the third representation, which is geometric display, another valid constraint specification tool. This step needs extensive work with Constructive Solid Geometry (Boolean operations on solids).

```
(defcon FRAME (span depth R SF BM)
            ((* MMULT) BM (global *equiv-load*)
                           (* (exp span 2.0) % 12.0))
            ((* RMULT) R  (global *equiv-load*) (* span 0.5))
            (* R SF 1.0)
            ((design fd) (global *breadth*) depth BM)
            )
```

Figure (2): Text Specification of Portal Frame Constraint

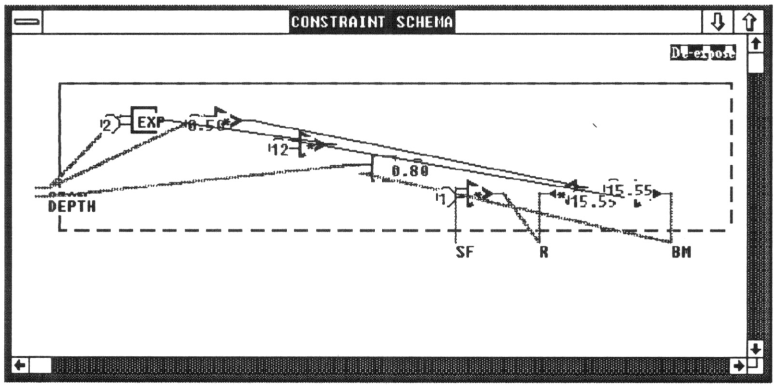
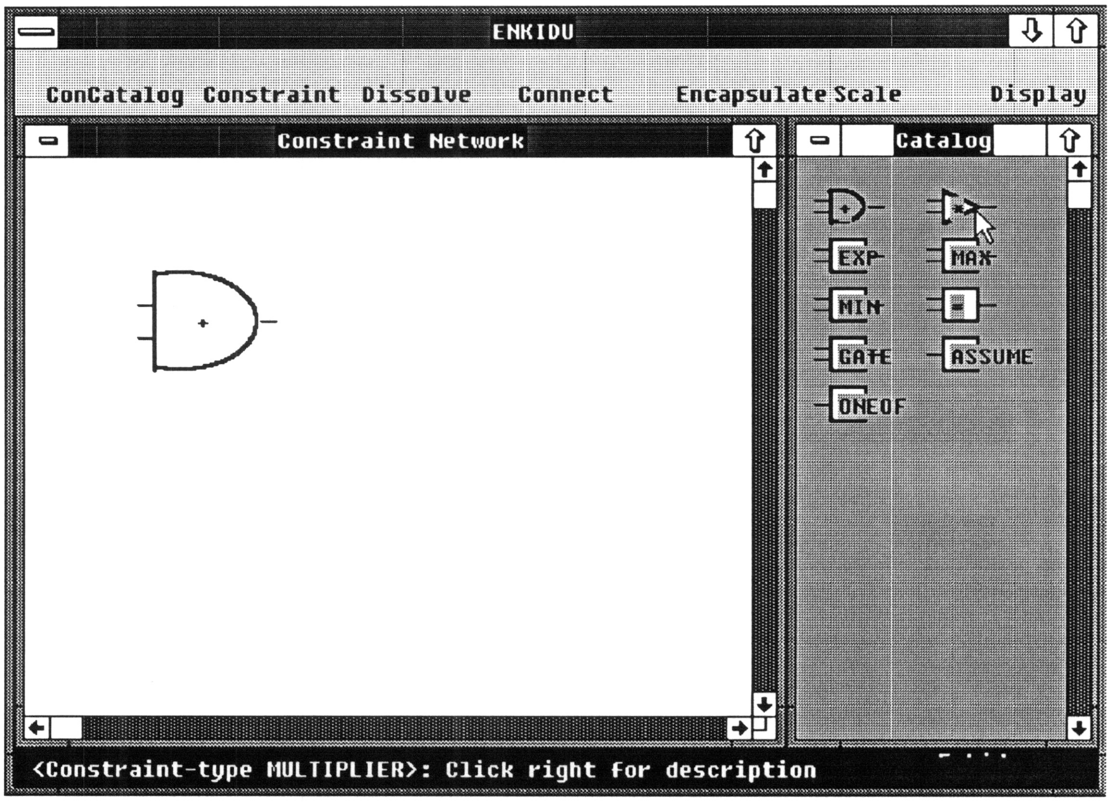


Figure (3): Schematic Editing

2. **User can iconically load different constraint libraries that represent the specifications he wants to use.** A screen dump of the constraint construction environment is shown in figure 4. A description of this environment is given in Chapter Four on *Programs description*, p. 88.



<Constraint-type MULTIPLIER>: Click right for description

Figure (4): The Interactive Environment

6.3. Limitations

This section discusses some of the obvious limitations of the implementation developed to flesh out the philosophy and methodology discussed in chapters One and Two.

1. Memory intensive

Representing the whole design in a constraint network may prove to be prohibitively expensive, especially in swap space. This problem gets exacerbated if the user wants to apply large number of specifications (such as several chapters of the ASTM) during design.

