Logic-Based Conceptual Structural Design of Steel Office Buildings

Deepak Jain
Helmut Krawinkler
Kincho H. Law

TECHNICAL REPORT
Number 49

May 1991

Stanford University

If you would like to contact the authors please write to:

c/o CIFE, Civil Engineering,
Stanford University,
Terman Engineering Center
Mail Code: 4020
Stanford, CA 95305-4020

Abstract

This research effort addresses automation of *conceptual* design of structures and development of knowledge-based computer tools that assist an engineer in the preliminary phase of structural design. The effort is aimed at using symbolic logic to develop a knowledge-based conceptual structural design application, while employing and extending a formal methodology for developing such systems. The methodology and its application to the domain of structural design of steel office buildings is presented in the dissertation. Issues in the use of symbolic logic for building design systems are discussed and some observations based on our experience are offered.

A computer system named *Galileo*, capable of (i) generating floor framing plans for transfer of gravity loads and (ii) designing perimeter moment resisting frames for transfer of lateral loads, has been developed as part of this investigation. Given the architectural plan of a building, Galileo can generate geometric configurations for locating structural elements and can design individual members while satisfying structural and exogenous constraints. Costs of different solutions can also be estimated. Epikit—a tool that uses a first-order predicate calculus based language, KIF, for representing knowledge—has been employed for developing the system.

Knowledge and reasoning behind Galileo are elucidated in the dissertation and implementation details are discussed. Representative examples are used to illustrate the working of the system

Acknowledgments

This report is reproduced from a doctoral dissertation submitted by Deepak Jain to Stanford University. The doctoral committee comprised of Professors Helmut Krawinkler, Kincho H. Law, Michael R. Genesereth, and Raymond E. Levitt. Financial support of the Center for Integrated Facility Engineering (CIFE) at Stanford University for this research is gratefully acknowledged. Computing facilities for the work were provided by the Civil Engineering Department.

We wish to thank Gregory P. Luth for providing much of the domain expertise for this study. In addition, we are also grateful to Mr. Roger E. Ferch of Herrick Corporation for providing input in the cost estimation process, and to Dr. Narinder Singh for assisting with Epikit and KIF. The library of standard core configurations contained in Galileo is adapted from the *CE 214 Expert Systems in Civil Engineering*, Spring 1988, course project of Charles Fell, Cristina Garcia, and Gregory Luth.

Contents

List of Tables

List of Figures

1 Introduction

1.1 Objectives

The field of structural engineering has come a long way since Galileo first systematically studied the strength of materials and behavior of cantilever beams. Since the publication of Galileo's *Two New Sciences* over 350 years ago, major advancements have been made in the areas of structural analysis and design. With the advent of computers, significant research effort has been directed towards automating analysis and design of structures. Computer-aided structural design, in particular, has received considerable attention in the last decade.

The conceptual design of structures, wherein the designer investigates many potential alternatives and makes fundamental choices that have major impact on the downstream decisions, is one of the important areas for investigation from the standpoint of automation. Several works in this direction have been undertaken recently or are currently underway (e.g., [2,21,29,37,45,49,57,60,80,81]). It is generally recognized that heuristics play an important role in the creative process of conceptual structural design. Another characteristic of conceptual design, arising from economic considerations, is that typically several feasible design solutions are developed and evaluated.

The two primary forms of representation of knowledge in conceptual design systems are *procedural* and *declarative*. Because of its virtues of extensibility and versatility, there seems to have evolved a consensus amongst researchers in favor of declarative representation for developing computer aids for structural design, particularly since many structural design tasks are not easily amenable to algorithmic solutions. However, no consensus on a global framework for these systems has emerged. Luth [57] has recently proposed a global framework for developing conceptual design systems in structural engineering. The framework, however, needs to be tested and extended (wherever necessary) through building specific applications with well-defined foci.

There are several approaches towards representation and reasoning in a declarative fashion, including semantic nets, frames, rule-based reasoning, inference nets, etc. There have been many previous efforts in the domain of civil engineering where these representation schemes have been employed (e.g., [9,39,54,60,76]). Another promising scheme for representing and reasoning about knowledge, though not as extensively employed for building civil engineering systems as some of the other schemes, is symbolic logic. Implementation of a prototype conceptual design application in a logical language like first-order predicate calculus can give us insights into the strengths and weaknesses of a logical environment for the purposes of developing substantial applications in structural engineering in the future.

The twin objectives of this study, then, are:

1. to employ and extend a formal methodology for automating conceptual structural design, by developing a prototypical application to perform conceptual structural design of steel office buildings, and

2. to investigate issues in the use of symbolic logic for building conceptual structural design systems.

Accordingly, this dissertation presents:

- a formal methodology for automating conceptual structural design, based on and extended from the work in Ref. [57];

- application of the methodology to the domain of conceptual structural design of steel office buildings; and

- description of and observations on the use of logic for implementing the said application.

1.2 Scope

A structural design project is conceived when there is a perceived need for a constructed facility. In the case of commercial buildings, typically the owner is interested in a facility that can yield tangible monetary benefits. The stages that the project goes through from the perspective of the structural engineer usually involve program development, conceptual design, detailed design, construction planning, construction execution, and facility management [57]. For the purpose of this research, we have concentrated on the *conceptual design* stage of the process. The implementation, in particular, is concerned with the conceptual structural design of multistory steel office buildings. There are additional restrictions on the geometry of buildings that can be handled by the prototype as described elsewhere in this dissertation.

Several factors influenced the selection of the domain for the prototype. Ready access to an expert with a substantial field experience in designing multistory office buildings was one. In addition, structural design of multistory commercial office buildings is a complex, but well-constrained problem, and thus quite manageable from the viewpoint of automation. Exogenous constraints are quite conspicuous in these types of structures and their use can be easily illustrated. The domain of steel design, as opposed to concrete design, is discrete since there are standardized sections available for use. This is advantageous while generating design solutions.

1.3 Organization of the Dissertation

Eight chapters following this one present the main substance of this dissertation. The next chapter elaborates on some background information and provides a context for discussion in the remainder of this dissertation. Prior research in the area of knowledge-based systems for structural design is examined and avenues of improvements in the prior work are suggested. These avenues serve as part of the motivation for this investigation. Chapter 2 also points out some similarities and differences between design in structural engineering and design in other engineering disciplines. The logic-based approach to artificial intelligence (AI) is introduced, followed by a summary of the results of a literature review of gravity and lateral load resisting systems (hereinafter referred to as gravity system and lateral system, respectively) for steel office buildings. A brief note about the sources of information used in this study concludes the chapter.

Chapter 3 presents the details of the methodology employed in this investigation. The methodology is intended to be general and applicable to other types of structures besides buildings. The reasoning in a conceptual structural design system, and the various kinds of knowledge needed to support such reasoning, are identified.

Chapters 4 and 5 describe application of the methodology to the design of gravity and lateral load resisting systems, respectively, for steel office buildings. This is done through elucidation of knowledge and reasoning behind a computer system, *Galileo*, the conceptual design system developed as a part of this investigation,.

A discussion of implementation issues follows in Chapter 6. The chapter discusses the implementation environment for Galileo, along with some user interface issues. Two examples in Chapter 7 illustrate the working of Galileo. A few observations in the penultimate chapter summarize our experience with logic for developing a prototypical structural design application. The final chapter offers some concluding remarks and directions for future work.

Three appendices supplement the subject matter of the chapters. Appendix A describes the syntax and semantics of propositional and predicate calculus, and exemplifies their usage with some engineering applications. Appendix B presents details of the KIF (Knowledge Interchange Format) language that is employed in the implementation. Appendix C provides a detailed description of the working of Galileo.

The fact is that civilization requires slaves. The Greeks were quite right there. Unless there are slaves to do ugly, horrible, uninteresting work, culture and contemplation become impossible. Human slavery is wrong, insecure, and demoralizing. On mechanical slavery, on the slavery of the machine, the future of the world depends.

— *Oscar Wilde,* The Soul of Man under Socialism *(1895)*

Machines—with their irrefutable logic, their cold preciseness of figures, their tireless, utterly exact observations, their absolute knowledge of mathematics—they could elaborate any idea, however simple its beginning, and reach the conclusion. Machines had the imagination of the ideal sort—the ability to construct a necessary future from a present fact. But Man had imagination of a different kind; the illogical, brilliant imagination that sees the future result vaguely, without knowing the why, nor the how; an imagination that outstrips the machine in its preciseness. Man might reach the conclusion more swiftly, but the machine always reached it eventually, and always the right conclusion. By leaps and bounds man advanced. By steady irresistible steps the machine marched forward.

— *John W. Campbell, Jr.*

2 Prologue

This chapter introduces some preliminary concepts and the background behind this research. In particular, the next section presents a review and some observations on prior work in knowledge-based systems for structural design. A comparison with design in other engineering disciplines follows in Section 2.2. Section 2.3 provides a brief overview of the logical approach to AI and some logic-based design systems. Summary of a literature review of lateral and gravity load resisting systems for steel office buildings follows in Section 2.4. A brief note about knowledge acquisition concludes the chapter.

2.1 Research in Knowledge-Based Systems for Structural Design

There have been many applications of knowledge-based systems in the domain of structural engineering. Several efforts have been undertaken since the early days of SACON [8], including those cited in Chapter 1. In Section 2.1.1, a few such relevant prior design systems are discussed and their limitations are mentioned. (Chapter 4 and Appendix A contain additional comments on previous works vis-a-vis pertinent subproblems.) Observations that are applicable in general to most of the earlier systems are summarized in Section 2.1.2 along with arguments for an improved approach for developing structural design systems.

2.1.1 Prior Research

One frequently cited structural design system is HI-RISE [60], a system that generates and evaluates preliminary structural designs for high-rise buildings. From a given input of structural topology, geometry, locations of service shaft and mechanical floor, occupancy, wind load, and live loads, HI-RISE generates a context tree where each path through the tree represents a complete, feasible structural design alternative. The design process is divided into two major tasks that are carried out in the following order: (i) design of lateral load resisting system, and (ii) design of gravity load resisting system. HI-RISE draws all its knowledge from the literature, whereas in Galileo, the computer system developed in this work, human experts have also contributed to knowledge. Galileo is also more focussed (with respect to building occupancy, material, and possible lateral systems), which results in generation of more realistic solutions for a particular subclass of problems. Such tasks as geometric configuration to determine the location of gravity load resisting systems are performed by Galileo; in contrast, information about structural topology is required as input for HI-RISE. The criterion used for evaluation of alternatives is also different and more practical in Galileo. On the other hand, the top-level decomposition of the structural design process (into lateral and gravity systems) used by HI-RISE has been retained in this work.

Sriram [80] proposed a conceptual model for integrated structural design in DESTINY by describing several knowledge modules, and an abstraction hierarchy of objects to facilitate communication amongst them through a blackboard. One knowledge module of DESTINY that was implemented was ALL-RISE, an extension of HI-RISE to include the capability of preliminary synthesis of low- and medium-rise buildings. Some of the observations made in the context of HI-RISE are applicable to ALL-RISE as well. Furthermore, Galileo employs a logical framework instead of a blackboard architecture for implementation. The constraint classification used in our study is based on Ref. [57], which is richer and more comprehensive.

Gero and colleagues [29,28] have suggested prototypes as a conceptual schema for representation of generalized design knowledge. A prototype provides a vocabulary of design elements, the intended interpretations (goals and requirements), knowledge about the vocabulary and interpretations, and parameterized design descriptions or parameterized design description generator. However, the knowledge that is proposed to be included in such prototypes is shallow heuristic knowledge (e.g., for a rigid frame, "IF lateral-node-load < 50 kips AND no-of-stories > 50 THEN material is steel ELSE concrete"). The selection of appropriate prototypes to be examined (in a given design context) in such a manner adversely affects extensibility of the system and results in brittleness.

Kumar and Topping [49] have discussed issues involved in the development of knowledge-based systems for detailed design of structures. They argue for representation of (i) theoretical knowledge about material properties and (ii) behavior of different types of structures under different types of loading conditions, so that when heuristic knowledge fails, the system can fall back on to fundamental principles. A system, DESDEX, part of a larger system (INDEX) for design of industrial buildings, is used to illustrate the feasibility of such an approach. The authors observe that reasoning from fundamental principles, though feasible, is time consuming, and suggest "intelligent" problem-solving strategies to achieve speed-ups in run time.

An integrated building design environment (IBDE) is being developed by Fenves et al. [21] at Carnegie-Mellon University. The major thrust of the work is in building an environment of processes and information flows to automate communication of design decisions in electronic form. The issue of the nature and contents of a global database needed for computer-integrated construction is also being investigated. The approach to vertical integration in IBDE involves sequential operation of several knowledge-based systems to perform architectural design, structural design, construction planning, and other such tasks involved in the design and construction of office buildings. Output of an upstream system serves as input to the following system in the sequence, which operates and performs the specified task, feeding the resulting information to the next system (if there is one). Of course, integration of various tasks is very important, but, in addition, integration through incorporating downstream considerations and accounting for effects of any decision on subsequent tasks is also desirable. IBDE plans to address this issue through process critics.

In a recent work, Sause and Powell [77] propose an organizational model, termed *multilevel selection-development* model, of the structural design process. A design problem is decomposed into selection and development subproblems at alternate levels of an hierarchy in this model. One of the limitations of the model is that the behavior of the overall system gets lost in the decomposition process. Also, the model sets up subproblems for the design of each individual component, irrespective of the fact that in a typical structure several identical instantiations of any component exist. Hence, even when several beams in a building are identical in every respect (e.g., span, loading and support conditions), design of every single one of them will be a subproblem in the model. Unless there is a method of aggregation of subproblems for identical components, design of large-scale structures will be unnecessarily repetitive and time consuming.

A system to decide on certain design variables (e.g., the construction material and the bay sizes), given a qualitative descriptions of labor costs, schedule requirements, and regularity of the floor shape, in addition to other input such as soil bearing capacity, cost of land, etc., has been developed by Haber and Karshenas [33]. The system works by searching from sets of values for different parameters so that the overall solution is optimized. Thus bay size is chosen from a predefined set that includes values like $15' \times 15'$, $20' \times 20'$, $30' \times 30'$, and $40' \times 40'$. To extend the system one will need to generalize the decision process so that values of design variables are not restricted to be members of predefined sets.

In a work closely related with the work described in this dissertation, Luth [57] has proposed a global framework for reasoning and representation for integrated structural design. He argues for explicit representation of form, function, and behavior in structural design systems and describes structural and exogenous constraints (organized in a formal constraint classification system) applicable to design of commercial office buildings. We utilize the same constraint classification system in this study and extend the work by applying the concepts to a specific problem. Ref. [57] is thus frequently cited in this dissertation.

In addition to the systems described in this section, analogical reasoning to develop designs based on previous cases is also being examined. Zhao and Maher [89] and Howard et al. [37] have reported successful applications of case-based reasoning in structural engineering. Recently, application of logic to structural engineering has emerged as an active research area and applications to some subproblems have been reported in the literature. Such applications are discussed later in this chapter in Section 2.3.1, along with application of logic to design in other engineering disciplines.

2.1.2 Arguments for Improved Approach

Heuristics have received considerable attention in previous works for automating structural design. Undoubtedly, they are indispensable in several specific instances, more so in the conceptual phase of structural design. However, first principles can often be used equally well while generating design solutions, and they have the added benefit of resting on a firmer basis. Though employing rules of thumb may be efficient for human experts whose time is precious, in an automated environment there is little to be gained by applying shallow heuristic knowledge when a slightly more detailed investigation based on laws of nature will yield better results. An example supporting this argument is presented later in Section 3.3. Even when heuristic knowledge is required, its place has to be properly identified in the overall framework. Also, for the system to have a practical outlook, the heuristics should preferably be obtained from human experts also in addition to text books.

Before embarking on implementation of a system meant for conceptual design, it is useful to develop a formal model of the domain as well as the design process itself. Lack of a formal model can contribute to brittleness and lack of extensibility of the system. A related issue here is that of representation of function. As pointed out by Luth [57], the reason for the existence of structure—to carry loads—is usually not represented explicitly in previous systems. Knowledge of the form "If the building is less than 20 stories then use a moment resisting frame" implicitly incorporates several facts, including that the lateral loads will exist on the structure and they need to be transferred to the ground through some elements and systems. As is demonstrated in Section 3.2, reasoning based on an explicit representation of function is much more general and can lead to systems that are easier to extend beyond their original application.

The most important criteria governing structural design are safety and economic efficiency. To that end, the designer strives for strength, physical integrity, and serviceability of the structure (to some extent, by following the provisions of codes of practice). However, issues like architectural requirements, MEP (mechanical, electrical, and plumbing) constraints, constructibility considerations, etc., that can impose substantial restrictions on the structural system and significantly affect its cost, also need to be addressed during the structural design process. Some of the earlier systems have primarily been concerned with structural considerations alone at the expense of exogenous ones. Integrated structural design aims to develop more practical design solutions by devoting the attention merited by such exogenous considerations, in addition to the structural ones, during the process of structural design.

2.2 Design in Other Engineering Disciplines

There are several interesting similarities as well as differences between the design process in structural engineering and design processes in other engineering disciplines like mechanical and electrical engineering. Design in other disciplines also involves more than one phase and there is usually a preliminary design phase (where some high-level decisions are made) followed by a detailed design phase (where finer-level details are sorted out). The idea of "cooperative design" is applicable to all design disciplines where many interacting—and possibly conflicting—entities are involved in design. Structural design is no exception since, in addition to the structural engineer, the owner, the architect, the mechanical engineer, etc., influence the structural design. Also, the design is usually meant for artifacts that are ultimately to be built, and incorporating downstream considerations at the design stage can can have significant favorable consequences for the production stage. Thus, we have the concept of *design for constructibility* in the case of structures and analogous concepts of *design for assembly* and *design for manufacturability* in the case of electrical and mechanical systems [85,22]. Moreover, design usually is an open-ended problem and multiple solutions are possible. Thus, some evaluation criterion is needed to judge the overall merit of the different alternatives.

One important aspect in which structural design differs from design in other disciplines is that structures are usually 'one-of-a-kind' systems [57]. Full-scale physical modeling of the artifact, which is economically viable in other disciplines because of the large production volumes, is usually impractical in the case of structural systems. The design of structures, thus, is most commonly based entirely on non-physical models.

In Ref. [16] Dym and Levitt have discussed application of knowledge-based systems to engineering design in general. A few engineering design systems that use logic are discussed later in this chapter.

2.3 Logic

Logic is a formalized treatment of knowledge and thought. To use logic as a knowledge representational formalism, one starts by asserting some *axioms*, which embody known facts or self-evident truths in a particular field of interest, in a chosen *language*. The fundamental unit of representation in the chosen language is a sentence. New, useful sentences are inferred from such axioms through syntactic manipulations, without any reference to the meaning of the symbols involved. This is important because a computer can be programmed to carry out syntactic manipulations, and depending upon the range of these manipulations and their correspondence to the spectrum of inferences made by human beings, one may enable a computer to emulate some part of the human thought process.

For reasons of ambiguity and imprecision, natural languages, such as English, are not appropriate for expressing the knowledge contained in the aforementioned axioms and drawing inferences from them. Over centuries, logicians have tried to devise languages that possess sufficient expressive power and are more precise than natural languages. Propositional calculus is one such language of abstract sentences that consist of *propositions* (truth symbols like `true` and `false` and propositional symbols) and *connectives* (for instance, *and*, *or*, *if-and-only-if*, etc.). However, this language is too coarse and primitive to express the concept of an object, properties of an object, or a relationship between several objects [61]. A more powerful logical language is predicate calculus. Appendix A describes both the languages and provides some examples of their possible uses in structural engineering problems.

The logical approach to AI is often mistakenly equated with pledging allegiance to the use of first-order predicate calculus as the representation formalism. That is a rather narrow interpretation of the logicists' position. Logic is not a particular syntax or a technique of coding; instead, it is a collection of ideas and rigorous mathematical tools for expressing knowledge about the world and analyzing the representational languages capable of doing so [35]. A more accurate characterization of the logicists' stance will involve (i) commitment to *declarative* representation of knowledge in general, (ii) with the use of any language that is at least as expressive as first-order predicate calculus [69]. Many researchers have defended both these tenets in the past, and there is considerable agreement even among "non-logicists" vis-a-vis the supremacy of declarative representation for building intelligent machines. Logicists furthermore believe that parts of the theoretical apparatus already developed in mathematical logic (e.g., proof theory and model theory) are necessary ingredients for the foundation of intelligent systems (see, for example, Refs. [35] and [67]).

Logic is not in contradiction with the *knowledge is power* [53] principle. Of course, one must have adequate domain-dependent knowledge in order to solve a problem. Logic provides only the *form* and not the *content* [69]. Adoption of the logical approach by no means rules out the use of heuristics in the design process. Heuristics can be expressed using predicate calculus as well as any other declaration.

In this investigation we have employed the KIF representation language (which is an extension of first-order predicate calculus) for formalizing the knowledge in the domain of structural design. (An overview of KIF is presented in Appendix B.) Based on our experience, some observations on the logical approach in the context of structural design applications are presented in Chapter 8.

2.3.1 Logic-Based Design

Use of logic for developing structural design systems has not been as extensive as use of some other representation formalisms. Logic-based design, both in the domain of structural engineering as well as other engineering disciplines, is reviewed in this section.

The application of symbolic logic to design axioms has been discussed by Kim and Suh [47]. Modeling the design process as a mapping from a set of functional requirements into a set of design parameters, this work illustrates how generalized design principles can be represented through symbolic logic axiomatizations. As a result, some design rules are shown to be direct implications of basic design axioms coupled with certain assumptions.

In one of the early uses of logic for structural design, Chan and Paulson [10] showed how constraint formulation, propagation, and satisfaction (using logic programming) can be applied for design of determinate planar truss structures. Constraints are represented as procedural attachments in this work, with a different procedure representing each possible use (e.g., conformance checking or variable instantiation) of the constraint, a practice not in strict conformance with the ideals of declarative programming. One of the primary limitations of the work is that constraints have to be manually formulated by the user and provided as input, which is cumbersome and unnecessary since a system with knowledge about structural behavior and performance can formulate some of the constraints automatically at run-time.

An application of formal logic to architectural design is reported by Coyne [14]. The work described can perform spatial layout of functional spaces in a building, and illustrates the use of deductive reasoning for complex design tasks. The system has been implemented in Prolog, a language that only partially exploits the expressibility and inference capabilities of predicate logic. Hence the knowledge representation scheme is essentially production systems modeled in logic. The work shows that important aspects of design can be modeled as logical processes; judgments often considered intuitive and instinctive can be simulated mechanically through axioms of a logical system.

In a recent application, Lakmazaheri [51] shows how a constraint logic approach can be used for design of two-dimensional trusses. One of the tasks in the process is *partial synthesis*, wherein locations of various components (such as members and supports) are determined to synthesize trusses, given loading information and fixed locations of other components. Effectively, the user defines the complete search space by identifying all possible locations of the nodes and the precise numbers and types of various structural components that should be used in the final structure; the synthesis is then a search for all possible configurations that result in stable structures while satisfying other applied constraints. Application of theorem proving to analysis and sizing of truss structures is also described in this work.

Another work, currently in progress, is *Designworld* [25], which envisions development of a computer-robotic system to assist in the production of small-scale electromechanical devices, such as disk drives, compact disc players, and robots. For reasoning, Designworld will work from a declarative representation of fundamental knowledge in the relevant design disciplines to achieve high-level performance while avoiding the brittleness often encountered in traditional expert systems. One key concept in the project is contemplation of a central database that will include product-specific information (such as manufacturing records, specifications, assembly plans, etc.) and product-general information (such as basic scientific and technological principles in electrical and mechanical engineering, details of the machinery available for manufacture and maintenance, etc.). As the reader will notice later, the methodology discussed in Chapter 3 has components that correspond well with these kinds of knowledge.

2.4 Gravity and Lateral Systems for Steel Buildings

Disregarding the foundation issue, the problem of conceptual structural design of an office building involves (i) design of the gravity system and (ii) design of the lateral system. The gravity and lateral systems are not independent of each other and can mutually constrain each other's options. Thus, if the load resisting systems are designed sequentially in an automated environment, a reconciliation step should evaluate the influence of a gravity system's decisions on lateral system's ones (and vice versa), and arrive at an efficient combined system, conceivably by modifying one or both the subsystems.

Though lateral loads often govern the sizes of members in the case of high-rise buildings, in this investigation we proceeded to design the gravity system independently so that the "premium" for the resistance to lateral loads can be assessed. Accordingly, a review was made of the options available for steel structural systems for resisting both the types of loads and the results are presented in Figs. 2.1 and 2.2. The figures are based on the information contained in several sources, including Refs. [3,13,41,84,88].

Not all of the alternatives shown in Figs. 2.1 and 2.2 have been implemented in the prototype. For instance, only composite metal deck and wide-flange hot-rolled beams are available for floor slab and support element, respectively, in Galileo. Similarly, only perimeter moment resisting frames are used for the lateral system. The figures are based on literature survey and do not represent the model of gravity and lateral systems in Galileo. They are shown here for illustration because they served as a starting point for some of the reasoning employed in Galileo. Thus, the heuristics shown in the figures are the ones contained in the literature; Galileo does not reason from any of these heuristics.

2.5 A Note about Knowledge Acquisition

In addition to literature, knowledge for Galileo was also acquired through interviews with human experts. Though it is often difficult to pin down experts for extended periods of time, the author was fortunate to have nearly unrestricted access to a structural engineer

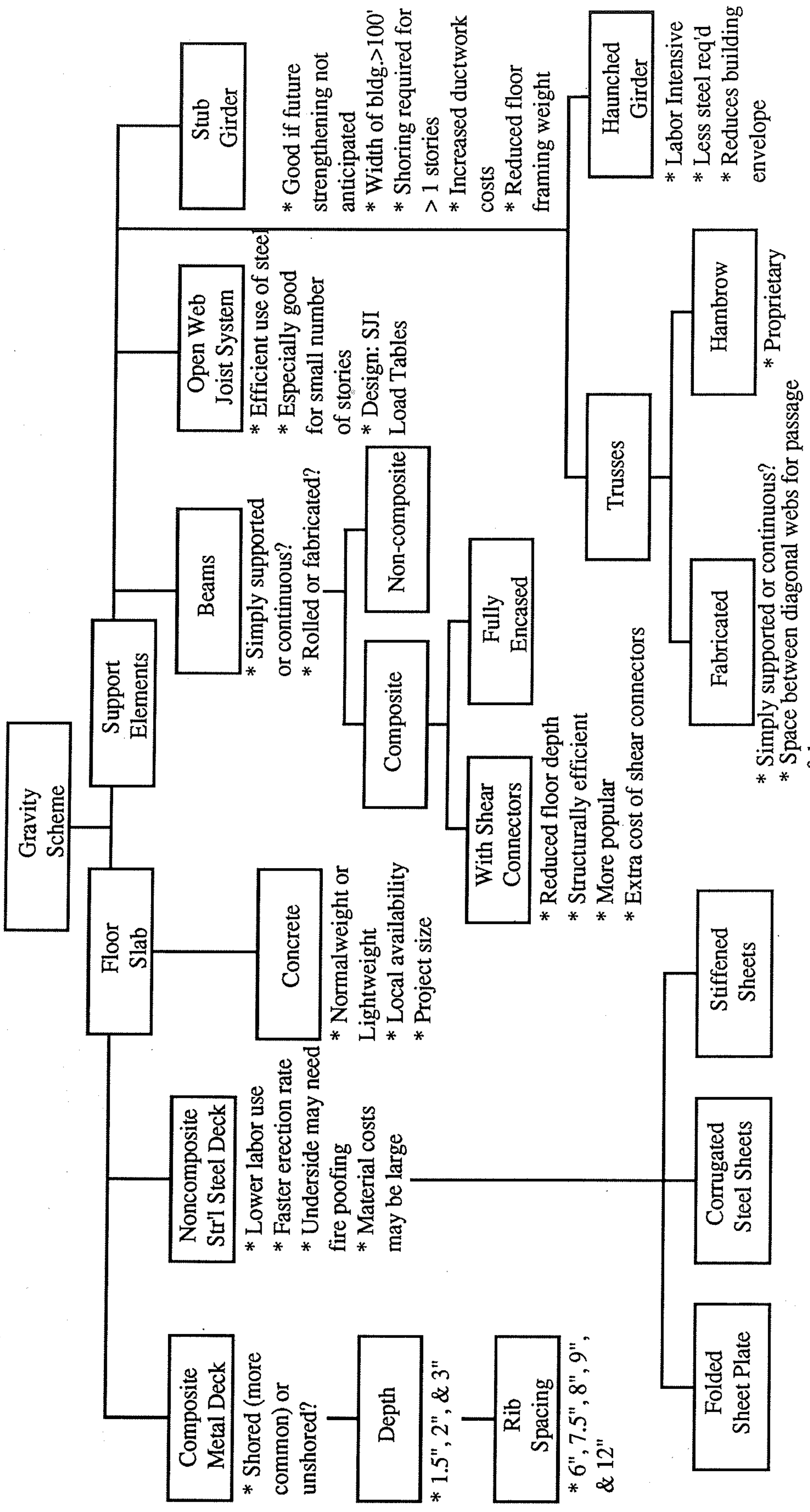


Figure 2.1: **Gravity System Alternatives.** Statements below the boxes summarize comments and choices involved, as obtained from references mentioned in Section 2.4.

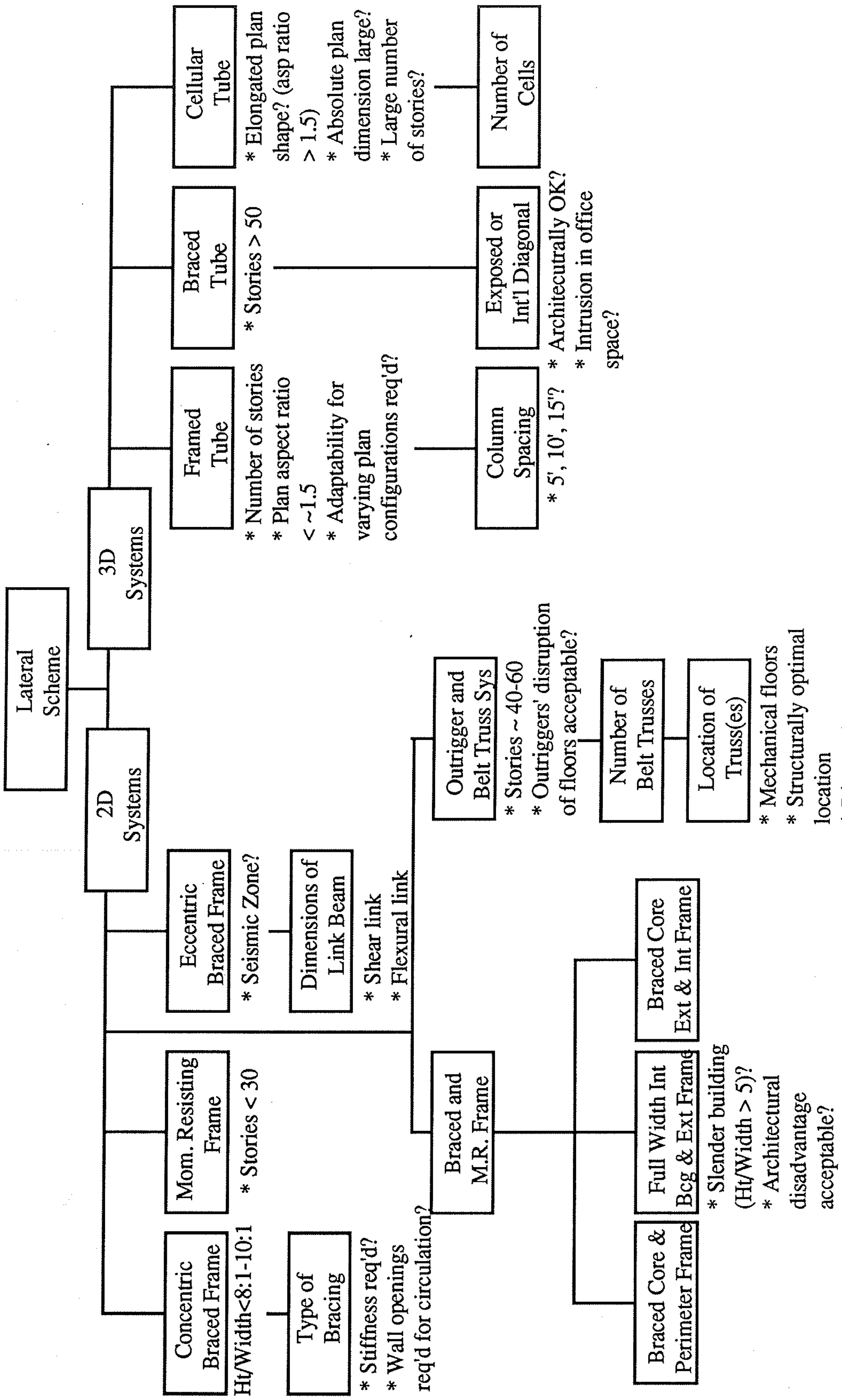

Lateral Scheme
2D Systems
3D Systems
Concentric Braced Frame
* Ht/Width<8:1-10:1
Type of Bracing
* Stiffness req'd?
* Wall openings req'd for circulation?
Mom. Resisting Frame
* Stories < 30
Eccentric Braced Frame
* Seismic Zone?
Dimensions of Link Beam
* Shear link
* Flexural link
Braced and M.R. Frame
Braced Core & Perimeter Frame
Full Width Int Bcg & Ext Frame
* Slender building (Ht/Width > 5)?
* Architectural disadvantage acceptable?
Braced Core Ext & Int Frame
Outrigger and Belt Truss Sys
* Stories ~ 40-60
* Outriggers' disruption of floors acceptable?
Number of Belt Trusses
Location of Truss(es)
* Mechanical floors
* Structurally optimal location
* Diagonal truss on perimeter acceptable?
* Employ Vierendeel truss?
Framed Tube
* Number of stories
* Plan aspect ratio < ~1.5
* Adaptability for varying plan configurations req'd?
Column Spacing
* 5', 10', 15'?
Braced Tube
* Stories > 50
Exposed or Int'l Diagonal
* Architecturally OK?
* Intrusion in office space?
Cellular Tube
* Elongated plan shape? (asp ratio > 1.5)
* Absolute plan dimension large?
* Large number of stories?
Number of Cells

Figure 2.2: **Lateral System Alternatives.** Statements below the boxes summarize comments and choices involved, as obtained from references mentioned in Section 2.4.

having considerable expertise and 14 years of experience in designing multistory office buildings. Knowledge acquisition is often the bottleneck of projects involving development of knowledge-based systems, and from personal experience the author can state that several conditions need to be met for successful knowledge acquisition. One of the most important is availability. It is not always possible to foresee all the aspects of a problem during an interview session, and incompleteness and conflicts in the acquired knowledge become apparent later on during implementation. If the expert can be contacted readily in such cases, the process of knowledge acquisition becomes much smoother.

In the case of this investigation, a confluence of factors presented an almost ideal opportunity for knowledge acquisition. The primary expert, Mr. Gregory P. Luth, was pursuing doctoral work alongside the author. It was possible to discuss the subject matter in detail with the expert in an informal setting fairly regularly and frequently. The meetings were usually not time-constrained. The expert could see the acquired (and implemented) knowledge being applied to problems and comment on its efficacy. In addition to having a wealth of practical experience, the expert was familiar with AI and knowledge-based systems concepts and terminology. The benefits of such familiarity cannot be overstated.

Another expert, Mr. Roger E. Ferch, provided much valuable expertise for the evaluation phase. Mr. Ferch furnished the basic data used for cost estimation and gave useful feedback regarding the presentation of results.

It makes no difference to a chess problem whether the pieces are white and black, or red and green, or whether there are physical pieces at all; it is the same problem which an expert carries easily in his head and which we have to reconstruct laboriously with the aid of the board. The board and the pieces are mere devices to stimulate our sluggish imaginations, and are no more essential to the problem than the blackboard and the chalk are to the theorems in a mathematical lecture.

— *G. H. Hardy,* A Mathematician's Apology *(1940)*

The fact that all laws and general propositions have their exceptions does not destroy the value of laws and generalizations as guides to human conduct—any more than the doctrine of justifiable homicide destroys the law against murder.

— *I. F. Stone,* The Trial of Socrates *(1988)*

3 Methodology

Musen [68] states that developing a knowledge system for performing an engineering task is, in many respects, analogous to developing a scientific theory: one must identify the underlying knowledge that defines the characteristics of the physical system being reasoned about, and one must also theorize the problem-solving approach of a given class of professionals in the domain of interest. Model-based reasoning has recently emerged as a useful paradigm for solving problems in diverse application areas [50,78]. In order to reason about a system in this approach, one explicitly represents the *form*, *function*, and *behavior* of the system being modeled, imparting flexibility and depth to the computer system.

The following definitions, adapted from Ref. [57], define the terms *form*, *function*, and *behavior* as seen from the perspective of structural engineering.

- *Form*: The form in the context of structures refers to the description of spatial arrangement of functional objects (such as beams, columns, etc.) and the physical attributes of such objects.
- *Function*: This refers to a qualitative description of the purpose of a structural system or an element. The primary purpose of the structure is to transfer the incident loads, as well as its self-weight, from their respective points of origin to the ground.
- *Behavior*: Behavior refers to the response of a structure to applied loads. Thus behavior is essentially the manifestation of the structural system performing its function—as a result of carrying loads the structure deflects and develops internal stresses.

Engineering design involves determining the *form* of a physical system, given its function and desired behavior, such that the system satisfies the constraints imposed on it. Analysis on the other hand involves determining a system's *behavior* given its form and function [57].

As has been mentioned earlier, structures are usually unique systems. As a result, testing of physical models to determine the suitability of the design solution is generally not feasible (although it can be applied, for instance, in the case of wind-tunnel model testing of high-rise buildings). However, because of human safety and serviceability considerations, it is necessary to develop reliable means to predict the behavior of the structure. Impracticality of the physical models, coupled with the need for behavior prediction, necessitates mathematical modeling (numeric as well as symbolic) of the structural response in terms of known quantities. In the case of design problems, to determine the form efficiently one also needs to model the problem-solving approach of experts in the domain.

Different types of models—ranging from diagrammatic to physical—can be used to represent an engineering system. The representation of the form, function, and behavior of the engineering system can be used in conjunction with the representation of meta-level problem-solving knowledge, forming a layered reasoning system modeling two distinct conceptual entities: (i) the physical system, and (ii) the human being reasoning about the physical system [57]. To define the architecture of a computer implementation of these models, one should clearly identify the various components involved and establish the roles of the components in the overall scheme. This chapter is intended to fulfil that goal. A methodology for developing knowledge systems to assist in conceptual design of structures is proposed in this chapter. The methodology builds on Ref. [57] and draws significantly from the reference. The next section presents an overview of the methodology while the subsequent sections consider the individual components of the methodology in detail. The final section summarizes the contents of the chapter.

3.1 Overview of the Methodology

In the methodology for performing conceptual structural design in an automated environment presented here, we first identify different items of knowledge that are relevant to the design process, organize this knowledge into distinct categories (described subsequently), reason from such knowledge to develop alternative solutions for a design problem, and finally, evaluate and critique the generated alternatives. An overview of the different facets of the methodology is presented in the remainder of this section.

1. *Functional Elements and Systems Knowledge*: This is one of the basic categories of knowledge, containing information about attributes like function, qualitative behavior

modes, and form of generic structural elements and the systems that can be synthesized from them. The attributes will usually have a value (or a set of values) that will always be true. Some examples of structural elements are *Beam*, *Wall*, and *Cable*. The synthesized systems can be *Truss*, *Grid*, *Moment Resisting Frame*, etc. To illustrate, the function of an *Arch* is to carry the loads across horizontal spans through the primary behavior mode of compression and secondary behavior modes of bending and shear. The form can vary; some possible forms are, parabolic, radial, and funicular.