Defense: But you have to have all your design information active if you are performing the design (parametrically) by hand.

Solution: Subdivide the network into smaller networks, e.g., conceptual architectural design, structural design, electrical design, mechanical design, etc.

Disadvantage: It limits the propagation of constraints across subnets, e.g., it will not be possible for the system to discover that the only solution for the congested air conditioning duct is to modify the structural depth of the beam under which the duct goes, which requires a modification of the structural system/subnet.

Another Solution: Draw thicker walls around the macro constraints to curtail the inter-queue traffic across macro constraints. Most of the traffic is for conflict resolution. Therefore, such a thicker wall could be made by adding a criterion for the conflict resolution mechanism that will make a cell, from a different macro constraint, the least possible suspect. If there are cells belonging to more than one macro constraint, ranking will be done according to the ultimate (topmost) owner of the cell.

2. Circular Dependency

It is a formidable problem especially when we deal with heavily connected networks, the result of a specification-rich domain. The simplest example for circular dependency is: $x^2=y$

Solution: Careful devising of networks may alleviate the problem partially.

Disadvantage: Not always true. Cannot be tolerated in a real design environment.

Another Solution: Transform the subnet that isolates the circular dependency into a primitive constraint using an algebraic systems, like Macsyma, Maple, etc. (Gosling 84).

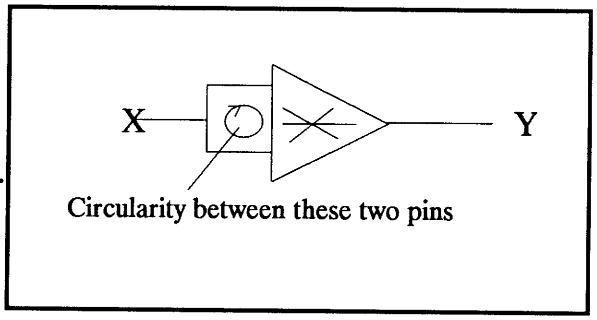


Figure (6.5): Circularity in X * X = Y

Advantages: For an equational system, the user will never be bugged by circularity dependency.

Disadvantages:

° You may lose the functional mapping of a macro constraint if the circularity stretches across two component macro constraints.

° An algebraic system cannot handle anything other than equations, which leaves us with inequalities and discrete variables (oneof) that cannot be handled.

3. Recursion:

No current implementation of constraint management systems can handle recursion. The obstacles that hinder the development of a recursive constraint are:

° The unformidable problem of circular dependency.

° The need for incorporating an unlimited number of boundary conditions as pins of the new constraint.

° The lack of a semantic model to represent a recursive constraint in reality, compared to the traditional schematic representation of constraints.

Possible Solution: Considering the obstacles. all together, leads us to the direction of developing a new class of constraints that will be checked for circular dependency right after formation. In that new class, special mechanism needs to be devised to continuously worry about the boundary conditions.

4. Differential Equations & Optimization:

Optimization and other applications of differential equations cannot be modeled easily because of the constants of integration resulting in the inverse differentiation (integration). Limiting the range of the numerical integration and using boundary conditions can help solve this problem.

5. Adequacy of Constraint Propagation for Geometric Parametric Modeling:

Geometry is quite an expensive domain for the constraint propagation paradigm. It is very expensive, if at all possible, to represent all the reasoning for every geometric constraint. A better approach will be to separate the parametric representation of the world into two parts:

° Functional Parametric Modeler that handles non-geometric parameters and their relationship with the major geometric parameters.

° Geometric Parametric Modeler that handles the whole geometry based on chronological precedence of constructing the geometry, called Constructive Solid Geometry (Mortenson 85).

6.4. Conclusion

The system presented in this environment was developed to test the validity of the philosophy of thethesis, as described in chapter one (Hypothesis). The use of the environment proved both the suitability and robustness of the concept and the modular extensibility of the environment. The specification-driven view of the design process turned out to be important in more aspects than initially thought. Frequently used specifications, such as standards, can be offered in an interactive (enforceable) form, in addition to the usual passive text format. Having an interactive version of standards proved to be very useful in both design synthesis and design checking. Such standards need to be encoded only once, then every body can use it.

The system had to have a constraint management mechanism. The system had its own implementation of the concept of local propagation of constraints. While local propagation is the most suitable approach for constraint management in design automation, it has some formidable limitations, as discussed in the section on Limitations. These limitations should not be a worry for long time, since the the concept of constraint local propagation is inherently parallel. With the spread of massively parallel architectures, local propagation will be a very powerful tool.
Throughout the system, we tried to build the system to be completely independent of both the concept of constraint management and the implementation of that concept such that an alternative approach to constraint management can be adopted.

The environment is an attempt to develop a design automation environment that can work on top of any Computer-Aided Design system. Therefore, different CAD systems were used in the various applications.

6.5. Future Work

The following are the features thought to make the system more representative of the philosophy discussed in chapter One.

1. Pattern Matching in a Case Library - Parametrization of Macro Constraints:

For example, constraint types need to accept variables for better representation and better search, as shown in the following example:

A three-story building should be of the same type as four-story building. A variable should represent the number of repetitive floors (devices).