2. *Behavior and Performance Knowledge*: The behavior knowledge refers to the fundamental principles of structural engineering which quantitatively describe the forces, stresses, deflections, etc., for structural elements and systems mentioned earlier. An illustrative example of knowledge in this category is the set of equations expressing the distribution of shear and bending stresses across the cross-section of flexural members. The general cable theorem, which relates the horizontal component of the cable tension with the geometry and external vertical loading on the cable, is another such example.

   Performance knowledge, on the other hand, refers to the knowledge about performance criteria imposed on the structure. Much of the knowledge contained in design standards and specifications is of this type. Design specifications are a source of statutory constraints that must be met by the final solution. Note that performance knowledge also encompasses knowledge about material properties. One sample usage of performance knowledge is the determination of the amount of individual loads and the load combinations for which a structure must be designed. Another example pertains to the control of wind-induced vibrations in long-span bridges.

3. *Product Knowledge*: Knowledge about specific products that can be used for construction is classified under this category. Such knowledge may range from geometrical properties of commercially available hot-rolled steel sections to market knowledge about the availability of certain products in a particular region and the associated cost data.

4. *Concepts*: Various abstractions, e.g., architectural and structural patterns in the floor plan of a building, *Cost/Value* ratio, etc., play a supporting function in formulating strategy and defining evaluation criteria. Such abstractions are collectively denoted by the term *Concepts*.

5. *Strategy*: While a large fraction of the generic domain knowledge is contained in the modules *Functional Elements and Systems Knowledge*, *Behavior and Performance Knowledge*, and *Product Knowledge*, such base-level knowledge alone is not sufficient for solving a design problem. To generate design solutions efficiently, one must also capture meta-level knowledge, or problem-solving knowledge, that operates on the base-level statements and specifies how to utilize them. This aspect of the methodology models the approach of a set of professionals to the application task, as mentioned earlier. Thus, *Strategy* is meant to emulate the human thought process and to perform decision making based on experts' technique(s) of approaching the problem.

6. *Reasoning with Constraints*: Conceptual structural design can be performed by formulating, propagating, and satisfying constraints based on the knowledge contained in the modules described earlier. By (i) formulating constraints based on the project context as well as project-independent information, (ii) propagating the effects of a constraint originating in structural engineering or exogenous domains to the same or other domains, and (iii) selecting the values of attributes so that the constraints are satisfied, one can synthesize alternative structural schemes that can serve as candidates for evaluation and feedback.

7. *Evaluation and Feedback*: For a given design problem, there is usually more than one feasible candidate solution that meets all the constraints, thus necessitating some evaluation mechanism. Evaluation may be based on an explicit or implicit consideration of the $Cost/Value$ ratio [12,57]. A well-defined evaluation criterion, or a set of criteria, is a requisite constituent of a design methodology. Evaluation can also lead to feedback on the advantageous and disadvantageous aspects of different alternatives, and suggestions on improving an alternative.

Each of these components is described in detail in the following sections and their usage is illustrated with suitable examples.

3.2 Functional Elements and Systems Knowledge

As mentioned in Section 3.1, this component contains knowledge about various types of elements and systems that can be used for structural design. In terms of the type of knowledge represented, the emphasis is on first principles of structural engineering and not heuristics. Thus, while knowledge of the form "a post-and-beam frame is unstable" will be included here, a statement similar to "a framed tube system is good only for buildings over 30 stories" will not be.

In the remainder of this section, we give some illustrative examples and show how reasoning based on *function* can be used to deduce the requirement for some structural components. We should reiterate that any declarative formalism can be employed for representing this knowledge. For instance, the following knowledge can be represented in terms of objects such as `Hanger`, `Floor Plate`, etc. having attributes *Function*, *Form*, and so on. Alternatively, it can be equivalently stated by means of first-order predicate calculus statements where, for example, *Form* is a relation that holds true between the objects `Floor Plate` and *2D-Horizontal.*

Hanger

Function	To transfer the applied loading in a vertical direction.
Primary Behavior	Axial Tension
Other Behaviors	None
Form	1D Vertical
Possible Materials	Steel, Wood
Supported By	Hanger, Transfer Girder, Wall Bracket

Floor Plate

Function	To collect vertical loads distributed in a horizontal plane and to provide a surface forming element.
Primary Behavior	Flexure
Other Behaviors	Shear
Form	2D Horizontal
Possible Materials	Reinforced Concrete, Steel Deck, Composite Deck, Plywood
Supported By	Beam, Column, Wall

Framed Tube

Function	To resist lateral loads.

Primary Behavior	Overturning moment resistance through axial forces in columns in the direction of the loads as well as the ones perpendicular; story shear resistance through bending in columns.
Other Behaviors	Shear Lag
Form	3D Vertical
Possible Materials	Steel, Reinforced Concrete
Supported By	Foundation

It should be noted that the preceding classification is based on functional objects. In the representation based on function we form instantiations from these objects to represent the actual physical entities. Thus, from a functional object like `Column`, we can form instantiations to represent `Column 1`, `Column 2`, and so on. Note, however, that there doesn't have to be a one-to-one mapping from physical objects to functional objects; the same physical object may perform more than one function. To illustrate, consider the case of structural design of a building. The *Functional Elements and Systems Knowledge* module will contain descriptions of `Floor Plate` (a member of the gravity load resisting system) and `Diaphragm` (a member of the lateral load resisting system), besides others. If the same physical entity performs the functions of both the `Floor Plate` as well as the `Diaphragm`, instantiation from both of them will result in the same object when the design solution is being synthesized. As we argue later, reasoning based on function results in greater flexibility and is more conducive to innovation.

To show how the previous knowledge can be used, let us again consider the case of structural design of a building. From the fact that the structure is a building, we can infer that there will exist vertical loads (in addition to other types of loads) which will be distributed in a horizontal plane. This reasoning suggests that a structural element that can perform the function of collecting distributed vertical loads in a plane will be needed. Looking at our knowledge base, we see that a floor plate can perform such a function, and thus is a candidate element to be used in the structural system. If no other element can perform the said function, a floor plate has to be used, thus establishing a definite requirement of floor plate in case of a building. However, if the structure was a transmission tower, there is no function of collecting distributed vertical loads, thus making a floor plate unnecessary.

3.3 Behavior and Performance Knowledge

Behavior knowledge embodies the relationships among numerical quantities like loads, stresses, and deflections. Behavior knowledge is primarily first principle knowledge. For instance, the equation

$$f = \frac{Mc}{I} \tag{3.1}$$

expresses the relationship between the bending moment, M, bending stress, f, moment of inertia, I, and the distance from the neutral axis, c, for a structural element under flexure within the elastic limit. Some other examples of behavior knowledge in the case of a simply-supported beam under uniformly distributed load, w, are given below (where the symbols denote their usual meanings).

$$M = \frac{wl^2}{8} \tag{3.2}$$

$$\delta = \frac{5wl^4}{384EI} \tag{3.3}$$

A related type of knowledge, namely performance knowledge, specifies the legally required constraints on the behavior of the structure or its individual components. For instance, in the case of beams, there is commonly a restriction of the following form on the permissible deflection:

$$\delta \leq \frac{l}{\alpha} \tag{3.4}$$

where l is the span of the beam and α is some numeric constant, such as 240 or 360.

With the help of Eqs. (3.1)–(3.4), we can illustrate how behavior and performance knowledge can be combined during the process of conceptual design. Consider, for example, a steel beam with an I-section. We need to consider only the deflection due to live loads when satisfying Eq. (3.4). Let r denote the fraction of the total load that is due to live load. Replacing w by rw in Eq. (3.3) and combining it with Eq. (3.4), we get

$$\frac{5rwl^4}{384EI} \leq \frac{l}{\alpha} \tag{3.5}$$

Based on Eqs. (3.1), (3.2), and (3.5), and the relation $c = d/2$ (where d is the depth of the beam), we can deduce the general relationship for the minimum depth of a beam in terms of average allowable stress, f, the span, l, and the factor α.

$$d \geq \frac{5rf\alpha l}{24E} \tag{3.6}$$

When designing a specific beam, the values of r, l, E, and α will be known and an estimate can be made for f. Thus the depth can be selected in such a fashion that deflection requirements are not violated. To illustrate, consider the case of an A36 beam for which $E = 29000$ ksi, $\alpha = 360$ (to achieve a deflection limitation of $l/360$), $r = 0.6$ (corresponding to a 60% contribution of live load to total load) and allowable stress, $f = 24$ ksi (for A36 steel). Substituting these values, and adjusting the equation to get d in inches while l is in feet, we get:

$$d \geq 0.45\, l. \tag{3.7}$$

If the beam depth is selected in accordance with this criterion, then the code-specified live load deflection limitation of $l/360$ is always satisfied for the illustrated case. This bit of knowledge is sometimes coded as a heuristic in structural design systems, but, as the previous example illustrates, it is unnecessary to do so in view of the ability to derive the relationship between d and l based on behavior and performance knowledge. Moreover, the relationship is more general and can be applicable in a wider variety of contexts (e.g., for different values of α or f) than the corresponding heuristic, which will hold true only for some combinations of variables.

Performance knowledge is often based on past observations about the behavior, and thus the distinction between behavior and performance knowledge is sometimes blurred. For instance, the stress-strain relationship for common construction materials like steel and concrete can be determined experimentally, thus providing the behavioral basis; however, an idealized relationship contained in the specifications can be used when actually designing the elements, thus using constraints derived from performance knowledge. Because of the link between them, and because design codes contain both types of knowledge, we have chosen to put behavior and performance knowledge together in a single component.

3.4 Product Knowledge

The knowledge about the attributes of the products that can be employed for constructing a facility can be used to formulate constraints regarding the set of possible solutions and to evaluate those solutions. This component of the methodology contains such knowledge including, for instance, the AISC table of steel shapes, manufacturers' catalogs of standard building components, pricing information relative to material and labor, etc. The specific knowledge contained in this component will depend upon the type of application being developed, and even for a given application the knowledge may be dynamic because of other considerations. As an example of the former, one need not represent properties of steel sections in an application meant to design concrete bridges. As an example of dynamism, some attributes of the products—cost being a prime example—may vary from region to region.

Also, some of the knowledge may be vendor-dependent. In the case of cold-formed steel decks, for one, the properties may vary from vendor to vendor. In other cases the knowledge will be independent of the vendor. For instance, the cross-sectional area of a #3 rebar will be the same irrespective of the vendor. Structurally, there is nothing inherently fundamental about most of product knowledge. For instance, at least in principle, one can use a rebar having a diameter of 3.5/8". However, since the final design must be constructible, current construction practices have to be reflected in the design process, thus necessitating this component.

3.5 Concepts

While developing a strategy or defining evaluation criteria, one may need to use some auxiliary concepts useful for encoding the problem-solving knowledge. To illustrate, architects and structural engineers often work in terms of geometric patterns during the conceptual design stage of buildings. The layout of a structural system may be strongly influenced by the presence of such patterns. Thus *pattern* becomes a concept that has to be recognized and accounted for while solving the problem. Another related example is the idea of *column grids.* Computer systems for generating floor framing schemes have employed this idea in the past to arrange columns in a regular fashion in the plan of a building. Such concepts should be explicitly identified in the knowledge base of the system.

As noted earlier, $Cost/Value$ ratio can be used as an evaluation criterion to compare alternative solutions. $Cost$, $Value$, and the $Cost/Value$ ratio are all supporting concepts useful for formalizing the evaluation process.

3.6 Strategy

Strategy refers to the approach of solving a problem—the knowledge about how to use other knowledge. In essence, strategy is a structured form of anticipatory knowledge about the relationships among form, function, and behavior that allows manipulation of the problem constraints to effect a desired outcome. The upshot of this component of the methodology is that one captures the experiential knowledge of the designers in a given problem domain. Coming up with the *form* of a design solution, which is a highly creative process that relies more on the ingenuity and experience of the designer than on the foundational knowledge in the domain, is accomplished largely through strategy.

A simple example will illustrate the usage of strategic knowledge. Recall that in Section 3.3 we deduced an expression for the minimum depth of a beam, d, from some behavior and performance considerations. Although mathematically it is equally valid to derive expressions for w or E instead, we implicitly recognized that expressions for w and E are not meaningful because, typically, d is the quantity that can be varied to satisfy the behavior and performance requirements. Hence, in order to block superfluous inferences from the represented knowledge, one also has to state explicitly how the represented knowledge can be used best.

Such control knowledge can be very important for the sake of efficiency; it may be used to pare down the search space of the feasible design solutions in the very early stages, based on some high-level considerations. Strategic knowledge, however, may be hard to acquire because it may be too implicit or obvious to the expert. The earlier example of w and E being relatively fixed quantities is a case in point.

Among the examples of knowledge in this category are knowledge about the decomposition of the problem, knowledge about when to formulate what constraints, and knowledge about how to utilize some concepts. As hinted earlier, an important aspect of strategy is anticipation; by anticipating the downstream decisions and the effects of present choices on them, one can minimize the revisions to the evolving design.

Heuristics are likely to be predominant in the knowledge contained in this component. Furthermore, problem-solving strategy may vary from designer to designer; hence any particular encoded strategy represents only a subset of candidate strategies. Since information in the *Functional Elements and Systems Knowledge* module is represented in a pure declarative—or task-independent—fashion, it should be possible to build different strategies that can operate on the same set of base-level statements. Thus, one can deduce design descriptions that use different approaches to arrive at the final solution, though all of them satisfy the applicable constraints. An instructive situation where this may be desirable occurs in the design of columns for multistory steel buildings. One possible strategy for the selection of steel sections for usage at different floors is to choose those sections that result in the least amount of steel used (minimum weight strategy). Another possible strategy, arising from splicing considerations, is to choose from only those sections that have the same internal depth. (W14 sections, for instance, fulfill this criterion.) The computer system may present the options to the user and let him or her make the decision regarding which strategy to use. Alternatively, the system may explore both the options and evaluate the resulting designs.

3.7 Reasoning with Constraints

Once the various types of knowledge described in the preceding sections are represented in a suitable format, one can reason from such knowledge to derive design solutions for a problem. The framework Luth [57] proposes for going about such a task is to formulate, propagate, and satisfy constraints. Constraints can be formulated based on information about the project context (e.g., location) as well as project-independent knowledge (e.g., general structural engineering principles). Thus, as demonstrated in Section 3.2, given the fact that the facility to be designed is a building (project-specific information), we can formulate the constraint that one must collect distributed vertical loads. Provision of a structural element `Floor Plate` will satisfy the constraint; however, through the process of propagation, one can formulate some additional constraints. For example, one now needs some structural element(s) that can collect the load from the floor plate and transfer it to the ground.

The constraints that can be formulated may arise from structural considerations or exogenous (e.g., architectural, mechanical, constructibility, etc.) considerations. Constraints arising in different domains may interact with each other, thus forming mutual constraints. As an example, consider the case of a floor system of a high-rise office building. In a typical floor system, mechanical and architectural elements (ductwork and ceiling, respectively) are also present in addition to the structural elements like girders and floor slab. Thus structural depth, mechanical depth, and ceiling height form a mutually constrained grouping such that, when taken in conjunction with the desired floor-to-ceiling height, they should not violate the restriction on the acceptable floor-to-floor height. The consequence of such a relation is that variation in the parameters of some domain may influence the decisions in other domain(s). Thus, if the depth of the mechanical ducts is increased, one may need to reduce the depth of the girders, or, alternatively, if the ducts were initially underneath the girders, they may now have to be passed through the girders.

The two top-level categories of constraints, namely *structural constraints* and *exogenous constraints*, can be further decomposed in accordance with the classification proposed in Ref. [57]. The subcategories are diagrammatically illustrated in Fig. 3.1 and are described in the following two subsections.

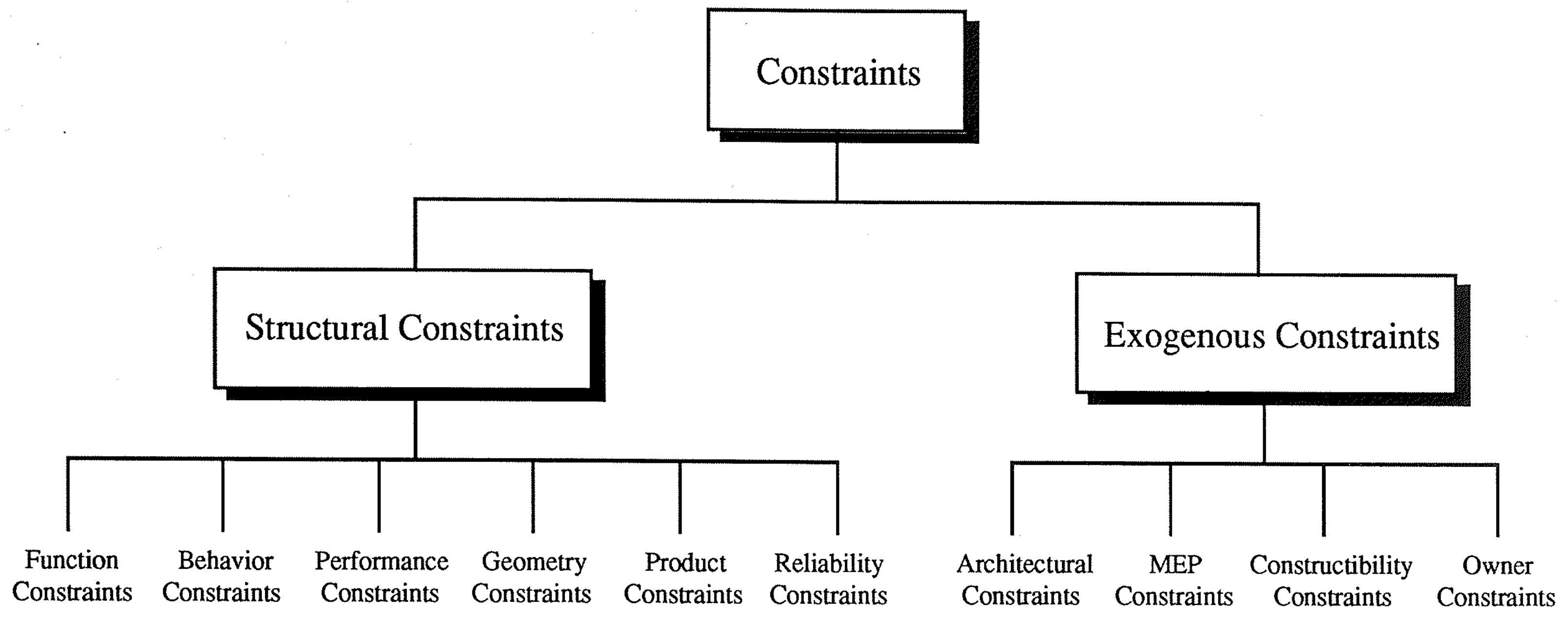


Figure 3.1 **Constraint Classification.** Constraints on the structural design may arise from both structural and exogenous considerations.

3.7.1 Structural Constraints[1]

Structural constraints include function, behavior, performance, geometry, product, and reliability constraints. Constraints arising from other subsystems (exogenous constraints) must be transformed into one of these types of constraints before their impact on the structure can be considered.

Corresponding to the primary function of the structure as mentioned earlier, *function constraints* refer to the loads and their locations relative to the ground. The loads can be described in terms of forces that have a magnitude, a direction, and a location in space. Some elements of the structure may also perform an architectural function; for example, a slab, besides performing the structural function of carrying loads, may also be a functional object `Floor` from the architect's point of view. In such instances there may be constraints on the physical object arising from its function in another domain.

Behavior constraints are derived from the behavioral part of knowledge in the *Behavior and Performance Knowledge* component. They are useful in determining the response of the structure while it is performing its function of carrying load. The behavior is dictated by such fundamental principles as Hooke's law and principle of superposition. Behavior constraints are absolutely "hard" constraints—they cannot be relaxed under any circumstances.

Performance constraints are based on the knowledge related to specifications and other performance criteria in the *Behavior and Performance Knowledge* component. They place limits on the values of the behavior the structure exhibits when subjected to loads, and thus enhance safety and serviceability of the facility. If performance constraints are violated, one may have to vary one or more of (i) the structure topology, (ii) member material, and (iii) geometric properties, to alter behavior in such a way that the applicable performance constraints are satisfied. Performance constraints can be further divided into *serviceability* and *safety* constraints. Serviceability constraints limit, among others, the deflection, vibrations, and cracking of a member or structure. Safety constraints, on the other hand, limit the internal stresses (in the case of working stress design) in the member, or specify a relation between the member force demand and the member capacity for that type of force (in the case of load and resistance factor design for steel, or strength design for concrete).

Geometry constraints define the location of the structural elements and spatial relationships amongst them. Product knowledge, in many cases, can also be transformed into geometry constraints. Constraints arising in other domains are often a source of geometry

[1]This and the next subsection are drawn from Ref. [57].

constraints on the structural system. Concrete beams that must be spaced at a specified interval to accommodate a particular arrangement of forms, spacing limits on steel beams that are a function of the cost of fabricating connections, and limits on member sizes based on crane capability or shipping requirements are all examples of geometry constraints that result from consideration of constructibility.

Product constraints exemplify the spectrum of choices available concerning specific materials and members. They are typically a result of transformation of the knowledge contained in the *Product Knowledge* component. *Functional Elements and Systems Knowledge* may also be a source of product constraints; for example, reinforced concrete and steel may be the only usable materials for a framed tube. In addition, the user may also impose certain constraints; for example, although both steel and concrete tubes may be possible, the user may want only the option of steel tubes to be explored. Many constraints in this category may originate within the construction domain also. An example of a constructibility constraint that transforms into a product constraint in the structural design domain would be the concrete strengths that can be produced in the area where the facility is located.

Reliability constraints allow exercising of engineering knowledge and/or judgment to account for the probability that the behavior of an alternative will be satisfactory. Redundancy, which is a property of the structure related to its function, is an example of a qualitative measure of the reliability of a structure. If there is only a single path for the loads to follow, the structure is "nonredundant;" if there are multiple load paths so that when an element in one path fails, the load can still be successfully transferred through an alternate path, the structure is "redundant." Redundant structures are considered more reliable. Though reliability aspects are implicitly considered in several instances (e.g., in the method of determining the seismic design loads, in the load and resistance factors used for limit state design of steel and concrete structures, etc.), methods of explicitly incorporating issues of reliability during the design of structures have, for the most part, not been formalized.

3.7.2 Exogenous Constraints

Exogenous constraints are those constraints that are relevant to the design of the structure, but which originate in a domain outside structural engineering. The source of these constraints may be architectural, MEP, constructibility, or owner considerations. Since buildings have many other important considerations besides the structural system, exogenous constraints have an especially pronounced impact in the case of buildings. Not all types of exogenous constraints described here may be present for all types of structures; for example, MEP constraints may not be applicable to bridge structures; however, they will be applicable in the case of power plants.

Architectural constraints arise because of the interconnection between the architectural design and structural design of a facility. The architectural form often defines the geometric context for the structural system within the facility. The aesthetic expression may affect the geometric arrangement of the members within the structure for visual effect. Individual features of the architecture also result in significant constraints on the structure. For example, placement of columns may be ruled out in the central arena of an indoor stadium.

MEP constraints are very relevant in the case of high-rise buildings and many other types of structures. Each of the mechanical, electrical, and plumbing subsystems involve tasks such as origination, distribution (or collection), and delivery. The subsystems are a collection of components that have physical attributes like size and weight. Because of their size, the MEP components compete with the structural and architectural system components for a share of the finite space defined by the building envelope. The weight attributes of the subsystem components become function constraints on the structure. Another type of interaction occurs as a result of the behavioral characteristics of the subsystems within the context of the building usage. As an illustration, noise and vibration resulting from the operation of equipment may have to be isolated from adjacent spaces.

Constructibility constraints result from the consideration of construction activities. Equipment capabilities, material availability, formwork considerations, etc., are all source of constructibility constraints. Moreover, because of the differences in the cost of labor, the available technology, and the available materials, as well as differences in the preferences of the designer and owner communities, certain structural systems are favored in some regions of the country. This preference is usually apparent in the prices associated with the systems and needs to be taken into account at the conceptual design stage. In several instances constructibility considerations are also implicit in the problem-solving strategy. For example, in the case of concrete a new form is required for every difference in the shape and size of a structural component. In view of this, one may strive for uniformity in the shape and size of structural members while generating solutions, sacrificing some degree of optimality in weight.

Owner constraints usually involve factors that affect the perceived value of the facility, the cost of managing the facility, or the schedule for the construction of the facility. Among these are constraints on vibration limits for floors, designation of certain areas as high load intensity areas, and the cost of modifying the structure to meet changing requirements. The owner may also have specific schedule requirements based on the need for the facility. In such a case an additional constraint on the conceptual design is that it should be constructible within the permissible amount of time.

3.8 Evaluation and Feedback

Since it is unusual for a design problem to have a unique solution, it is necessary to define some mechanism to determine the relative ranks of the generated alternatives and to critique them. In the past the typical approach for assessing different solutions has been to define an evaluation function based on certain parameters (e.g., flexibility for future modifications, speed of construction, uniformity in the sizes of structural components, unit weight of the structure, etc.) that will be used to rank the solutions. Weights are associated with each of these parameters to reflect their relative importance. Actual values of these parameters are then computed for an alternative based on a system of reward and penalty as compared with some normalized values. The weighted mean of the actual values of the parameters for an alternative is then assigned to the evaluation function, whose value is taken as a measure of the intrinsic merit of an alternative.

We believe that the net result of such an exercise is only to provide an indirect measure of the $Cost/Value$ ratio. The quantitative value of the evaluation function is not very useful for a human designer and the method of indirect measurement cannot be precise. Firstly, some degree of arbitrariness is introduced in deciding the weights and "normal" values of parameters. The problem is further complicated when qualitative responses of the user have to be accommodated. To illustrate, in response to a question about the availability of a certain material in a particular region, the user may have options of *Excellent*, *Good*, *Fair*, and *Poor*. The conversion of such values for usage in computing the evaluation function may not be universally acceptable.

One solution to such problems is to measure the effect of all relevant parameters in terms of either $Cost$ or $Value$, and compute the $Cost/Value$ ratio to get an indication of the merit of an alternative [58]. To be precise in the $Cost/Value$ analysis, one should use the costs and value based on the life-cycle of the facility. However, long term estimates of cost and value involve variables that are beyond the control of the participants in the design/construction process. Moreover, the $Value$ part is often highly subjective and not amenable to measurements by a computer. For instance, the aesthetic value of alternatives may defy precise measurement. As another example, the *worth* of 'flexibility for future modification' in an alternative may vary from person to person. The compromise that we have adopted in the face of such difficulty is to associate an estimate of short-term cost with each alternative and present such data for all alternatives to the owner. The owner can then select one based on the respective perceived values of the alternatives. The default choice can be the one with the least cost.

Another aspect of evaluation is to critique the alternatives and provide feedback, if any, on how the alternatives can be improved. For instance, while determining the cost of erecting beams in a floor, the system may notice that all but two of the beams are of the same size and the other two are only slightly smaller. Upon further computation, the system may find that the extra cost of ordering/erecting the two smaller beams more than offsets the savings in the material cost. In such a case the evaluation process may result in a feedback to use the same beams throughout the floor. One may also try to anticipate downstream constraints while evaluating the alternatives and providing feedback. For instance, if the width of a concrete floor beam framing into a concrete column is less than the width of the column, there may be complications in erecting the formwork at the joint. The system can provide feedback about possible problems of this nature when evaluating an alternative (see, for instance, Ref. [22].

3.9 Summary and Concluding Remarks

To summarize, we regard the explicit definition and representation of function, form, and behavior of different structural systems and elements as essential for developing flexible conceptual design systems. Using these aspects one can model the structure, and when coupled with a model of the design process (i.e., strategy), one can efficiently generate solutions for diverse design problems. The reasoning can be carried out through the process of constraint formulation, propagation, and satisfaction. Knowledge of various types, e.g., quantitative behavioral description of structural elements and systems, design specifications, available products, etc., can serve as the source of constraints. Once the alternatives are generated, the $Cost/Value$ ratio can serve as the measuring yardstick for evaluating solutions.

The emphasis on the explicit representation of the qualitative functions of elements and systems is very important, as can be illustrated through a simple example. When designing a high-rise building, one can establish that there should be a structural system to resist the lateral loads. Resisting lateral loading can be further refined into resisting overturning moment and resisting shear. By choosing (i) axial forces in columns in the line of applied loading as the mode of resisting overturning moment, and (ii) flexure in columns as the mode of resisting story shear, and reasoning that beams are required for moment equilibrium of the columns at the joints, we can synthesize a moment resisting frame as the structural system. Reasoning along the same line, we can include columns in the direction perpendicular to the line of force for resisting overturning moment, thus "inventing" the framed tube. Extending the example further, by changing the mode of resisting story shear from flexure in columns to axial forces in inclined elements, we can "invent" the braced tube. Thus, through a process of deciding the basic behavior modes for satisfying functional needs, it is possible to compose structural systems to perform the desired function.

The discussion of the methodology in this chapter has deliberately been general and applicable to several other types of structures besides buildings. Chapters 4 through 7 illustrate the application of the methodology to a specific problem, namely, conceptual structural design of multistory steel office buildings. We begin in the next chapter by discussing design of the gravity system of such structures.

The sciences do not try to explain, they hardly even try to interpret, they mainly make models. By a model is meant a mathematical construct which, with the addition of certain verbal interpretations, describes observed phenomena. The justification for such a mathematical construct is solely and precisely that it is expected to work.

— *John von Neumann*

History suggests that the road to a firm research consensus is extraordinarily arduous.

— *Thomas S. Kuhn,* The Structure of Scientific Revolutions *(1962)*

4 Gravity Load Resisting System

This chapter describes the knowledge and reasoning behind FFG (for Floor Framing Generator), the part of Galileo that performs design of the gravity load resisting systems for multistory steel office buildings. Salient aspects of the methodology presented in Chapter 3 are illustrated in the context of design of gravity systems in this chapter. While the description in this chapter is at a fairly high level, a detailed account of the working of FFG can be found in Appendix C.

The remainder of this chapter is organized as follows: We first present a brief introduction to the problem of floor framing generation in the next section. Section 4.2 contains a review of relevant previous works. An elaboration of the various types of knowledge, constraints, concepts, strategy, and evaluation follows in the succeeding sections. A brief summary concludes the chapter.

4.1 Introduction

The primary function of structural system in a building is to transfer the loads from their points of origin to the ground. Depending on their direction, the loads are classified as either *lateral loads* or *gravity loads.* Schemes to transfer both types of loads need to be devised during the conceptual phase of structural design of a building. It is in the context of the transfer of gravity loads that the problem of floor framing generation arises.

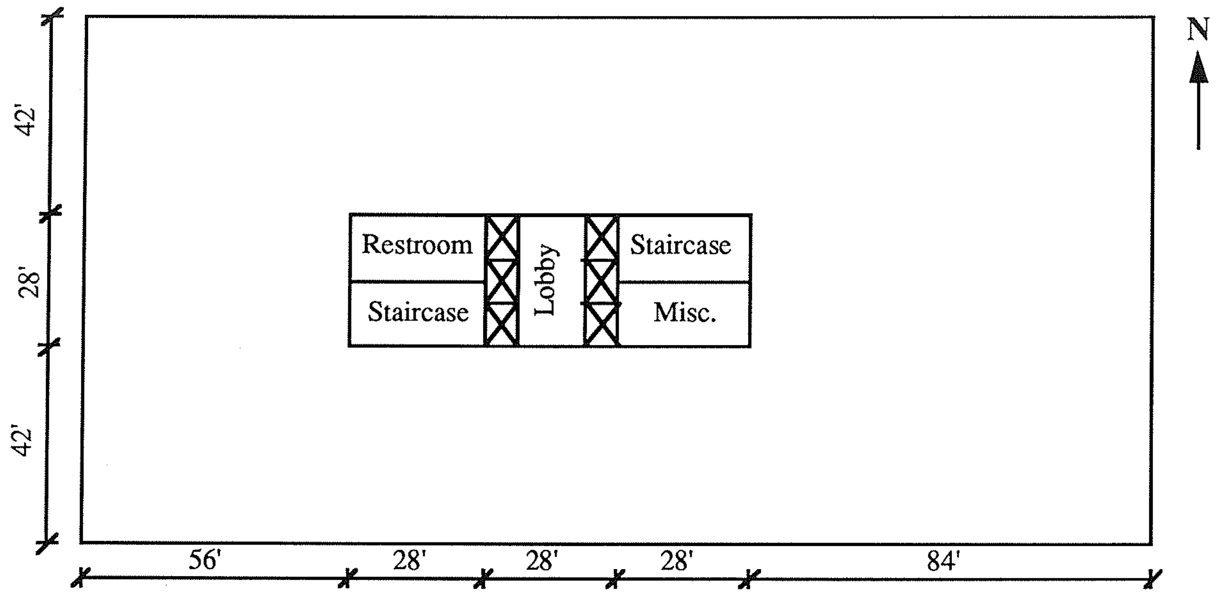
Floor framing generation involves providing a path to transfer the gravity loads to the ground through various structural elements in an architectural plan, while meeting the requirements imposed by other entities (such as the architect, the mechanical engineer, and the contractor) involved in the design/construct process. One of the tasks in the process is to determine the locations of columns, beams, and girders. Determining the locations of beams and girders, in turn, involves deciding on their respective orientations and spacings. The process can be illustrated with the aid of Fig. 4.1. Figure 4.1(a) shows part of a sample input (i.e., the architectural floor plan) to the process whereas Fig. 4.1(b) represents a corresponding partial output (i.e., the framing plan). (The filled squares in Fig. 4.1(b) represent columns.)

FFG generates floor framing schemes for steel office buildings that are rectangular in plan and have a single service core. FFG requires that all architectural spaces (such as restrooms, hallways, staircases, etc.) within the plan be rectangular. Although the approach presented here has been developed in the specific context of high-rise steel office buildings that have rectangular plan shapes, it is extensible through appropriate modification of some concepts and constraints, and introduction of others, to make it applicable to other types of buildings as well. In particular, with minor modifications, one should be able to handle buildings whose plan is not rectangular but can be divided into rectangular components. Handling arbitrary geometries, however, would require major enhancements.

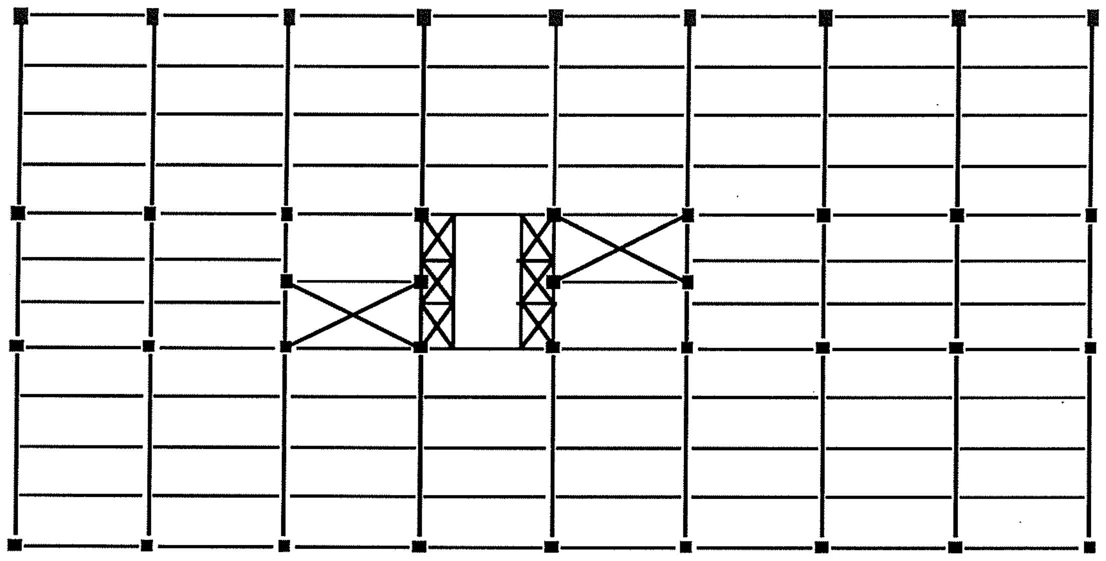
4.2 Background

There have not been many knowledge-based systems in the past that have concentrated primarily on the issue of floor framing generation. Some of the well-known knowledge-based systems for preliminary structural design, such as HI-RISE [60] and ALL-RISE [80], do not perform floor framing generation; instead, the framing plans are supposed to have already been generated and provided as input to the system.

One system that did address the problem in detail was FLODER [46]. FLODER generates, analyzes, and evaluates floor framing plans for floor plans that can be subdivided into rectangular areas. For the generation part, FLODER works in terms of column lines whose placement is largely guided by the minimum and maximum economically feasible spans of the framing material. If the constraint on the maximum economic span of the material is violated, the situation is rectified by inserting additional column lines and thus subdividing the original span. The reverse takes place in case the constraint on the lower limit is violated. Girders are placed along the so generated column lines in both orthogonal directions. If the spacing between consecutive girders exceeds the maximum economic span of the slab material, beams are generated parallel to the longer side of the rectangle. In case the floor plan is square, beams are generated in the X-direction. Again, the number of beams



(a)



(b)

Figure 4.1: **Generation of Floor Framing Plans.** (a) Architectural plan. (b) Floor framing plan.

generated is such that the criteria of span limits are met.

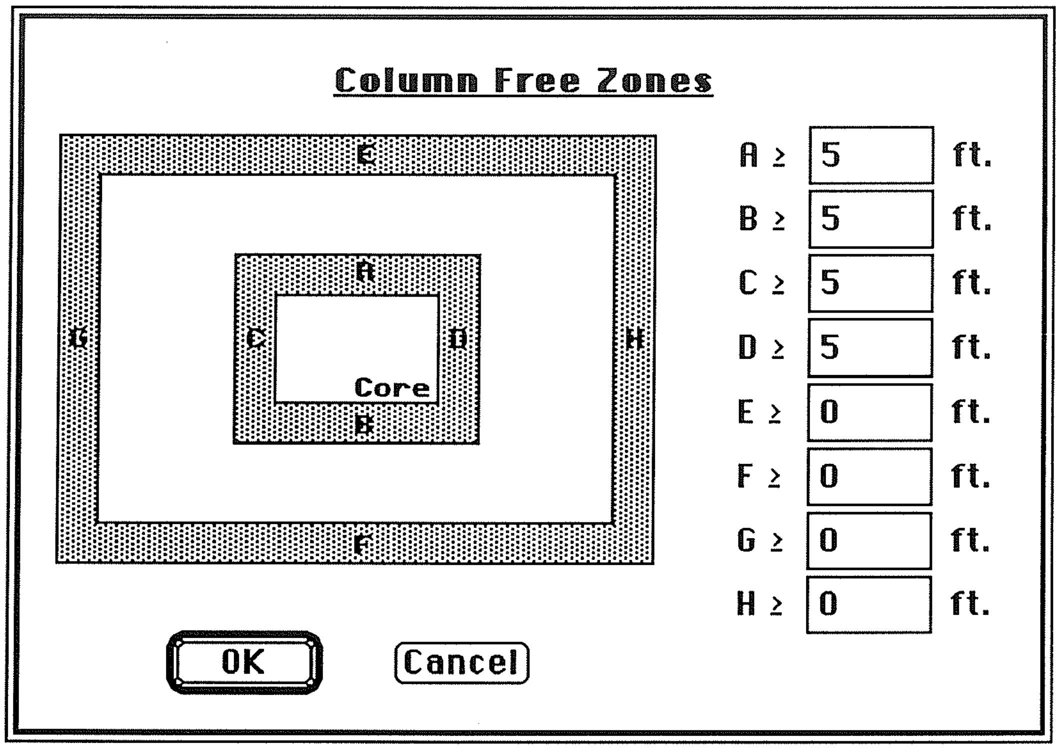
FLODER represents a good start and emphasizes one important aspect of a good floor framing alternative: the necessity of meeting the economic span criteria. However, there are additional considerations that a human designer employs when generating solutions. Thus, FLODER's methodology needs to be enhanced to make it more useful for practical purposes. One disadvantage of FLODER is that in order to come up with an efficient framing system, it takes the liberty to rearrange the location of the mechanical shaft and hallways in the building plan. This is impractical as the location of the shaft and hallways is governed by many other (more important) considerations such as architectural constraints, maximum rentable space, building services, etc. The framing plan usually has to be worked around fixed locations of these spaces, and only in rare cases (e.g., where there is a significant saving in cost) are the locations altered.