2. Design as an optimization process:

Optimization is a problem with more than one design criterion. The multiple criteria are applied in the form of an Objective function to establish the relative weights of the various criteria, e.g., minimize the side sway of the frame.

A by-product of such representation is the introduction od **Sensitivity analysis.**

This work is dependent on the development of a class of constraints that can handle calculus.

3. Harvesting

After a design is made, and is under construction, every problem that gets reported back to the design office needs to backpropagate through the reasoning chain that led to the fault in the first place. A negative credit needs to be assigned to every ring in this chain. As time goes on, negative and positive credits accummulate throughout the constraint network. Design synthesis should take this credit accummulation issue into considration. In this case, we have to synthesize an innovative solution. Here, the work of (Doyle 88), and (Urlich 88) could be of great relevance to the problem.

4. Adapting Algebraic Transformation as a Partial Solution of Circular Dependency

Since the system has the capability of detecting circular dependency, it should not be hard to develop a transformation module, similar to that described in (Gosling 84), that will do the following:

° Take a circular subnet,

° Reduce it to the minimal manifestation of circularity,

° Transform the minimal circular subnet into a set of algebraic equations, if possible.

° Develop a single primitive constraint that represent the set of circular equations.

° Replace the subnet by the new primitive constraint.

5. Versioning & Engineering Change Orders:

Changes in engineering design are quite common. Controlling/organizing different changes throughout a design is a formidable task. While constraint propagation facilitates the changes, a bookkeeping mechanism is needed to keep track of the various revisions of the design. Such a mechanism has to be included in the top-level propagation mechanism, in the function called *run!*.

6. Specification-driven Design & Primitive (Constructive Solid) Geometry:

Work needs to be done to establish a constraint-based representation of constructive solid geometric modeling. Constraints are better suited for this application than the chronological backtracking schemes followed in all CSG modelers.

7. Utilization of Macro Constraints in Reasoning:

The reasoning mechanisms ignore the existance of the boundaries of the macro constraints. There is no difference in the system between a general cell in a constraint network and a cell that is inside a macro constraints. This indistinction does not conform with the concept of macro constraints, and it overwhelm the user with suppressable details.

8. Use of Locally-Acquired Knowledge that gets Globally Refuted in Harvesting:

What happens locally to an acquired piece of knowledge that was not adapted during the harvesting super generalization session. Currently the system discards such an assertion. More work needs to be devoted to the issue of assumption-based truth maintenace systems since they may have a solution to it.

References

Abelson, Harold, Michael Eisenberg, Mathew Halfant, Jacob Katzenelson, Elisha Sacks, Gerald Sussman, Jack Wisdom, Ken Yip, Intelligence in Scientific Computing, Cambridge, MA: MIT - AI Lab, AI Memo 1094, November, 1988

Abelson, Harold, Gerald Sussman & Julie Sussman, Structure and Interpretation of Computer Programs, New York, NY: McGraw-Hill, 1984

Adams, James L., Conceptual Blockbusting: A Guide to Better Ideas, Third Edition, Reading, MA: Addison-Wesley, 1986

Aho, Alfred, Hopcroft, John and Ullman, Jeffrey, The Design and Analysis of Computer Algorithms, Reading, MA: Addison-Wesley, 1974.

Akin, Omer, Expertise of the Architect, in Michael D. Rychener (ed.), "Expert Systems for Engineering Design", San Diego, CA: Academic Press, 1988.

Alexander, Christopher, *Notes on the Synthesis of Form*, Cambridge, MA: Harvard University Press, 1964

Bridgman, P.W., *Dimensional Analysis*. New Haven, CT.: Yale University Press, 1931.

Boothroyde, A., *Design for Manufacturability*, University of Rhode Island, Industrial Engineering TR-11, 1984.

Borning, A., *The Programming Language Aspects of Thing-lab*, a Constraint-Oriented Simulation Laboratory, *ACM TOPLAS 3, 4*, Oct. 1981, pp. 252-387.

Changeaux, Jean-Pierre, *Neuronal Man: The Biology of Mind*, Oxford, UK: Oxford Univ. Press, 1985.

Charniak, Eugene, Riesbeck, Christopher, McDermott, Drew, and Meehan, James, *Artificial Intelligence Programming*, 2nd Ed., Hillsdale, N.J.: Lawrence Erlbaum Associates, 1987.

Chomsky, Noam, Aspects of the Theory of Syntax, Cambridge, MA: MIT Press, 1965.

Cohen, Jacques, *Constraint Logic Programming Languages*, *Commun ACM 33*, 7, July 1990, pp. 52-68.

Colmerauer, Alain, *Introduction to Prolog III*, *Commun of the ACM 33*, 7, July 1990, pp. 69-90.

Connor J. & F. Chehayeb, *GEPSE: A Rule Based Shell For Design of Buildings*, MIT Intelligent Engineering Systems Laboratory , IESL- 10, 1987

Davis, William S., *Systems Analysis and Design: A Structured Approach*. Reading, MA: Addison Wesley, 1983.