A more recent work [7] attempts floor framing generation from the perspective of context-sensitive grammars. The *Structural Generators* in this work perform the spatial layout, using 30' as the preferred bay size, 23' and 35' as the minimum and maximum column spacings, respectively, and a preference for symmetrical layouts. The knowledge contained in the generators, however, needs to be made more comprehensive and deep. "Hard-wiring" the values for span ranges also adversely affects the flexibility of the system.

Ideally, a general floor framing generator should be able to take constraints specified by the exogenous entities as input and incorporate such constraints while generating potential solutions. For instance, as shown in Fig. 4.2, the architect may desire that there be no columns within 5' of the boundaries of the service core. Present systems do not provide flexibility to handle such exogenous constraints. As described later, FFG can handle some such considerations that arise outside the structural engineering domain. Also, the problem of hard-wiring is addressed in FFG by structuring the knowledge in terms of parameters that have certain default values which can be overridden by the user. With this background, we proceed to discuss the knowledge and reasoning used by FFG, which follows the concepts outlined in Chapter 3.

4.3 Functional Elements and Systems Knowledge

In the case of buildings, satisfying the primary function of the structural system translates to generating schemes consisting of structural elements like floor plates, beams, and columns to collect the incident and dead loads, and ultimately transferring them to the



A ≥	5	ft.
B ≥	5	ft.
C ≥	5	ft.
D ≥	5	ft.
E ≥	0	ft.
F ≥	0	ft.
G ≥	0	ft.
H ≥	0	ft.

Figure 4.2: **Specifying Exogenous Constraints.** The architect may desire certain areas of the plan to be column-free.

ground. FFG contains information about the attributes like *Function*, *Form*, and *Behavior* of generic structural elements and systems, based on which gravity load resisting schemes can be synthesized. The elements and systems for a scheme are so chosen that the desired function is performed effectively. As an example, FFG establishes the need for a structural element, `Column`, through reasoning based on function. From the fact that the structure is a building, it can be inferred that there will exist loads which act at a level above the ground. Thus, a transfer of load in the vertical direction is needed. Upon searching through the *Functional Elements and Systems Knowledge* module, FFG determines that three elements, namely, `Hanger`, `Column`, and `Wall`, can perform the desired function. The element `Hanger` is described in Chapter 3. Description of the other two elements follows.

Column

Function	To transfer the applied loading in a vertical direction.
Primary Behavior	Axial Compression

Other Behaviors	Flexure, Shear
Form	1D Vertical
Possible Materials	Steel, Reinforced Concrete, Wood, Masonry
Supported By	Column, Wall, Transfer Girder, Foundation

Wall

Function	To transfer the applied loading in a vertical direction.
Primary Behavior	Compression
Other Behaviors	Flexure, Shear
Form	2D Vertical
Possible Materials	Reinforced Concrete, Masonry, Wood
Supported By	Wall, Column, Foundation

Since FFG is presently restricted to steel as the material for vertical support system, the choice of `Wall` is eliminated from consideration. This is done by an axiom which states that there should be at least one common member in the list of *Possible Materials* for an element and the list of actual material(s) being considered. That leaves `Column` and `Hanger` as the possible candidates. Since FFG does not currently have expertise for generating floor framing plans with hangers, these are also eliminated through another axiom which states that `Hanger` is unusable. Thus, it is clear that columns are needed. It should be noted, however, that one will have the option of generating solutions containing hangers also, if the system has the expertise to handle them. Reasoning based on function is general enough to handle such situations.

4.4 Product Knowledge

This component contains knowledge about the commercially available products that can be used for construction. Properties of steel floor decks and hot-rolled steel sections, along with their respective cost components, are examples of knowledge belonging to this category that is contained in FFG. Of the available options for the type of floor systems for steel buildings, we have restricted ourselves to composite metal decks, with either lightweight or normalweight concrete on top of a formed steel deck. In such an arrangement the deck acts as the form as well as positive reinforcement for the concrete. This arrangement is typical for floor systems in steel high-rise buildings [56,84] since it offers many benefits, including a more flexible system for wiring, availability of an instantaneous working platform, and protection of workers beneath because of the metal deck [13,34]. Composite action offers advantages such as reduction in the gauge of the metal deck, structural efficiency, larger load capacity, capability for longer spans, and integral floor diaphragms [41].

As a specific example, for given combinations of (i) the depth of the steel deck, (ii) the gauge of the deck, (iii) the depth of concrete, and (iv) the type of concrete, FFG knows about the following quantities:

- spanning capability of the deck,
- the self weight of the composite deck,
- the unit volume of concrete, and
- the recommended wire fabric.

In addition to the properties of composite metal decks, knowledge of attributes of hot-rolled wide-flange steel sections, like unit weight, cross-sectional area, radius of gyration, etc., is also included. Values of some other attributes, for instance the shear and moment resistance capacities, depend upon the grade of the steel and whether the action is composite or noncomposite. Accordingly, the *Product Knowledge* module contains the values of such attributes for different combinations of steel grade and type of action. (Note that the computation of resistance capacities of sections involves utilizing behavioral knowledge as well.)

4.5 Behavior and Performance Knowledge

Behavior and performance knowledge is indispensable in designing elements like beams, girders, columns, and the floor deck. For instance, when designing the floor beams (which are taken to be simply supported) carrying uniformly distributed load, the following behavioral relation is used to determine the end reaction, R:

$$R = \frac{wl}{2},$$

where w denotes the load intensity and l denotes the beam span. The end reaction, in turn, acts as a concentrated load on the girder on which the beam rests. The bending moment, M, at the center of the girder due to such a concentrated load is given by another behavioral relation:

$$M = \frac{Ra}{2},$$

where a is the distance from the point at which the load is acting to the nearest support. (Girders are also assumed to be simply supported.) If several beams are supported by the girder, the bending moment at the center due to all such concentrated loads is computed using another piece of behavior knowledge, namely the *principle of superposition*. In a similar fashion, behavior knowledge is also applied to compute the deflection of the members resulting from the loading.

Performance knowledge specifies the limits on the observed behavior. We employ Load and Resistance Factor Design (LRFD) in FFG, and a majority of the performance criteria is derived from the applicable design standards [6,40]. Examples of performance knowledge encoded in FFG follow.

- *Specifications pertaining to loads*: For instance, magnitudes of various types of loading (such as live load, partition load, etc.), live load reduction, load factors, and the load combinations for which the structure must be designed.

- *Specifications pertaining to safety*: For instance, resistance factors for various stress modes, effective flange width of a composite beam section, and strength reduction factors for shear studs.

- *Specifications pertaining to serviceability*: For instance, permissible live load deflection and the effect of shoring and cambering on serviceability requirements.

4.6 Constraints

The reasoning for generating the framing plans is carried out in terms of constraints described in this section. As elaborated in Chapter 3, the constraints are classified as either *structural* or *exogenous* depending on whether they originate in the structural domain. Since a large component of the process of floor framing generation involves geometric decisions regarding placement of structural elements, many high-level constraints originating from structural and exogenous considerations ultimately need to be transformed into their geometric implications. Therefore, in addition to detailing the constraints in the following sections, we also mention their influence on the geometry of the framing plan. The list of constraints described in this section is not comprehensive—the ones described here are those implemented in FFG. There are other constraints that are applicable to the problem but have not been implemented.

4.6.1 Structural Constraints

Several structural constraints result from the knowledge contained in the modules explained earlier. Loads are one obvious example of constraints in this category. In addition, there are several others as described here.

Minimum and Maximum Economic Spans: Various horizontal elements that can be used to bridge parts of a building have an associated economic span range, i.e., a range of spans for which a particular horizontal system is economically (as opposed to structurally) viable. The range may vary with such factors as the location of the building and the material. The minimum and maximum economic span criteria result from the behavior and performance constraints on structural elements and help to constrain the spacing of column lines.

Minimum and Maximum Economic Spacings: Similar to the economic span ranges, there are economical ranges for spacings in the case of beams. The range for beam spacing may be governed by the spanning capabilities of the overlying steel deck and the incident loading.

Minimum and Maximum Depths of Beams and Girders: Because of the need to pass mechanical ducts through the structural system, or considerations of overall building height, there may arise constraints on the minimum and maximum depths of beams and girders in the framing plan.

Fire Resistance: Corresponding to the design specification, there is a performance constraint regarding fire rating. This constraint can influence the thickness of the concrete on top of the metal deck or can require spray fireproofing to obtain additional fire resistance.

Column Lines: A column line is a straight line defining potential locations for columns. Placement of columns is mostly constrained to be on the column lines. In essence, column lines are results of propagation of various other constraints.

4.6.2 Exogenous Constraints

Besides structural considerations, architectural, MEP (mechanical, electrical, and plumbing), constructibility, and owner considerations also influence the process of floor framing generation. Architectural considerations, in particular, have a pronounced impact since any improperly placed structural element may seriously interfere with the functionality of the building. In addition to those, this section contains MEP, constructibility, and owner constraints that have been implemented in FFG.

Architectural Constraints

Planning Module: Various elements in an architectural plan are typically aligned with a grid consisting of uniformly spaced lines in two orthogonal directions. The distance between two consecutive grid lines is not arbitrary; it is usually chosen to correspond with the standard dimensions for various building services such as lights and architectural fixtures. This distance is termed as the planning module and has typical values of 5' or 4'. The planning module may be influenced by the geographical location of the building (one particular value may be predominant in the region). For floor framing generation, the planning module defines an architectural constraint such that various structural elements should, as far as possible, coincide with the grid based on such a planning module.

Minimum Office Width: Based on the intended functional usage of the building, there is a limiting dimension that provides a lower bound on the minimum clear span in the functional areas of the building [79]. In the case of office buildings we denote such a quantity by the term *Minimum Office Width*. Minimum office width defines an architectural constraint on the floor framing that has to be satisfied by the column spacing in the framing plan.

Openings: Staircases, elevators, shafts, etc., are openings in the floor. Openings are important because no horizontal elements (such as beams and girders) can pass through them except at the edges. This constraint can result from mechanical considerations (vertically continuous shaft) or functional considerations (vertical transportation through staircases and elevators).

No Column Zones: For architectural or other functional reasons, certain areas in the floor plan may be designated as no column (or column free) zones—columns cannot be placed within the boundaries of such zones. For instance, all openings are no column zones. Similarly, lobbies are also no column zones.

MEP Constraints

Ductwork: This constraint has its origin in the MEP domain. The ductwork within the floor areas can be either restricted to pass through the structural system or below it. In the former case the overall height of the building will be smaller due to the reduced floor thickness. This will result in smaller material costs as estimated by FFG. The latter, on the other hand, will be simpler to erect. FFG's capabilities with respect to estimating costs associated with punching holes (for the passage of ducts) in flexural members are minimal at present. Through manual computation, one can assess these costs and add them as penalty to the cost computed by FFG (in the case of ductwork passing through the structural system) to arrive at the total cost, thereby providing a basis for comparison of the two alternatives.

Constructibility Constraints

Constraints Pertaining to the Method of Construction: Incorporated in FFG are constraints that permit one to specify the construction of flexural members as either shored or unshored, cambered or uncambered, and composite or noncomposite. In the case of columns one can provide the frequency of splicing in terms of number of stories. Also, FFG provides the option of choosing a design strategy such that sections of all columns have the same internal depth. This is useful for certain methods of column splicing in multistory buildings.

Constraints Pertaining to Materials: In FFG the grade of steel, the type (lightweight or normalweight) and strength of concrete, and the type of shear studs can be constrained based on the availability of materials to be used in the framing plans. The effects of these constraints will be reflected in the total cost and the unit steel weight of the structural system.

Many constructibility considerations are strongly influenced by localized construction practices and an explicit and comprehensive formulation of these constraints, which is globally applicable, is not possible. A few such constructibility considerations are incorporated in FFG through other constraints. *Column lines* are one such example since the idea of uniformly spaced columns is beneficial from the constructibility viewpoint also.

Owner Constraints

Minimum Floor to Ceiling Height: The owner may set the minimum floor to ceiling height based on value considerations. In FFG the height has implications for the loading and the length of columns.

4.7 Concepts

The concepts described herein are abstractions that perform supporting functions in formulating the strategy or defining the evaluation criterion. Additional concepts can be identified and the strategy can be modified accordingly to handle alternate geometries and different types of buildings.

Patterns: Schodek [79] states that there are often strong and easily identifiable patterns present in the functional organization of buildings and in the structural systems used. The patterns formed by the two are usually intimately related. Thus, while generating floor framing plans in an automated environment, there should be mechanisms for identifying common functional patterns and incorporating them in the structural schemes. Examples of common functional patterns found in office buildings include arrangements of office modules and arrangements of two parallel elevator banks in the core separated by a lobby.

Characteristic Dimension: A common structural pattern in buildings is composed of a series of uniformly spaced parallel lines defining locations of the vertical support system, thus forming an aggregation of repetitive bays. We denote the spacing between the parallel lines by the term Characteristic Dimension (CD). As stated earlier, functional and structural patterns are usually intimately related. Therefore, the CD's should ideally correspond to some architectural patterns in the building.

Influence Zone: The concept of influence zone is used to demarcate a certain region around the core in which columns cannot be placed. If no hallway is present around the core, influence zone is a rectangular strip with thickness equal to the minimum beam span. In case a hallway is present, the influence zone extends from the core boundaries to an imaginary boundary that is away from the hallway boundaries by a distance equal to the minimum beam span, with the hallway boundary itself being the exception. The two cases are illustrated in Fig. 4.3. The concept of influence zone arises from both functional and structural considerations. Placing columns within the influence zone will either result in interference with the movement around the core, or violation of the economic span limits, or both.

Core Partitions: Architectural spaces within the core are separated through partitions. Partitions provide better potential locations for placing columns as compared to the inside of architectural spaces, especially since the architectural spaces within the core are small and placing columns within them would seriously undermine their functionality.

4.8 Strategy

The problem of floor framing generation can be decomposed into the following tasks and subtasks:

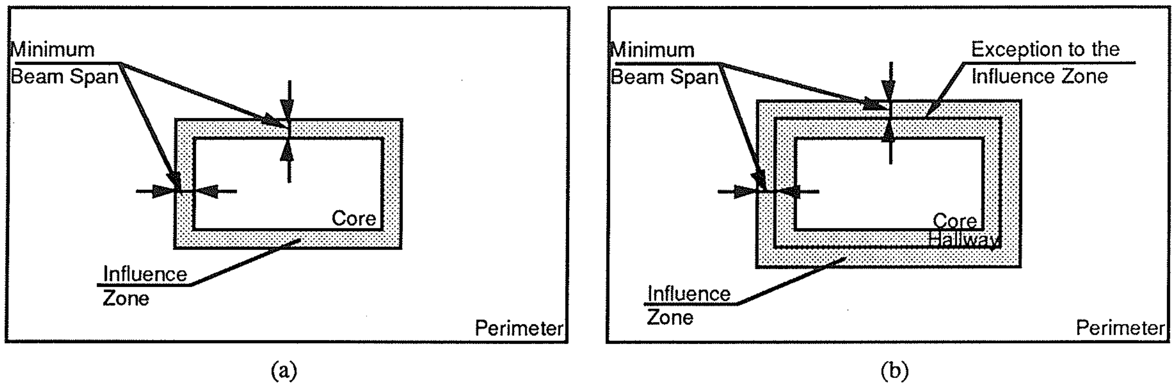


Figure 4.3: **Influence Zone for Two Different Cases.** (a) No hallway is present. (b) Hallway is present.

- Generation of column locations
  - Generation of column locations through consideration of areas outside the core
  - Generation of column locations through consideration of areas inside the core
- Configuration of the floor system
  - Decision on the type of floor system
  - Generation of beam locations
  - Generation of girder locations
- Design of members
  - Selection of steel grade and sizing of beams and girders
  - Selection of steel grade and sizing of columns

Elaboration of each of the tasks follows in subsequent sections.

4.8.1 Generation of Column Locations

The reasoning for generating column locations through consideration of areas outside the core is quite different from the one employed for generating locations through consideration of areas inside the core. The following subsection first gives a brief overview of the reasoning involved in generating column locations from outside considerations, followed by an elaboration of the individual phases. Generation of column locations from consideration of areas inside the core is described in the subsequent subsection.

Column Locations Through Consideration of Areas Outside the Core

The following three phases comprise the generation of column locations through outside considerations:

1. Controlled Generation,
2. Testing, and
3. Modification.

The process of controlled generation entails pruning down an infinite search space of framing plans to a manageable size through some of the constraints and concepts discussed in Sections 4.6 and 4.7. This is accomplished by first generating sets of column lines in the North-South (NS) and East-West (EW) directions, based on the geometry of the plan. (The plan is taken to be aligned with the four directions.) Once the sets of column lines are generated, each set of NS column lines is considered in conjunction with each set of EW column lines to yield tentative column locations at the intersections of these lines. This results in multiple alternatives. The testing phase involves checking column locations in an alternative for constraint violations. If there are no violations, the alternative is acceptable. Otherwise, an attempt is made to modify the locations of offending columns in the modification phase. The modified alternative is retained if such an attempt is successful; otherwise, the alternative is discarded. Each remaining alternative is considered for generation of column locations within the core boundary.

Two means are employed for generating column lines from outside considerations. One works in terms of patterns whereas the other depends on decomposition of the plan into different zones. We first describe the former and follow it with the description of decomposition-based column line generation.

In pattern-based column line generation, column lines in a particular direction are placed at regular distances based on some characteristic dimension. The CD's may be extracted from architectural patterns or from other features within the plan. As an example of the former, backsides of elevators are excellent locations for lateral bracing systems since there is no horizontal transportation across them. Thus, the backsides provide us with good candidates for placement of columns in anticipation of the lateral system. As mentioned earlier, a common architectural pattern involves two parallel banks of elevators separated by a lobby. The preference for uniformity, coupled with that for placing columns at the backs of the elevator banks, would thus imply that the structural grid uses a spacing equal

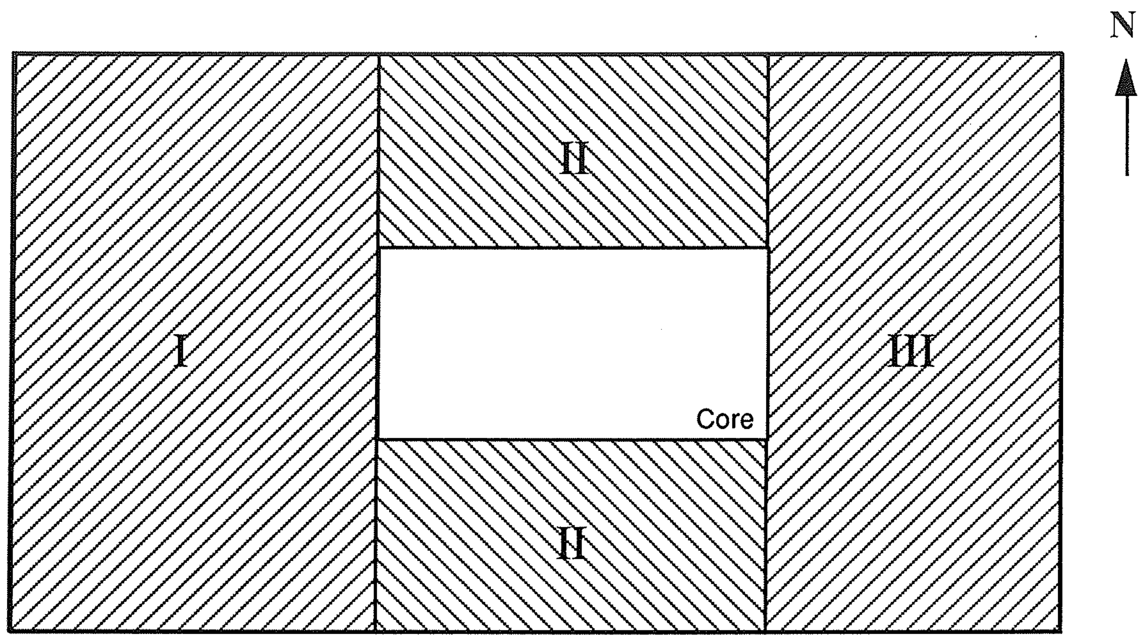


Figure 4.4: **Decomposition of Floor Plan for Generating Column Lines in the North-South Direction.** The decomposition defines three zones which are considered separately for column line generation.

to the one separating the backs of two parallel elevator banks. Thus, the distance between the backs of the elevator banks yields a candidate CD. Other potential candidates for the set of CD's include the width and the length of the core. The set of "raw" CD's extracted from these and other considerations is pruned to satisfy the constraints like minimum office space, planning module, and maximum economic girder span. Of the remaining set of CD's, a further subset is computed for each of the two directions (NS and EW) based on the criterion that the perimeter dimension should be an integral multiple of the member CD's.

Another mechanism for generating column lines is decomposition of the plan, facilitated by the assumptions of rectangular plan and core. When generating column lines in a particular direction, say, NS, we can project the core boundaries in the NS direction to divide the areas outside the core into three separate zones as illustrated in Fig. 4.4. Each such rectangular subarea can now be reasoned about individually. One possibility is to use a uniform dimension that divides each of the three areas integrally, and also meets the constraints of minimum office space and maximum girder span. If no such dimension exists, then for each of the three areas we find the respective largest dimensions that divide the areas integrally and still meet the constraints mentioned earlier. In the latter case the spacing of the structural grid will not be uniform throughout the plan.

Columns can tentatively be placed at the intersections of orthogonal column lines once sets of column lines are generated. However, some column placements may be found unsatisfactory. For instance, if a column is placed such that beams (or girders) will not frame into the column in two orthogonal directions, the column will be laterally unsupported, unless the floor slab provides adequate lateral restraint. As another example, a column location may lie within the influence zone. The aim of the *testing* phase is to detect all such inconsistencies. All potential solutions are checked to see if any constraint is being violated. If a solution contains one or more column locations within the no column zone, it is earmarked for modification phase; otherwise the generated column locations are acceptable and define a feasible vertical support system outside the core.

If the testing phase discovers any column placements that violate some constraints, an attempt is made to modify the solution in the *modification* phase. For instance, if a column is placed at an edge of a lobby, then it may be possible to salvage the solution by replacing such a column with two columns at the two ends of the edge on which the column lines intersected. A distributor beam can be placed between the new columns which will also provide vertical support for linear horizontal elements in the direction perpendicular to the edge. This is diagrammatically illustrated in Fig. 4.5. Such modifications, however, may not always be possible. If the modification phase is successful, the modified solution is retained; otherwise the generated solution is discarded.

Column Locations Through Consideration of Areas Inside the Core

For each solution remaining after the testing and modification phases, appropriate column locations within the core can be determined. The operative concept when placing columns in the core is partitions. Partitions between different areas of the core yield ideal locations for columns—provided that the columns do not interfere with the functionality of some areas. Note that some columns at the core boundary or even inside the core may have already been placed prior to this stage as a result of intersection of column lines as described in the previous section. These columns have to be considered when deciding on additional column locations for the core.

Through reasoning about the geometry of the core, one can extract all the partitions in the core. Such partitions are potential column lines. The criterion of minimum beam span may be used to eliminate some parallel column lines that are too close to each other. The generation of column locations from the remaining column lines by intersection of

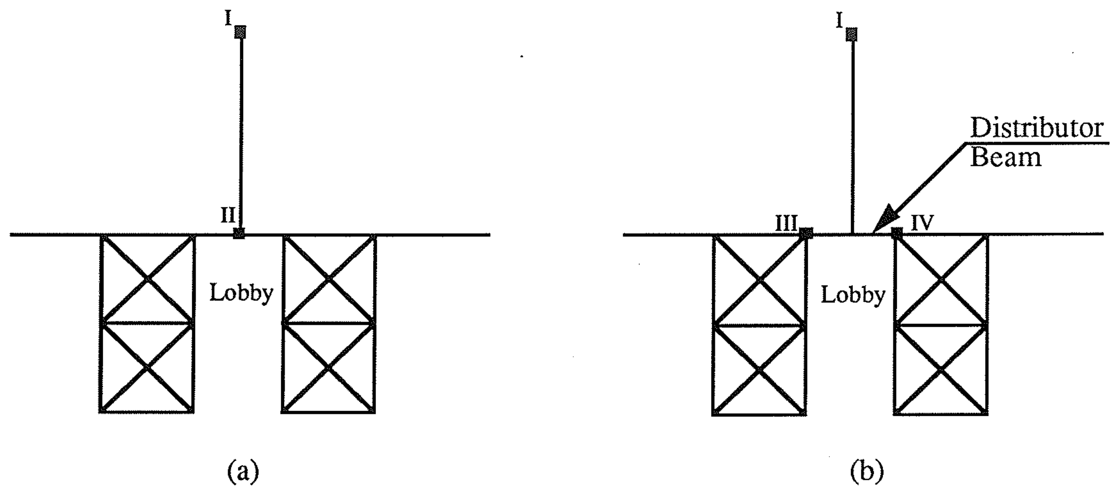


Figure 4.5: **Modification of Column Lines Based on Interference with Lobby.** (a) Column II is on the edge of a no column zone. (b) The intruding column is replaced by two other columns and a distributor beam.

orthogonal column lines is a straightforward operation.

4.8.2 Configuration of the Floor System

Upon generation of feasible alternatives for the vertical support system, we can reason about the floor system for each such alternative individually. The biggest zone of the plan is considered first since it has the potential of having the maximum impact on cost. A zone is defined as a contiguous part of the plan where conditions are the same everywhere and thus can be considered as a single entity for reasoning. Thus, orientations and spacings of beams and girders in a zone will be the same throughout. [Two alternative orientations of beams and girders are explored: (i) beams in the NS direction and girders in the EW direction, and (ii) beams in EW direction and girders in the NS direction.] For the biggest zone, a beam spacing within the range of minimum and maximum economic beam spacing is determined so that there are an integer number of full spans. A corresponding deck gauge and depth can be selected for such spacing based on spanning capabilities of decks as contained in the product knowledge data. The thickness and the depth of the deck will be the same throughout the plan for a particular solution.

The next step in the process is to choose the type of concrete, either lightweight or normalweight. Lightweight concrete, in general, has higher unit cost. However, since it needs less thickness of slab to meet fire resistance requirements, the required volume of lightweight concrete is less. Lightweight concrete can result in savings on other counts also, such as savings in material for other structural elements because of the reduced dead load, and savings in cladding costs because of reduced height of the building. For the lightweight concrete alternative the cost savings due to reduced concrete volume and reduced column steel are computed explicitly. Other savings (in flexural members, foundations, cladding, etc.) are estimated presently as 5% of the unit cost of lightweight concrete. Depending upon which alternative is less expensive at the end of this estimation process, lightweight or normalweight concrete is selected.

The orientation of the beams and girders is such that the girders align with the column lines (and thus rest on the columns) while the beams are in the perpendicular direction, supported at the end by either the girders or the columns. In any given zone beams are spaced apart at a uniform distance consistent with the economic range of beam spacings. (Different zones within the same framing plan, however, may use different beam spacings.)

4.8.3 Design of Members

Two grades of steel are considered for beams and girders as well as columns: A36 and steel with a yield strength of 50 ksi (referred to as A50 in this dissertation). The user may restrict the choice of steel grade to one of the two above, or may ask the system to explore both alternatives and select the least expensive one. For simplicity in sizing of structural members around the service core, the core is treated as a hole for design of gravity systems; i.e., the members on the core boundary are assumed not to carry any loading from spaces inside the core.

Selection of steel grade for beams and girders is based on the controlling criteria for sizing of beams and girders. If the sizing is governed by stiffness criteria, A36 steel is selected. If, on the other hand, strength criteria prevail, sizing is done for both A36 as well as A50 steels and the costs are compared before making a decision. For the chosen steel grade, the members are sized so as to meet both stiffness and strength requirements. Beams and girders may be designed as composite or noncomposite sections. In the case of composite design FFG performs local optimization by opting for partially composite behavior (and providing fewer shear studs than needed for developing fully composite behavior) if the resultant section is sufficient to resist the flexural demand.

For selecting the steel grade for columns, columns at the top story of the building are sized for both A36 and A50 steel and the cost of the material is estimated. If A50 steel turns out to be cheaper, it is chosen for all columns without further cost comparison. If A36 columns are cheaper in the top story, an estimate of the total column steel weight (for all stories) is made for both A36 and A50 steel by designing the columns in the first story and assuming a linear variation in steel weight from the top story to the first story. The steel weights multiplied by the unit price of steel for the respective grades can then be compared to indicate the cheaper alternative. All columns are subsequently sized for the chosen grade every few stories, depending on the chosen frequency of splicing.

4.9 Evaluation

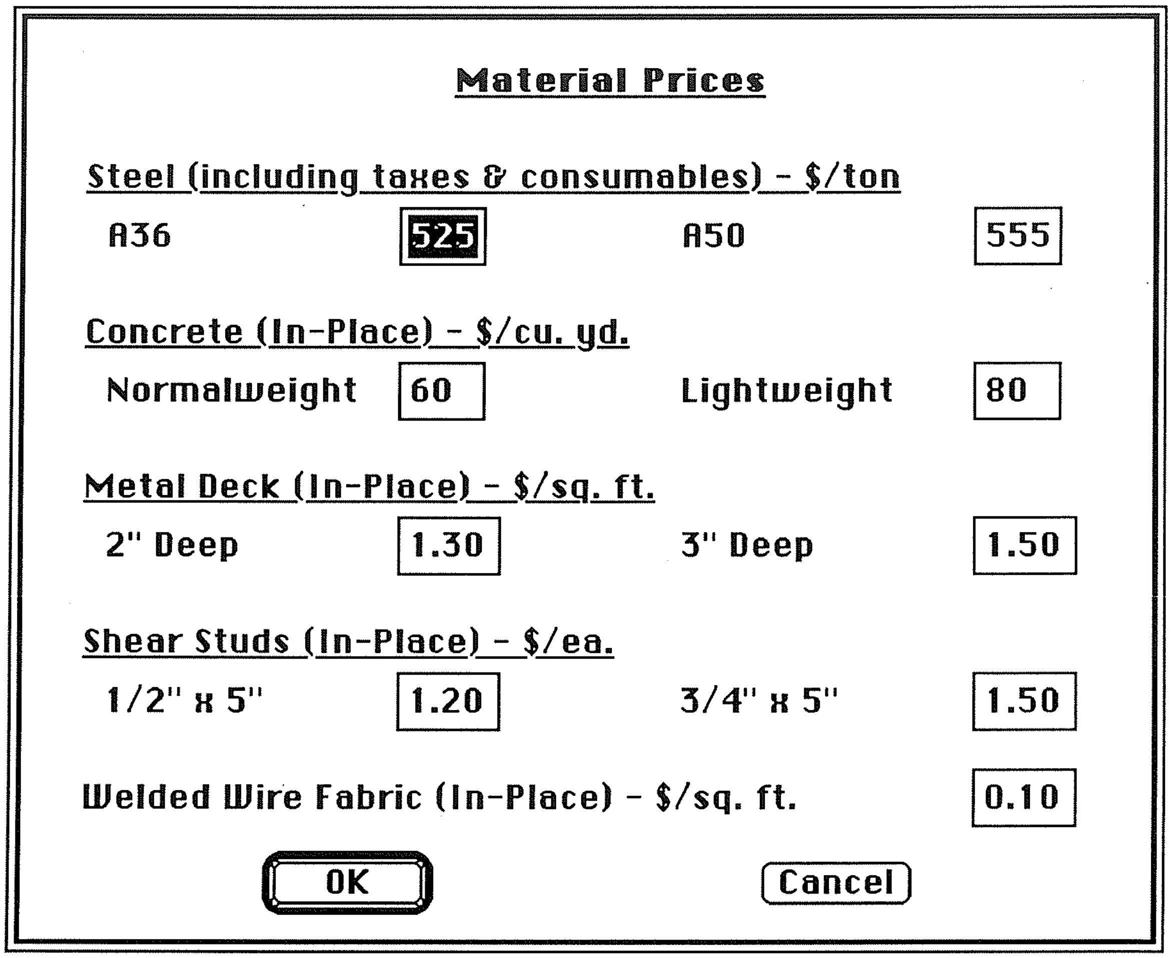
Once candidate solutions are generated, evaluation of the solutions is carried out to associate a measure of cost with them. The items comprising the total cost can broadly be categorized under the following headings.

- *Material Costs*: These include costs of the metal deck, floor concrete, shear studs, welded wire fabric, and the structural steel for beams, girders, and columns. The cost associated with each of these items is computed by multiplying the unit price of the item in question with the corresponding units needed in the structure.

- *Fabrication Costs*: Fabrication cost of an element is based on its type. For instance, the cost of fabricating a column in a moment frame differs from that of a column in a braced frame, which, in turn, differs from that of a gravity support column. Estimating fabrication costs is straightforward once the number of pieces of any given type and the corresponding fabrication price are known. (Fabrication prices are quoted in terms of pieces; thus, if columns are spliced every two stories, a piece refers to a two-story long column.)

- *Erection Costs*: Erection costs are based on the number of pieces to be erected and the project size. Depending upon the total steel weight of the building, the number of erectable pieces per day and the required crew size are determined. Given the data regarding price of erection per person-day, one can compute the total erection cost for the project.

- *Auxiliary Costs*: These are costs associated with shoring, cambering, fire proofing, and transportation. Some of these costs may be zero; for instance, the shoring cost will be zero if unshored construction is used. Shoring and fire proofing costs are based



Material Prices

Steel (including taxes & consumables) - $/ton

A36	525	A50	555

Concrete (In-Place) - $/cu. yd.

Normalweight	60	Lightweight	80

Metal Deck (In-Place) - $/sq. ft.

2" Deep	1.30	3" Deep	1.50

Shear Studs (In-Place) - $/ea.

1/2" x 5"	1.20	3/4" x 5"	1.50

Welded Wire Fabric (In-Place) - $/sq. ft. 0.10

Figure 4.6: **Price Options.** Unit Prices of different materials can be set through the above dialog box. Another dialog box is available for setting other price options (e.g., fabrication, erection, fire proofing, transportation, etc.).

on the floor area. The cambering cost depends on the number of pieces to be cambered whereas the transportation cost is based on the total steel weight.

The default unit price for each of these items is provided, but the user has the option to change any or all of the values (Fig. 4.6). In addition to the cost evaluation of different alternatives, information regarding (i) steel weight per square foot of the building, and (ii) steel construction cost per pound of steel is also furnished to the user.

4.10 Summary

FFG illustrates that the methodology presented in Chapter 3 can be successfully applied to specific problems in the domain of conceptual structural design. Reasoning in terms of function, FFG establishes the need for structural elements based on the match between the requirements of the facility and the capabilities of structural elements. First principles behavior knowledge coupled with performance knowledge is used for the design of individual components, keeping in consideration what products are available for construction. FFG makes its decisions so as to satisfy many different types of constraints originating from diverse sources, including the structural and mechanical engineers, the architect, the contractor, and the owner.

One feature of FFG is that the need for recognizing architectural patterns is acknowledged. Functional organization of the elements of an architectural plan can form some patterns. Detection of such patterns and their incorporation in the structural system is a desirable goal and FFG works towards that goal. Also, FFG operates at a more comprehensive level of knowledge than some of its earlier counterparts. For example, it has knowledge about openings, planning module, and partitions and can reason about the location of vertical support systems in terms of such concepts. Thus, FFG recognizes that columns cannot be placed within an area like a lobby and tries to modify locations of any tentative columns violating such criteria. Similarly, it avoids passing linear horizontal elements through shafts, staircases, and elevator banks, which are openings in the floor.

Nevertheless, in many respects FFG is still incomplete. Allowing different load intensities in different areas of the floor plan, for instance, will enhance the utility of the program. Furthermore, the generated framing plans are applicable only to typical floors. Considerations will be quite different for, say, a mechanical floor.

There are several additional concepts that human designers employ that are not incorporated in the approach discussed in this chapter. One such concept is symmetry. One can enhance the strategy to include a mechanism that gives preference to symmetric framing configurations. Another useful enhancement will involve generating solutions that include nonorthogonal linear horizontal elements as well. Additional concepts will have to be identified and incorporated for this purpose. It is important to note that all such enhancements can be carried out without modifying the pure declarative representation of fundamental domain knowledge contained in the *Functional Elements and Systems Knowledge*, *Product Knowledge*, and *Behavior and Performance Knowledge* modules.

Conceptualizations are our inventions, and their justification is based solely on their utility. This lack of commitment indicates the essential ontological promiscuity of AI: Any conceptualization of the world is accommodated, and we seek those that are useful for our purposes.

— *Michael R. Genesereth and Nils J. Nilsson,* Logical Foundations of Artificial Intelligence *(1987)*

Computers are good at following instructions, but not at reading your mind.

— *Donald E. Knuth,* The TEXbook *(1984)*

5 Lateral Load Resisting System

As was discussed in Chapter 2, in our approach the conceptual design process is decomposed into several phases, starting with design and evaluation of gravity system options, followed by design and evaluation of lateral system options, and concluded by a third phase in which the influences of the two subsystems on each other are evaluated and, conceivably, the subsystems are modified in order to arrive at efficient combined systems. It is anticipated that in future developments this sequential decomposition of the tasks will be altered and the two subsystems will be developed in parallel, using more advanced behavior-based and case-based reasoning techniques to perform simultaneous reasoning on both structural subsystems.

For several reasons we have decided on sequential design of the two subsystems and to focus primarily on the design of the gravity system. Firstly, a sequential design approach can be implemented easier with the present state of knowledge. Secondly, according to the primary domain expert consulted in this research, this approach emulates more closely the process employed in present design practice. Thirdly, and perhaps most importantly, the emphasis of this work is not on the development of a comprehensive computer tool but on formalizing the reasoning process, studying the feasibility and limitations of a logic-based representation and reasoning scheme, and developing a simple prototype that demonstrates the potential of the approach and may be useful as a starting point for further development. The FFG module of Galileo, which focuses on a comprehensive design of the gravity system, is intended to fulfill these objectives. Conceptual design of the lateral load resisting system is only partially implemented in Galileo, and the third phase, reconciliation of gravity and lateral systems, is not part of this work.

From all the lateral schemes illustrated in Fig. 2.2, only the option of moment resisting frames at the perimeter is implemented in Galileo. For each of the generated gravity system alternatives for a given problem, perimeter frames are designed to resist wind and/or seismic loads defined in the behavior and performance knowledge module. No changes are made to the locations of columns generated at the perimeter of the gravity systems. In the evaluation process, the total costs of the resulting combined systems as well as the cost premiums associated with resisting lateral loads are estimated.

This partial implementation is intended to demonstrate that the general methodology, which was implemented comprehensively in the context of gravity systems, can be applied effectively to the design of lateral systems also. This chapter, organized in a manner similar to the preceding chapter, presents the salient aspects of lateral system design in Galileo.

5.1 Functional Elements and Systems Knowledge

Existence of lateral loads on a building (due to wind or earthquake) above the ground implies a need for some structural system capable of carrying the loads to the ground. The only system Galileo's *Functional Elements and Systems Knowledge* library knows about is a moment resisting frame. (An exhaustive library will include knowledge about the other alternatives shown in Fig. 2.2.) The description of moment resisting frame follows.

Moment Resisting Frame

Function	To transfer the lateral loads and tributary gravity loads.
Primary Behavior	Overturning moment resistance through axial forces in columns in the direction of the applied loading; story shear resistance through bending in columns and beams.
Other Behaviors	Shear in members
Form	2D Vertical
Possible Materials	Steel, Reinforced Concrete
Supported By	Foundation

Since lateral loads can assume any direction, moment resisting frames in at least two different directions are required. Galileo places the frames in two orthogonal directions corresponding to the two axes of the building plan. Furthermore, the locations of frames coincide with the perimeter of the building. The decision to place the frames on the perimeter is part of *Strategy*, which also ensures that, for a given solution, column locations for any such frame are identical with those generated earlier for the gravity system. The ends of the frames are taken to be rigidly connected to the ground.