DeJong, G., and Mooney, R., *Explanation-Based Reasoning: An Alternative View*, Machine Learning 1 (2), p. 145-176, April 1986.

Descartes, Rene, *Discourse on Method*, La Salle, IL: Open Court Classics, 1962.

Dixon, J.R., *Design Engineering: Inventiveness, Analysis, and Decision Making*. New York, NY: McGraw-Hill, 1966.

Doyle, Jon, *A Truth Maintenance System*, Cambridge, MA: MIT-AI Lab, AI Memo 521, May 1977.

Doyle, Richard J., *Hypothesizing Device Mechanisms: Opening the Black Box*", Cambridge, MA: MIT-AI Lab, AI-TR 1020, 1988.

Eastman, Charles M., *Explorations of the Cognitive Processes in Design*, Pittsburgh, PA: CMU - Dept. of Computer Science, February, 1968.

el-Shafei, Nayel S., *Quantitative Discovery and Qualitative Reasoning about Failure Mechanisms in Pavement,* Cambridge, MA: MIT, M.S. Thesis, 1986.

Erkens, A., *Beiträge zur Konstruktionserziehung*, Z. VDI[1] 72, 17-21, 1928.

Feldman, J., and Ballard, D., *Connectionist Models and their Properties*, *Cognitive Science, 6 (3)*, p. 205-254, 1982.

Flanagan, Owen J., Jr., *The Science of the Mind*, Cambridge, MA: MIT Press - A Bradford Book, 1984.

Foley, J.D. and Van Dam, A., *Fundamentals of Interactive Computer Graphics*, Reading, MA: Addison-Wesley, The Systems Programming Series, 1982.

Freeman-Benson, B.N., Maloney, J., Borning, A., *An Incremental Constraint Solver*, *Commun. ACM 33*, 1, pp. 54-63, January 1990.

Freuder, Eugene C., *Partial Constraint Satisfaction*, pp.278-283, Detroit, MI: Eleventh International Joint Conference on Artificial Intelligence, August 1989.

1 VDI is an acronym for Verein Deutsher Ingenieure.

Garey, Michael, & Johnson, David, *Computers and Intractability*, W.H. Freeman and Company, 1979.

Gaylord, M & L Gaylord, *Handbook of Structural Design*, 7th Edition, New York: McGraw-Hill, 1984.

Gazdar, Gerald, & Mellish, Chris, *Natural Language Processing in Lisp: An introduction to Computational Linguistics*, Reading, MA: Addison-Wesley, 1989.

Geach, Peter, *Abstraction Reconsidered.* In Ginet, Carl, & Shoemaker, Sidney, (eds.), *Knowledge and Mind - Philosophical Essays*, Oxford, U.K.: Oxford University Press, 1983.

Gentner, D., *Structure-Mapping: A Theoritical Framework For Analogy*, Cog. Sci, 7, p. 155-170, 1983.

Gero, John S., & Coyne, Richard D., Developments in Expert Systems for Design Synthesis, in Proc. of a Symposium on Expert Systems in Civil Engineering, sponsored by ASCE-TCCP, Seattle, WA, p. 193-203, April 1986.

Gieck, Kurt, Engineering Formulas, 5th American Ed., New York: McGraw-Hill, 1986.

Gosling, James, *Algebraic Constraints*. Pittsburgh, PA: Carnegie-Mellon University, Ph.D. Thesis 1983

Grigorev, D.Y., and Vorobjov, N.V., *Solving Systems of Polynomial Inequalities in Subexponential Time. J. Symbolic Comput.* 5, 1988, pp. 37-64.

Hammond, Kristian J., *Case-Based Planning: Viewing Planning as a Memory Task.* San Diego, CA: Academic Press, 1988.

Harrelson, William & Silvistri, Mark, "Constraints as Representation for the Third and Fourth Levels of PDES", 1988.

Haugeland, John, Mind Design - Philosophy, Psychology, Artificial Intelligence, Cambridge, MA: MIT Press - A Bradford Book, 1981.

Helal, Mohammed, Design of Reinforced Concrete Structures, 2nd Edition, Cairo, Egypt: Cairo University Press, 1980.

Highway Research Board, Standard Nomenclature and Definitions for Pavement Components and Deficiencies. Special Report 113, TRB, 1970.

Hillis, Danny, *The Connection Machine*. Cambridge, MA: MIT Press, 1986.

Hofstadter, Douglas R., *Goedel, Escher, Bach: An Eternal Golden Braid.* New York: Vintage Books (A division of Random House), 1979.

Jaffar, J. & Lassez, J-L., *Constraint Logic Programming.* In *Proceedings of the Fourteenth ACM Symposium of the Principles of Programming Languages*, Munich, 1987, pp. 111-119.

Keene, Sonya, *Object-Oriented Programming in Common Lisp: A Programmer's Guide to CLOS.* Reading, MA: Addison-Wesley, 1989.