5.2 Product Knowledge

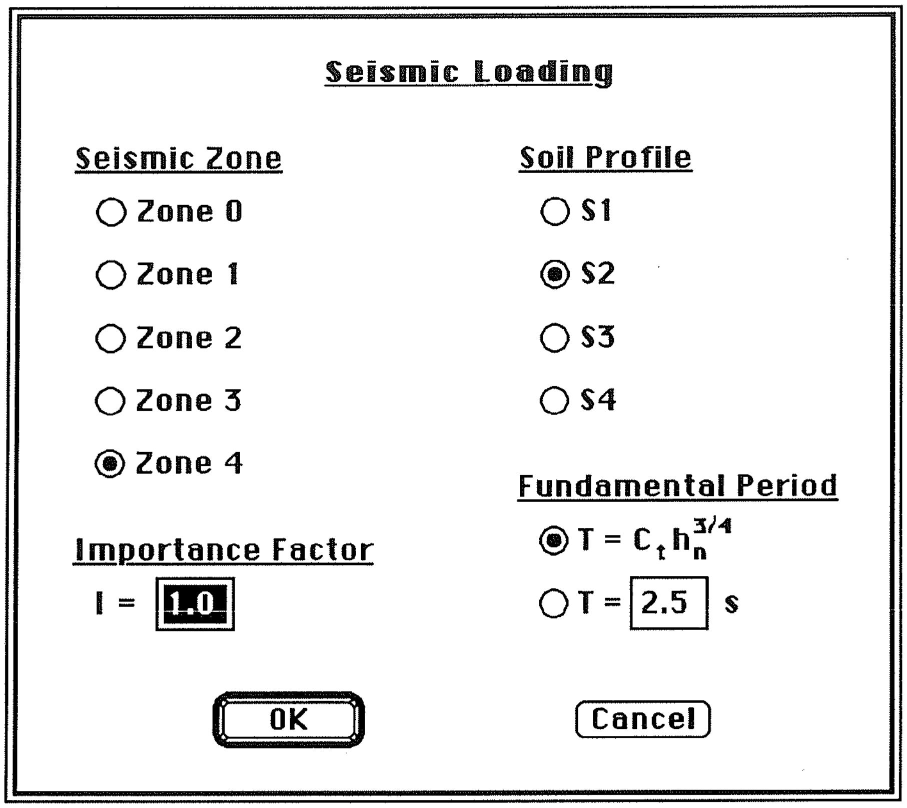
Product knowledge required for lateral system design is a subset of the product knowledge needed for gravity system design. Recall that, in addition to properties of decks, properties of hot-rolled wide-flange steel sections were represented in the *Product Knowledge* module for the gravity system. Besides sectional attributes like moment of inertia, these properties included bending, shear, and axial capacities as well. Sizing of individual elements of the lateral system requires exactly the same quantities. Hence, no additional product knowledge needs to be represented for the lateral system.

5.3 Behavior and Performance Knowledge

Behavior and performance knowledge is used for determining the magnitude of lateral loads acting on the structure, the distribution of such loads, individual member forces due to these loads, and the sizes of members that would meet the demands of safety and serviceability. For sizing individual members, some provisions are common to both gravity and lateral systems design. Some pertinent examples of performance and behavior knowledge follow. (The notation here is consistent with 1988 UBC [40].)

- Design wind pressure: $p = C_e C_q q_s I$, where $q_s = 0.00256 V_{30}^2$
- Design base shear (seismic): $V = (ZIC/R_w)W$
- Concentrated force at the top: $F_t = 0.07TV$ (but less than $0.25V$) if $T > 0.7$ s, 0 otherwise.
- Accidental seismic torsional moment: Based on a mass displacement of 5% of the building dimension.
- Sample loading combinations

  $1.2D + 1.3W + 0.5L$
  $1.2D + 1.5E + 0.5L$
  $0.9D - 1.3W$
  $0.9D - 1.5E$



Seismic Loading

Seismic Zone

- Zone 0
- Zone 1
- Zone 2
- Zone 3
- [✓ Zone 4]

Soil Profile

- S1
- [✓ S2]
- S3
- S4

Importance Factor

I = 1.0

Fundamental Period

- [✓ $T = C_t h_n^{3/4}$]
- T = 2.5 s

Figure 5.1: **Seismic Loading Options.** Determination of the magnitude of lateral loads can be controlled by setting options.

- Permissible interstory deflection (wind): $\delta \leq h/400$
- Overturning moment: $M_{OT}^{x} = \sum_{i=x}^{n} F_i(h_i - h_x)$

The user can input parameters governing the lateral loading and thus control the magnitude of loads. Options for seismic loading, for instance, are shown in Fig. 5.1.

5.4 Constraints

The geometry constraints which affected column locations for the gravity system indirectly influence the lateral system, since the location of columns on the building perimeter for the lateral system conforms with the placement for the gravity system. In addition, behavior and performance constraints, derived from the knowledge contained in the module described before, control the design of members. Reliability constraints have partially been incorporated implicitly by the governing agencies in deciding on methods for computing lateral loads and determining capacities of sections. Amongst the exogenous constraints, constructibility constraints such as those pertaining to materials, MEP constraints on the depths of beams and girders due to ductwork, and owner constraint on minimum floor to ceiling height are accounted for in lateral system design.

5.5 Concepts

The concepts of *Cost*, *Value*, and *Cost/Value* ratio as described earlier in the dissertation are valid in the context of evaluation of lateral system also. It should be emphasized once again that in the absence of any formal methods for objectively estimating *Value*, Galileo computes only the *Cost* part and relies on user's judgment for selecting a solution based on the perceived *Value*s of different alternatives.

One additional concept introduced by the lateral system is that of premium for resisting lateral loads. The premium is defined as the difference between the overall system's cost (resisting both gravity and lateral loads) and the gravity system's cost, expressed as a percentage of the cost of the gravity system. An example in Chapter 7 illustrates the concept of lateral loads cost premium.

5.6 Strategy

Several simplifying assumptions have been made for the purpose of analyzing and designing lateral systems. They are noted throughout this section wherever appropriate. One such assumption is that the columns are so aligned that their webs lie in the respective frame directions. Corner columns are so aligned that the shorter side of the building plan coincides with the direction of the web of the column sections. If the plan is a square, the web of corner columns is aligned with the North-South direction.

5.6.1 Analysis

Modified portal method is used for computing the member forces due to lateral loads. In ordinary portal method, story shear at any given story is distributed to columns in a 2:1 ratio for interior and exterior columns respectively. The shear distribution in the modified portal method is based on the relative stiffnesses of beams, which is taken to be in inverse proportion to their spans. Galileo still takes axial force in interior columns due to lateral loads to be zero, though ideally a free body analysis will give more accurate results. The results of the modified and unmodified portal methods will be identical if all the bays in a frame have the same span.

Instead of a detailed and accurate analysis to determine the member forces in the frame due to gravity loads, the fixed end moments multiplied by a factor of 1.1 are used as the estimates of vertical load moments at beam and girder ends. Any unbalanced moment at a joint is divided equally amongst the columns framing into the joint.

For wind load analysis, only wind acting orthogonal to the faces of the building is considered and effects of non-orthogonal wind are not explored. For gravity load analysis, unbalanced live loads are not considered when computing the demands on various members.

5.6.2 Member Sizing

Framing members in the second story from the ground and the second story from the top are designed for the load combinations mentioned earlier in this chapter. Section weights for other stories are then estimated through linear interpolation.

In addition to satisfying the strength criteria by successfully resisting the factored moments due to gravity and lateral loads computed in the analysis step, the beams and girders must also satisfy deflection criteria and meet restrictions on their moments of inertia. For assessing the required minimum moment of inertia, the contribution of beam and girder flexural deformations to story drift is assumed to be 75%. Composite action of linear horizontal elements, if opted for by the user, is ignored when comparing the required moment of inertia with the actual moment of inertia of a section. It is assumed that column sizes are not controlled by drift considerations.

As per the configuration of perimeter frames, any non-corner exterior column belongs to only one frame. Thus, the joints between such a column and the beams that are in a direction perpendicular to the direction of the frame need not be moment-resisting. Coupled with the fact that the web of the column is aligned with the direction of the frame (a direction in which joints are moment-resisting), one can infer that weak axis buckling is likely to govern the design of these columns. The effective length factor corresponding to the weak axis, K_y, for these columns is taken to be 1.0 for the hinge connection condition. Strong axis buckling of non-corner columns is not checked.

The objective of column design in Galileo is not to determine the precise sections, but to estimate column weights so that total costs can be approximated during the preliminary design stage. Accordingly, Galileo does not carry out a detailed and elaborate sizing of perimeter columns that carry both axial loads as well as moments. Instead, it settles for an approximate method in which the weights of the sections are conservatively estimated. This is done by computing the equivalent effective axial load due to moments (as described in the LRFD Manual) and selecting a W14 section capable of resisting the equivalent effective axial load. Though it is realized that sections deeper than W14 will be more efficient in flexure and thus be better candidates for column sections, this approach is deemed to be adequate because steel material and transportation costs for exterior columns amount to less than 10% of the overall cost. Hence, even when the estimated weight of the columns is off by as much as 30% of the true value, the impact on the total cost will be less than 3%.

Corner columns are checked to see if they will develop tension under the loading combinations of $(0.9D - 1.3W)$ or $(0.9D - 1.5E)$. If so, the columns are flagged and the user is notified of the condition. Strong column weak girder check is reserved for the detailed design phase, since the exact sections chosen for the columns are likely to be different from the W14 sections designed by Galileo.

5.7 Evaluation

Many of the costs computed for the gravity system remain unaffected by lateral load considerations. For instance, the costs of metal deck and floor concrete stay the same (unless diaphragm considerations severely affect the design). On the other hand, steel material cost (for both columns as well as linear horizontal elements), fabrication cost, and transportation cost can change significantly because of lateral loads. Part of the reason is that the sections used for gravity loading alone may not be sufficient to carry the additional lateral loads, causing upward revision in the sizes of sections. This will increase the amount of steel required, affecting both the material costs as well as transportation costs (which are a function of steel tonnage).

Fabrication costs increase because of the need to provide moment resisting connections for lateral system. (Recall that simply supported connections were used for gravity system design.) As was stated in the previous chapter, fabrication costs are based on the type of the piece. Thus, in the case of columns and linear horizontal elements that are part of the lateral load resisting system, a higher unit price is used for computing fabrication costs for the lateral system. The impact of these additional costs is illustrated through an example in Section 7.2.

6 Implementation Issues

Galileo has been implemented in a logic programming environment on a Macintosh II personal computer. The tool used for knowledge representation and automated reasoning in Galileo is Epikit [17], a tool that employs the KIF language for representing knowledge and is written in Common Lisp. The implementation details are the subject of this chapter. As a preface to this discussion, it is worthwhile to mention that the reasoning part of Galileo is hardware-independent and completely portable. The interface part, on the other hand, is hardware-dependent since the code for menus, windows, graphics, etc., is Macintosh-specific. The next section begins by providing a brief overview of Epikit. A general discussion of the user interface concepts and their application in Galileo follows in Section 6.2. We conclude with some closing remarks in the final section.

6.1 Epikit

Epikit [17] is a tool for knowledge representation and automated reasoning. For representing knowledge, Epikit employs KIF, a language based on first-order predicate calculus. (For a brief description of KIF, see Appendix B.) As stated in Ref. [27], the two primary advantages of KIF are:

1. The representation has *pure* declarative semantics; i.e, one can understand the meaning of expressions in a knowledge base without recourse to an interpreter that manipulates those expressions. The same cannot be said of representations based on specific interpreters, such as Emycin and Prolog.

2. The representation is epistemologically comprehensive in that it allows expression of any sentence expressible in first-order predicate calculus. A feel for the expressive

power of KIF can be ascertained from Appendix B. This flexibility distinguishes KIF from Prolog-like representations, which allow only Horn clauses, and production systems, which are restricted to expression of implications.

Thus, KIF provides a convenient mechanism for declaratively specifying the domain knowledge, while inference subroutines in Epikit enable one to deduce inferences from such knowledge. Epikit has a wide spectrum of inference subroutines (based on demodulation, paramodulation, etc.) and several search strategies (including depth-first, breadth-first, best-first, and iterative deepening) for reasoning. In addition, Epikit provides a library of subroutines to create, modify, and examine a knowledge base.

A very useful feature of Epikit is the theory mechanism (a theory in Epikit is simply a collection of facts). One of the uses of the theory mechanism in Galileo is to explore multiple solutions. Alternative solutions generated by Galileo are always mutually inconsistent (if any two solutions agreed on everything and there were no contradictions, they will not be *two* solutions). Since *any* statement can be inferred from an inconsistent set of facts, some mechanism is needed to separate the alternative solutions. Thus, separate theories are used to contain different solutions so that consistency is maintained between all the facts in a single theory.

The organizational structure of the knowledge base can also be made clearer by using theories. For instance, in Galileo there is a theory `product`, which is supposed to include all the product knowledge (see Section 3.4). In the implementation, `product` actually includes several different theories such as `decks` (containing information about the properties of steel decks) and `steel-sections` (containing information about the properties of hot-rolled steel sections). Moreover, the theory mechanism also helps in improving efficiency by permitting one to partition the knowledge base into a theory structure, such that only pertinent facts are searched at any given time. Epikit's inference subroutines can take name of a theory as an optional argument; only facts within the specified theory and its included theories are then searched by the subroutine. Hence Galileo does not search through the properties of composite steel sections when the behavior of beams and girders has been chosen as noncomposite by the user.

One of the strengths of KIF is its forced declarative semantics. In contrast, Prolog constructs like `cut`, `fail`, `asserta`, etc., that lack any declarative semantics and are only intended to control the actions of the interpreter, result in a loss of separation between knowledge and control. The same is true of Common Lisp, which includes constructs like `setf`, `loop`, `progn`, and `rplaca`. Diversions from pure declarative representation also occur in more subtle forms in other languages. Examples include arrangement of rules in a particular order within the knowledge base by the programmer, or ordering of clauses within a rule to exploit the peculiarities of the interpreter. As the examples from Galileo's knowledge base given in Appendix B illustrate, the emphasis on pure declarative semantics in the case of KIF does not restrict the expressive capabilities of the language by any means, but, on the other hand, enhances the transparency and cleanliness of the knowledge base.

6.2 User Interface Issues

In addition to the reasoning aspects, one also has to address the user interface issues when developing a computer system for automating conceptual structural design. This is necessary to ensure the acceptance of the design system by practicing professionals. Besides the obvious benefit of making the experience of using the system a pleasant one, a well-designed interface can also result in improved efficiency through savings in learning and usage times.

The user interface of Galileo has been developed using Allegro Common Lisp for the Macintosh. The interface is interactive and menu-driven; the menus are shown in Fig. 6.1. Menu items can be used to enter the input, set constraints, change the values of parameters, change default options, generate core configurations and structural schemes, and view the output. (Core configurations are normally expected to be provided as part of the input; however, if the project is at a very early stage and all that is known about the plan are the perimeter dimensions, Galileo's library of standard core configurations can be used to generate a configuration based on the plan area and the number of stories.) Some of the relevant aspects of Galileo's user interface are the subjects of the following sections.

6.2.1 Error Detection and Handling

Checks are performed on the input acquired from the user. Early detection of errors can save users' time and prevent erratic behavior on the program's part. Checks can be of several kinds. The simplest ones, *syntactic checks*, relate to the syntax of the input and indicate discrepancies between the expected type of input for a certain parameter and the type of the actual quantity obtained from the user. For instance, the number of stories in a building should be a positive integer; anything else is erroneous and is detected by a syntactic check. *Semantic checks*, on the other hand, detect logical inconsistencies in the input. For example, if the user has provided a value of 20' to be used as the minimum economic span for beams and 15' as the maximum economic span, it is clear that the range

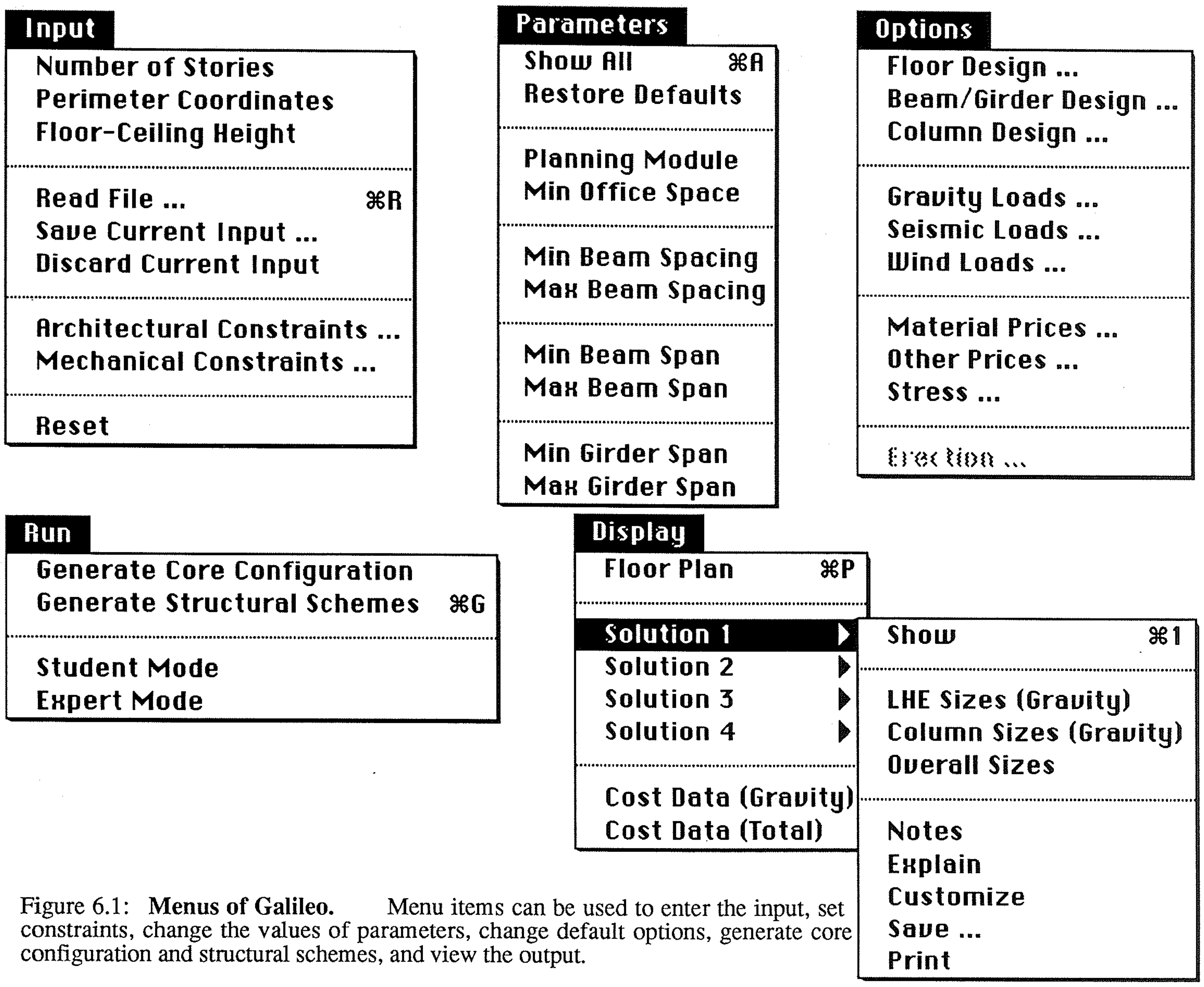


Figure 6.1: **Menus of Galileo.** Menu items can be used to enter the input, set constraints, change the values of parameters, change default options, generate core configuration and structural schemes, and view the output.

of economic spans for the beam is ill-defined. Such inconsistencies are detected through semantic checks. For both syntactic and semantic errors, the user must correct the input before proceeding with the solution stage.

A third type of checks, namely *plausibility checks*, are also useful in engineering applications. Plausibility checks result from the recognition that design parameters usually have a set (or range) of "normal" or plausible values. Continuing the earlier example, a value of 100' as the minimum economic beam span is questionable and is probably a result of user's oversight. As opposed to the other two types of checks, however, the program does not force the user to change the input and permits values outside the plausible range to accommodate atypical cases. But such values are brought to the attention of the user by either issuing a warning, or asking for a confirmation of the entered value.