Kim, J.J. & Gossard, D.C.: *Reasoning on the Location of Components for Assembly Packaging.* Proc. ASME Design Automation Conference, Montreal, Canada, p.251-257, Sept.,1989

de Kleer, Johan and Brown, John Seely, *A Qualitative Physics Based on Confluences.* In Bobrow, Daniel G. (ed.), Qualitative Reasoning about Physical Systems, Cambridge, MA: MIT Press, p. 7-84, 1985.

de Kleer, Johan, *A Comparison of ATMS and CSP Techniques*, pp.290-296, Detroit, MI: Eleventh International Joint Conference on Artificial Intelligence, August 1989

Knuth, Donald E., *The Art of Computer Programming*, Volume 3: *Sorting and Searching*, Reading, MA: Addison-Wesley, 1973

Kowalski, Robert, Logic for Problem Solving. Amsterdam: North-Holland The Computer Science Library, AI Series, 1979

Kuffler, Stephen W. and Nichols, John G., *From Neuron to Brain: A Cellular Approach to the Function of the Nervous System.* Sunderland, MA: Sinauer Associates, Inc., 1976.

Kuipers, Benjamin, *Commonsense Reasoning about Causality: Deriving Behavior from Structure.* In Bobrow, Daniel G. (ed.), Qualitative Reasoning about Physical Systems, Cambridge, MA: MIT Press, p. 169-204, 1985.

Kyburg, Henry E., Jr., *Epistemology and Inference*, Minneapolis, MN: University of Minnesota Press, 1983.

Laird, J., P. Rosenbloom, & A. Newell, *Chunking in SOAR: The Anatomy of General Learning Mechanism, Machine Learning, 1 (1)*, p. 11-46, 1986.

Lathrop, Richard, & Temple, John, *ARIADNE: Pattern-Directed Inference and Hierarchical Abstraction in Protein Structure Recognition, Comm. of the ACM*, November 1987.

Mackworth, Alan, *Constraint Satisfaction*. In Stuart Shapiro (ed.), *Encyclopedia of Artificial Intelligence*, New York: Wiley-Interscience, 1987.

Maher, MaryLou, *Hi-Rise: An Expert System for Preliminary Structural Design*. In M.D. Rychener (ed.), *Expert Systems for Engineering Design*, San Diego, CA: Academic Press, 1988.

Martin, Charles E., *Indexing Using Complex Features*. In proc. *Case-Based Reasoning Workshop*, Sponsered by DARPA Information Science and Technology Office, p.41-44, May 1989.

McAllester, David A., *A Three-valued Truth Maintenance System*. Cambridge, MA: MIT-AI Lab, AI Memo 473, May 1978.

McDermott, Drew & Doyle, Jon, *Non-Monotonic Logic*, Cambridge, MA: MIT-AI Lab, AI Memo 468b, July 1979.

Michalski, Ryszard, A Theory and Methodology for Inductive Learning. In Michalski, R. et al. (eds.), Machine Learning: An Artificial Intelligence Approach, Palo Alto, CA: Tioga Publishers, Kauffmann, 1984

Minsky, Marvin, *Matter, Mind, and Models*. In Minsky, Marvin (ed.), Semantic Information Processing, Cambridge, MA: MIT Press, p. 425-432, 1968, Heuristic Value of Quasi-Separate Models.

Minsky, Marvin, The Society of Mind. New York, NY: Simon & Schuster, 1987.

Mitchell, T., Keller, R., & Kedar-Cabelli, S., *Explanation-based Generalization: A Unifying View*, Machine Learning, 1: 47-80, 1986.

Moavenzadeh, Fred, & Brademeyer, Brian, A Stochastic Model for Pavement Performance and Management. University of Michigan, Ann Arbor, MI, 1977.

Mortenson, Michael E., *Geometric Modeling*. New York: John Wiley & Sons, 1985.

Nii, H.P., Aiello, N. and Rice, J., Framework for Concurrent Problem Solving: A report on CAGE and POLIGON, in Engelmore, Robert and Morgan, Tony, Blackboard Systems, Reading, MA: Addison-Wesley, 1988.

Pahl, G. & W. Beitz, Engineering Design: A Systematic Approach, Berlin, Germany: Springer-Verlag, 1988 (Ch.3: The Design Process & Ch.5: Conceptual Design).

Pentland, Alex & Johm Williams, Perception of Non-Rigid Motion: Inference of Shape, Material and Force, pp.1565-1570, Detroit, MI: Eleventh International Joint Conference on Artificial Intelligence, August 1989

Pereira, Fernando C.N., & Shieber, Stuart M., Prolog and Natural-Language Analysis, Stanford, CA: Center for the Study of Language and Information (CSLI), Lecture Notes, 1987.

Pople, Harry, Jr., *Heuristic Methods for Imposing Structure on Ill-Structured Problems: The Structuring of Medical Diagnosis*. In P. Szolovits (ed.), *Artificial Intelligence in Medicine*, AAAS, 1979. The remedy presented by the paper is the idea of synthesized links of generalization as the adhesive factor in polymorphism.

Potter, J.L. (ed.), *The Massively Parallel Processor*, Cambridge, MA: MIT Press Scientifc Computation Series, 1985.

Ressler, Andrew Lewis, *A Circuit Grammar for Operational Amplifier Design*, Cambridge, MA: MIT-AI Lab, AI-TR 807, 1984.

Rieger, Chuck & Grinberg, Milt, *The Causal Representation and Simulation of Physical Mechanisms*. Technical Report TR-495, University of Maryland, November 1976.

Rivin, Igor, and Zabih, Ramin, *An Algebraic Approach to Constraint Satisfaction Problems*, pp.284-289, Detroit, MI: Eleventh International Joint Conference on Artificial Intelligence, August 1989.

Rumelhart, D., Hinton, G., and Williams, R., *Learning Internal Representations by Error Backpropagation*, in Rumelhart, D., and McLelland, J., (eds.), Parallel Distributed Processing, Cambridge, MA: MIT Press, p.318-362, 1986.

Rumelhart, D., and McLelland, J., (eds.), *Parallel Distributed Processing*, Cambridge, MA: MIT Press, 1986.

Sacks, Elisha, *Hierarchical Inequality Reasoning*, Cambridge, MA: MIT-Lab for Computer Science, LCS-TM 312,1987.

Sandars, N.K. (Translator), *The Epic of Gilgamesh*, New York, NY: Penguin Classics, 1972.

Sandler, Ben-Zion, *Creative Machine Design: Design innovation and the right solutions*, New York, NY: The Solomon Press, 1985

Saraswat, Vijay, *Concurrent Constraint Programming*, Pittsburgh, PA: Ph.D. Thesis, CMU, 1989.

Schank, Roger C., *Dynamic Memory: A Theory of Learning in Computers and People*, Hillsdale, N.J.: Lawrence Erlbaum, 1982.

Simon, Herbert, The Sciences of the Artificial, Cambridge, MA: MIT Press, 1966.

Simon, Herbert, Models of Discovery, Boston, MA: Reidel Co.- Pallas Paperbacks, 1977.

Smillie, K.W., *An Introduction to Regression and Correlation*. London, U.K.: Academic Press, 1966.

Sriram, Duvvuru, Knowledge-Based Approaches in Structural Design, Ph.D. Thesis, Carnegie-Mellon University, Civil Engineering Department, 1985.

Stallman, Richard M., & Sussman, Gerald J., *Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-aided Circuit Analysis*, Artificial Intelligence, 9: 135-196, 1977.

Steele, Guy L. Jr., & Sussman, Gerald J., *Constraints*, Tech. Rep. MIT-AI Memo 502, Nov. 1978.

Steele, Guy L. Jr., *The definition and Implementation of a Computer Programming Language Based on Constraints*, Cambridge, MA: MIT-AI Lab, AI-TR 595, 1980.

Steele, Guy L. Jr., *Common LISP: The Language*, 2nd Ed., Billerica, MA: Digital Press, 1990.

Sussman, Gerald J., SLICES: *At the Boundary between Analysis and Synthesis*. AI Memo 433, Cambridge, MA: MIT - AI Lab, July 1977

Sutherland, Ivan E., SKETCHPAD: *A Man-Machine Graphical Communication System*, Cambridge, MA: MIT - Lincoln Lab TR 296, January 1963

Taylor, E.S., *Dimensional Analysis for Engineers*. Oxford, U.K.: Clarendon Press, 1974.

Thagard, Paul, & Holyoak, Keith J., *Why Indexing is the Wrong Way to Think About Analog Retrievals*, In proc. Case-Based Reasoning Workshop, Sponsered by DARPA-ISTO, p.41-44, May 1989.

Thornton, Charles H., and Lew, I. Paul, *Concrete Design and Construction*, in Frederick Merritt (ed.), Standard Handbook for Civil Engineers, Third Ed., Section Eight, New York, N.Y.: McGraw-Hill, 1983.

Ulrich, Karl, Computation & Pre-Parametric Design, Cambridge, MA: MIT-AI Lab, AI-TR 1043, 1988.

Veth, Bart, An Integrated Data Description Language for Encoding Design Knowledge, in P.J.W. ten Hagen & T. Tomiyama (eds.), Intelligent CAD Systems I: Theoritical and Methodological Aspects, Berlin, Germany: Springer-Verlag, EurographicSeminars, p. 295-313, 1987.

von Neumann, John, *The Computer and the Brain*, New Haven, CT: Yale University Press - Silliman Milestones in Science, 1958.

von Wright, Georg Henrik, *On Causal Knowledge*. In Ginet, Carl, & Shoemaker, Sidney, (eds.), *Knowledge and Mind - Philosophical Essays*, Oxford, U.K.: Oxford University Press, 1983.

Waltz, David L., *Generating Semantic Descriptions from Drawings of Scenes with Shadows*. AI TR-271. Cambridge, MA: MIT - AI Lab, November 1972.

Waltz, David L., *Is Indexing used for Retrieval?*, In proc. Case-Based Reasoning Workshop, Sponsered by DARPA-ISTO, p.41-44, May 1989.

Warren, D.H.D., *An Abstract Prolog Instruction Set*. Tech. Note 309, SRI International, Menlo Park, CA, 1983.

Winston, Patrick H., *Artificial Intelligence*, 2nd Edition, Reading, MA: Addison-Wesley, 1984.

Winston, Patrick H., & Horn, Berthold KP, *LISP*, 3rd Edition, Reading, MA: Addison-Wesley, 1989.

Winter, George & A. Nilson, *Design of Concrete Structures*, New York: McGraw-Hill, 1983.

Yip, Ken Man, *A Computer Program that Autonomously Explores Dynamical Systems*, Cambridge, MA: MIT Ph.D. thesis, 1988.

Appendix A
Π-Theorem

This appendix briefly illustrates the theoritical basis upon which the dimensional analysis is built. Dimensional analysis forms dimensionally-homogeneous groups, which are central to hypothesizing new quantitative relationships. These new quantitative relationships are a major component of the *harvesting* mechanism for improving the model of an evolving domains. The whole technique is based upon the Π-theorem (Bridgman 31).

If we have a complete[1] equation $\varphi(\alpha,\beta,\gamma)=0$ with (n) variables and (m) fundamental dimensions, then its solution has the form:

$$F(\Pi_1,\Pi_2,\ldots)=0$$

where the Π's are the n-m products of the arguments α, β, ... etc. which are in the fundamental units (Bridgman 31).
The typical Π has the form:

$$\Pi = \alpha^a \beta^b \gamma^c \ldots$$

where a, b, c, ... etc. are chosen such that P is dimensionless. Substituting the fundamental dimensions[2] (units) (U_1, U_2, U_3,...) for the variables $\alpha, \beta, \gamma, \ldots$, we get as example:

$$\alpha = U_1^{\alpha_1} U_2^{\alpha_2} U_3^{\alpha_3} \ldots U_m^{\alpha_m}$$

where $\alpha_1, \ldots \alpha_m$ are the dimensions (or the exponents) of the variable α in the fundamental units.

Thus, we get (m) equations, each with (n) terms, i.e., unknowns:

$$\left\{\begin{matrix} \alpha_1 a + \beta_1 b + \gamma_1 c + \ldots = 0 \\ \alpha_2 a + \beta_2 b + \gamma_2 c + \ldots = 0 \\ \ldots \\ \alpha_m a + \beta_m b + \gamma_m c + \ldots = 0 \end{matrix}\right\} \quad \ldots \quad (1)$$

In general n will be greater than m. Thus there will be n-m independent sets of solutions, i.e., there will be n-m independent dimensionless products and the arbitrary

1 A complete equation is an equation independent of the units used to measure its dimensions.

2 A possible set of fundamental dimensions may be composed of mass (M), length (L), and time (T).

function F will be a function of n-m variables. The above-mentioned equations can be solved simultaneously by the use of any algebraic techniques.

A.1. Conditions

1. The theorem, which is the core of the dimensional analysis, is based upon the principle of dimensional homogeneity. This principle restricts the possible candidate relationships between the variables under consideration very effectively. It may be considered a good way to limit the combinatorial explosion of the candidate relationships.

2. The fundamental units (dimensions) should be independent. This condition could be checked by getting a non-vanishing determinant of exponents.[3] Therefore, a check of the determinant of exponents is very essential at the beginning of the analysis. If we get a zero value for the determinant, we should revise our set of fundamental units before applying the theorem. If we cannot change the set fundamental units, then we have to form the dimensionally-homogeneous groups manually.

A.2. Limitations of P-Theorem

$\varphi(\alpha, \beta, ...) = 0$ is assumed to be the only relationship between the variables included in it. Otherwise, the partial differentiation used in the derivation of the theorem does not hold. Such assumption may hinder the approach from expressing the multi-phased relationships. We can override this assumption by the use of binary variable (0, 1) to help divide the domain of the problem into several sub-problems according to its phases.

3 The determinant of exponents is formed from the last set of equations (1).

Appendix B
Example on Harvesting

The example is from the area of design of asphalt roads. It assumes the development of longitudinal cracks in a stretch of a highway. Thermal and traffic impacts are suspected to be the cause. The data for the example is taken from (el-Shafei 86). This is an example for the use of quantitative discovery capability of the system in improving the model of failure mechanisms in pavement. Quantitative discovery is the collective name for the hypothesis capability using dimensional analysis and stepwise regression analysis.

To demonstrate the hypothesis capability of the system, we assume that there is neither a sufficiently-similar case in teh case library, nor a relationship that predicts the development of logitudinal cracks.
The system picks the relevant parameters from a large number of parameters. The criterion for picking up the parameters is their existence in the *CAUSED-BY* and *MAYBE-RELATED-TO* fields in the state frame in the knowledge base. The system backward-chain through the semantic network till it reaches a fact or an input.

Input:
We use an exemplary case which has only one manifestation of distress, longitudinal crack. For more details about the case, refer to the case file in figure (B.1).

The only symptom mentioned in the input case file is longitudinal cracking. Therefore the state frame LONG-CRACK is triggered. The backward chaining through the causal network triggers the other state frames including the computational frames shown hereafter:

```
State:                          EQUIVALENT-THICKNESS
STATE-TYPE:                     COMPUTATIONAL
CLASS:                          STRUCTURAL
REPRESENTATIVE-VAR:             H
VARIABLE-DIMENSIONS:  M(1), L(0), T(-2)
VALUE:                          NIL
EQUALS:                         (+ (* SURFACE-STIFFNESS  SURFACE-THICK)
                                   (* BASE-ELAST-MOD.  BASE-THICKNESS)
                                    (* SUBGRADE-MOD.  SG-EQUIV-THICK))
```

State:	SURFACE-HEAT
Case:	Route-495, Milepost 125-130
Type-of-pavement:	Flexible
AADT:	15,000 beh.per lane
%-trucks:	24
Ave-speed-mph:	49
lane-width-ft:	12
#-of-lanes:	3
shoulder-width-ft:	8
Pavement-thick-inch:	8
Base-thickness-inch:	15
CBR:	3.0
Temperature-variation:	((summer 85) (fall 60) (winter 30) (spring 50))
Subgrade-texture:	Gravel
Dry-density-pcf:	135
Corrected-CBR-%:	(86 37 5)
Surface-stiffness-ksi:	120
Base-elast-mod-ksi:	20
Subgrade-mod-psi:	1500

<u>Symptoms</u>	<u>Severity</u>
Longitudinal-crack	6 ft

Figure (B.1): A Part of A design Case of a highway

STATE-TYPE:	COMPUTATIONAL
CLASS:	ENVIRONMENTAL
REPRESENTATIVE-VAR:	SH
VARIABLE-DIMENSIONS:	M(0), L(2), T(0), DEG(1)
VALUE:	NIL
EQUALS:	(* AVE-ANN-TEMP #-LANES LANE-WIDTH 5280)

State:	ACCUM-LOAD
STATE-TYPE:	COMPUTATIONAL
CLASS:	TRAFFIC
REPRESENTATIVE-VAR:	X
VARIABLE-DIMENSIONS:	M(1), L(1), T(-2)
VALUE:	NIL
EQUALS:	(* AADT 2 EQUIV-AXLE-LOAD)

B.1. Hypothesizing:

* The Heuristic Pre-Processor (HPP) classify the input parameters into two groups, a structural group and an environmental group. Still the groups may contain irrelevant parameters.

The dimensional analysis unit forms, at least one, dimensionallyhomogeneous group out of every heap classified by the HPP. Thus we get one environmental (thermal) dimensionally-homogeneous group, and two dimensionally-homogeneous structural groups, one of which has the REPRESENTATIVE-VARIABLE of longitudinal-crack isolated and raised to the unary power. This latter group will be in teh left-hand side of the proposed relationship.

The stepwise regression analysis takes the values of the current case as an instance to be added to the other instances in the case library. Assume that the case library had 11 relevant cases before. Thus the stepwise regression analyzer starts working on 12 cases. It builds the relationship gradually, in two stages. It shows us that the thermal properties have the major impact upon the development of longitudinal cracks. In the second degree of importance, comes the structural properties.

Longitudinal Cracking due to Thermal & Traffic Impacts:

Variable \ Unit -->	. m	. l	.. t	. deg	
H	1	0	-2	0	Equiv-Thickness (ksi.in)
A	0	0	1	0	Asphalt-Age (years)
S	1	-1	-2	0	Strength-of-Asphalt (ksi)
T	0	2	0	1	Surface-Heat (1000 deg. sq ft)
C	1	0	-2	-1	Sect-Heat-Capacity (ksi.mi/deg.)
X	1	1	-3	0	Accum-Load/yr (1000 ton/yr)
L	0	1	0	0	Accum-Length-of-Cracks (ft)

Number of arguments:	7
Dimensional analysis:	yes
Number of fundamental units:	4
Polynomial Expansion:	no
Minimum enhancement:	1 %
Case title:	Longitudinal cracking due to thermal and traffic impacts
Number of dimensional groups:	3

Output:

Dimensional Group # 1 : $X_1 = H^{-3} A^0 S^2 T^1 C^1 X^0 L^0$

= *The thermal impact term.*

Dimensional Group # 2 : $X_2 = H^{-2} A^1 S^1 T^0 C^0 X^1 L^0$

= *The strucutural term.*

Dimensional Group # 3 : $Y = H^{-1} A^0 S^1 T^0 C^0 X^0 L^1$

= *The LHS including the parameter 'under consideratio with a unary power of exponentiation.*

Data Set:

H	A	S	T	C	X	L
15	2	120	-2	5.1	250	7
14	12	100	6	6.7	90	15
18	5	130	20	5.8	200	25
14	4	90	55	7.4	60	40
11	3	110	35	5.6	200	20
16	6	125	45	6.9	150	38
20	5	122	0	5.8	70	5
25	4	115	-5	6.2	80	0
10	9	102	22	6.5	350	45
12	7	95	33	7.1	380	55
16	6	140	53	7.2	200	30
12	3	190	57	7.5	290	58

The relationship:

(1) $Y = 122.166 + 0.0001 X_1$

Correlation = 0.820

(2) $Y = 61.740 + 0.0001 X_1 + 0.0001 X_2$

Correlation = 0.9397