A convenient interface should also facilitate easy correction of erroneous input. The interactive input commands of Galileo are so built that they can be invoked repeatedly. In case the value of a parameter has already been entered, invoking the corresponding command again results in deletion of the earlier value and retention of the most recently entered value. Thus, any time the user enters an incorrect value, all he or she has to do is to execute the appropriate command again and enter the correct value.

6.2.2 Knowledge Structure and Content

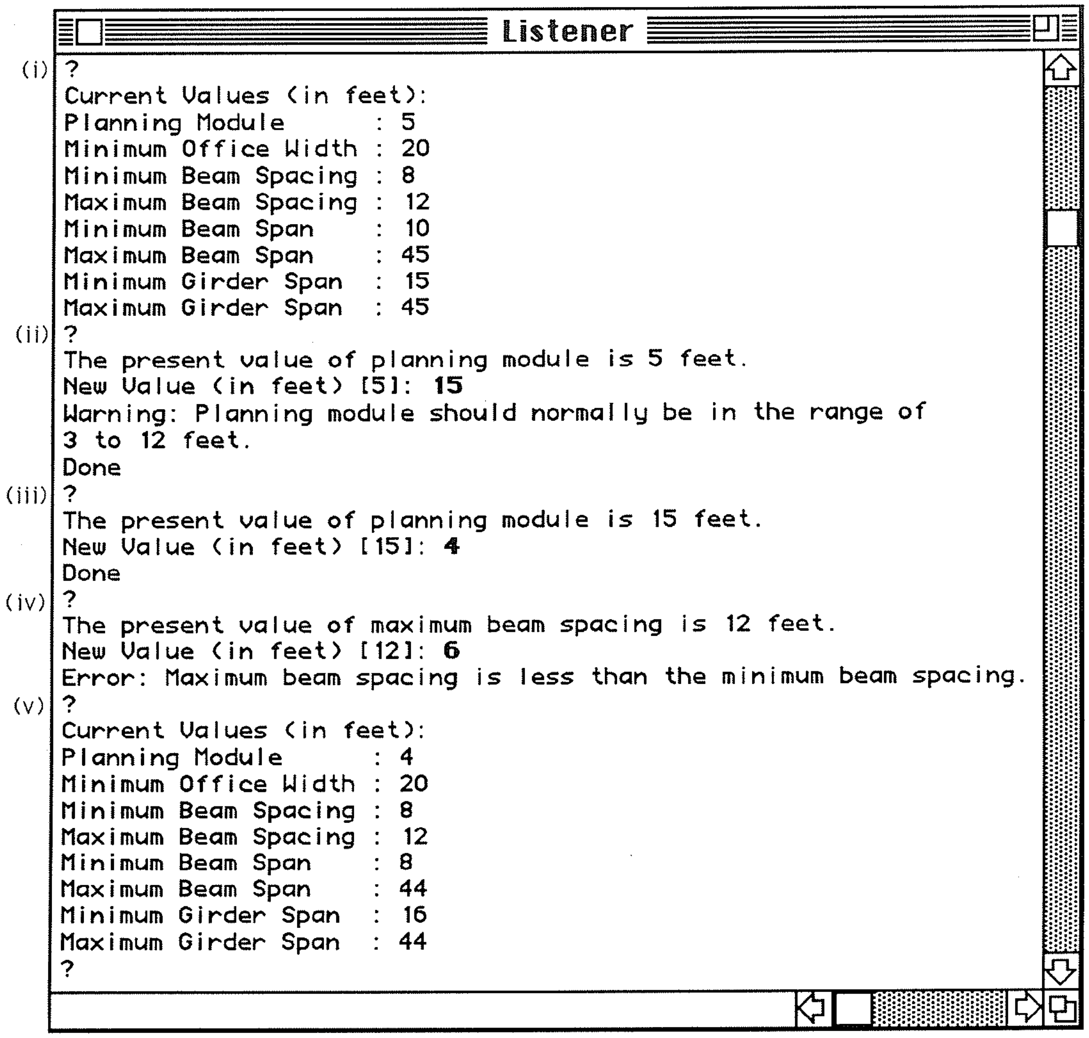
Though knowledge structure and content issues do not fall completely within the purview of interface issues, they are included here to the extent they influence the user interface. One of the most important points in this regard is that heuristic values for quantities that influence design (say, for instance, spanning capabilities of construction materials) should not be "hard-wired" in the system. These values may change with time and technology and thus invalidate the existing knowledge-base. Instead, the parameters that are relevant in the design process have been identified and the knowledge has been structured in terms of these parameters. Thus, instead of the statement

```
IF width_of_the_bay > 40'
...
```

we prefer a more general statement of the form

```
IF width_of_the_bay > maximum_economic_beam_span
...
```

where the parameter `maximum_economic_beam_span` can assume different values at different times. Essentially, the domain knowledge is captured through statements that contain



```
(i)   ?
      Current Values (in feet):
      Planning Module      : 5
      Minimum Office Width : 20
      Minimum Beam Spacing : 8
      Maximum Beam Spacing : 12
      Minimum Beam Span    : 10
      Maximum Beam Span    : 45
      Minimum Girder Span  : 15
      Maximum Girder Span  : 45
(ii)  ?
      The present value of planning module is 5 feet.
      New Value (in feet) [5]: 15
      Warning: Planning module should normally be in the range of
      3 to 12 feet.
      Done
(iii) ?
      The present value of planning module is 15 feet.
      New Value (in feet) [15]: 4
      Done
(iv)  ?
      The present value of maximum beam spacing is 12 feet.
      New Value (in feet) [12]: 6
      Error: Maximum beam spacing is less than the minimum beam spacing.
(v)   ?
      Current Values (in feet):
      Planning Module      : 4
      Minimum Office Width : 20
      Minimum Beam Spacing : 8
      Maximum Beam Spacing : 12
      Minimum Beam Span    : 8
      Maximum Beam Span    : 44
      Minimum Girder Span  : 16
      Maximum Girder Span  : 44
      ?
```

Figure 6.2: **Working with Parameters.** The following commands from the `Parameters` menu were executed respectively at the `?` prompts: (i) `Show All`, (ii) `Planning Module`, (iii) `Planning Module`, (iv) `Max Beam Spacing`, and (v) `Show All`.

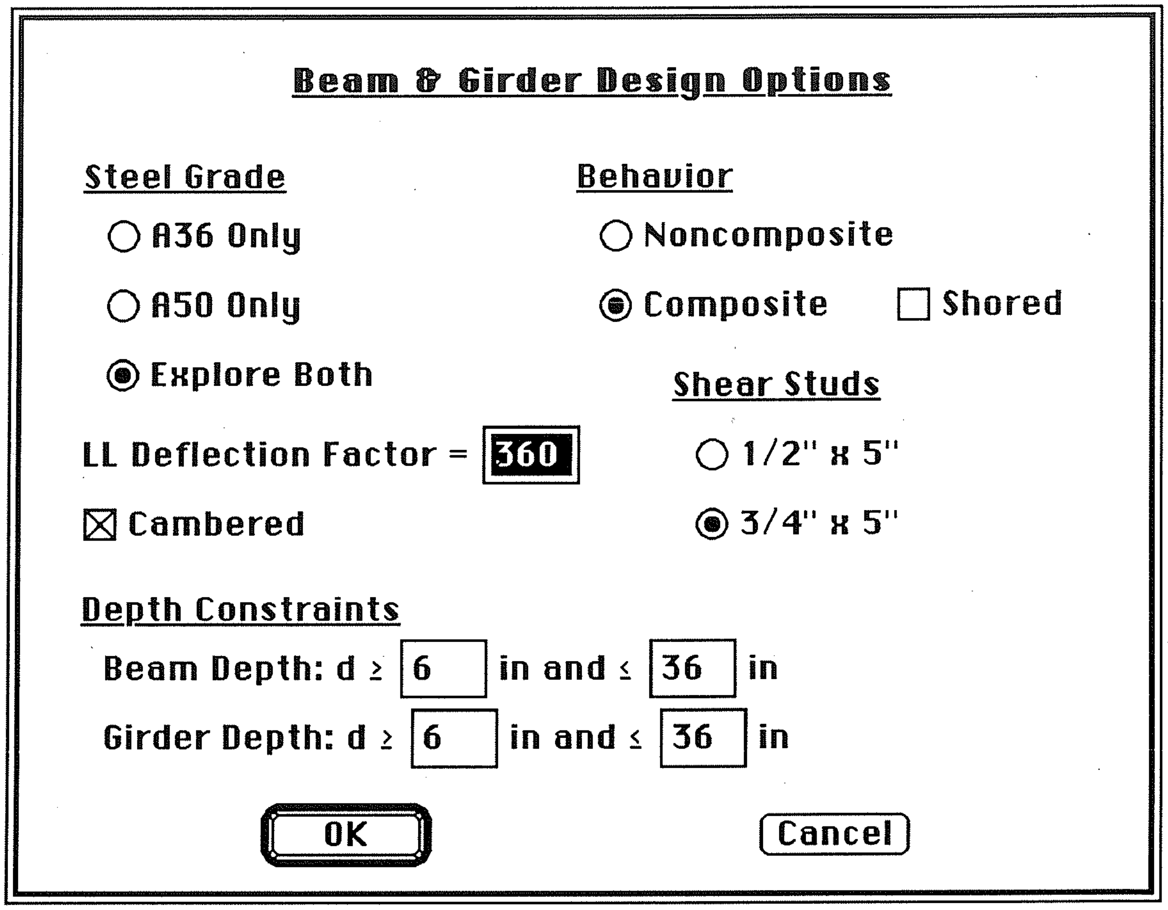
---

variables representing the design parameters. Plausible default values for such parameters are provided (for example the `maximum_economic_beam_span` parameter presumably can have a default value of 45'), but there is a mechanism to enable the users to override these defaults and suggest alternate values according to their preferences. Figure 6.2 shows an example involving a few such parameters and the mechanism for overriding their default values.

6.2.3 Additional Considerations

One visual aspect of the interface relates to windows. Galileo exploits the multiple window capability to present several alternative solutions simultaneously. Though a single window can be used to display all the generated structural schemes, employing separate windows leads to better maneuverability and flexibility in comparing the solutions. Thus, the designer can look at only those solutions that he or she wants to examine without being distracted by the other alternatives in view. Additionally, different types of information about a solution are presented in different windows resulting in a better conceptual organization. To illustrate, while the geometry of a floor framing plan is graphically sketched in one window, a separate window is used to display the textual information (e.g., notes on the construction method) accompanying the solution. A third window presents the chosen sections for various members while yet another window is used for explaining how the solution was arrived at. Once again, though such information can be combined in a single window, distinct windows give the advantage of choice—e.g., ability to concentrate on the information of interest by getting rid of unnecessary windows—while retaining the functionality of a single window, i.e., ability to view all the information at once. Galileo also communicates parts of output through concise tabular summaries (e.g, member sizes, cost data) that facilitate easy comprehension and comparison.

The philosophy behind the development of Galileo has been to make it an "advisable" [25] system. Thus, the system can be forced to forego certain decision-making processes and instead adopt the decisions made by the designer. Figure 6.3 illustrates one such example relevant to beam and girder design in Galileo. An associated concept implemented in Galileo is the idea of *modes* to accommodate users with varying degrees of expertise in the domain. The interaction with the system is different in different modes. For instance, in the *expert* mode the user has access to certain advanced options; for example, quantities like crew size and erectable pieces per day, used for estimating erection costs, can be altered by the user in the expert mode. In the *student* mode the explanation can be at a more elementary level and geared towards learning. Justification that an expert may find trivial or uninteresting may be useful for an inexperienced engineer for learning purposes. As the reader will see in the next chapter, explanations generated by Galileo are not mere traces of "firings of rules," but are more comprehensible and logically organized statements of interest, with a vocabulary consistent with the professionals in the field.



Beam & Girder Design Options

Steel Grade

- ○ A36 Only
- ○ A50 Only
- [✓ ◉ Explore Both]

LL Deflection Factor = 360

[✓ ☒ Cambered]

Behavior

- ○ Noncomposite
- [✓ ◉ Composite] ☐ Shored

Shear Studs

- ○ 1/2" x 5"
- [✓ ◉ 3/4" x 5"]

Depth Constraints

Beam Depth: d ≥ 6 in and ≤ 36 in

Girder Depth: d ≥ 6 in and ≤ 36 in

Figure 6.3: **Options for Beam and Girder Design.** Preferences for beam and girder design can be set through the above dialog box. Other dialog boxes can be used to set design options for floors and columns.

6.3 Closing Remarks

Flexibility is one of the most important guiding principles for good interfaces. Flexibility can be manifested in many ways, including the ability to choose among design strategies, suggest values for design parameters, and the ability to override the system's decisions. Another important consideration is consistency. Consistency can be apparent in the organization of menus, behavior of various commands, layout of dialog boxes, etc. Effort has been made to adhere to these principles while designing the user interface of Galileo.

The discussion in this chapter was designed to give the essence of the implementation issues involved in Galileo's development, without deluging the reader with low-level details.

Finer details are available in Appendix C.

Theorists conduct experiments with their brains. Experimenters have to use their hands, too. Theorists are thinkers, experimeters are craftsmen. . . . The theorist operates in a pristine place free of noise, of vibration, of dirt. The experimenter develops an intimacy with matter as a sculptor does with clay, battling it, shaping it, and engaging it.

— *James Gleick,* Chaos: Making a New Science *(1987)*

You need an arsenal of all-purpose irrelevant phrases to fire back at your opponents when they make valid points. The best are:

You're begging the question.

You're being defensive.

Don't compare apples and oranges.

What are your parameters?

The last one is especially valuable. Nobody, other than mathematicians, has the vaguest idea what "parameters" means.

— *Dave Barry,* How to Win Arguments, As It Were

7 Examples

This chapter presents two representative examples of problem-solving sessions with the design system developed in this investigation. The first example is restricted to floor framing generation (using the FFG part of Galileo). The next example is more comprehensive and is designed to show some features of Galileo that are not illustrated in the first example.

7.1 Example I

The architectural plan shown in Fig. 4.1(a) is adapted from the plan of a real ten-story building and is used as a sample input to illustrate FFG's working. The architectural layout of the floor is based on a 4 ft. planning module. Accordingly, the planning module in FFG was changed to 4 ft. from the default value of 5 ft. FFG generated four floor framing schemes for the plan, one of which is shown in Fig. 4.1(b). Other framing plans are displayed in Fig. 7.1. Member sizes for the first framing plan shown in Fig. 7.1 follow in Fig. 7.2. Textual notes and explanation for Framing Plan 1 are shown in Figs. 7.3 and 7.4, respectively. A cost comparison of the four alternatives is presented in Fig. 7.5.

It is interesting to compare the solutions generated by FFG with the ones generated by the engineer responsible for the structural design. Of the four schemes generated by FFG, two matched with those generated by the engineer. However, there was one scheme that was generated by the engineer but not FFG; this scheme involved nonorthogonal girders. FFG is capable of generating solutions involving only orthogonal linear horizontal elements.

One of the ways in which FFG can be used is to estimate the effect of changing some design variant(s). For instance, we solved the aforementioned problem for two different cases to obtain a range of the unit steel weight for the resulting structural systems. First,

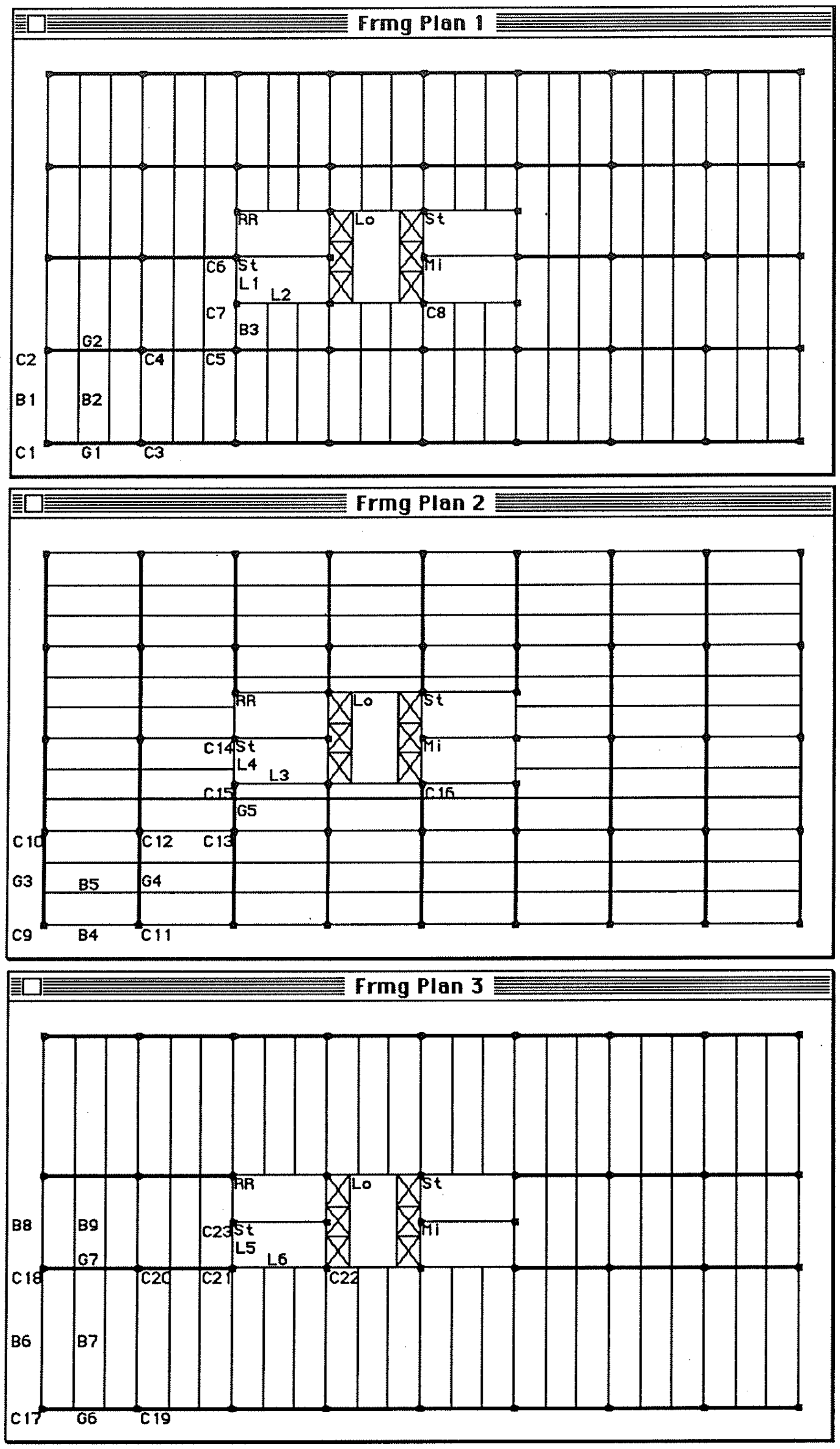
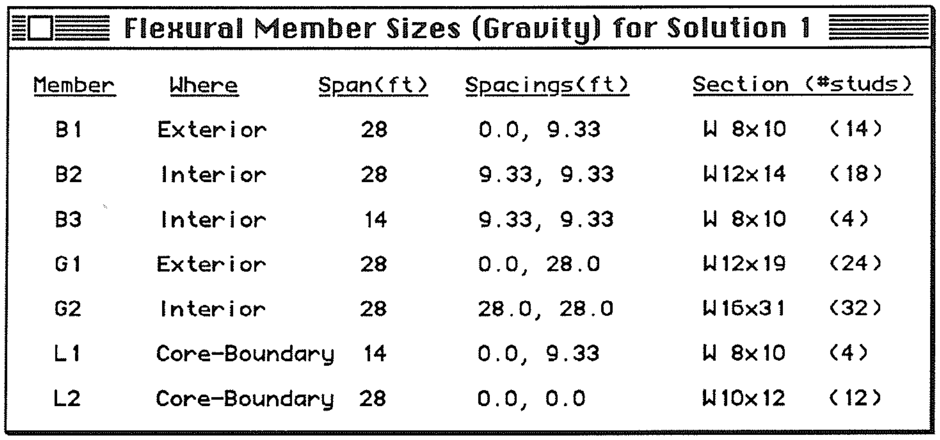
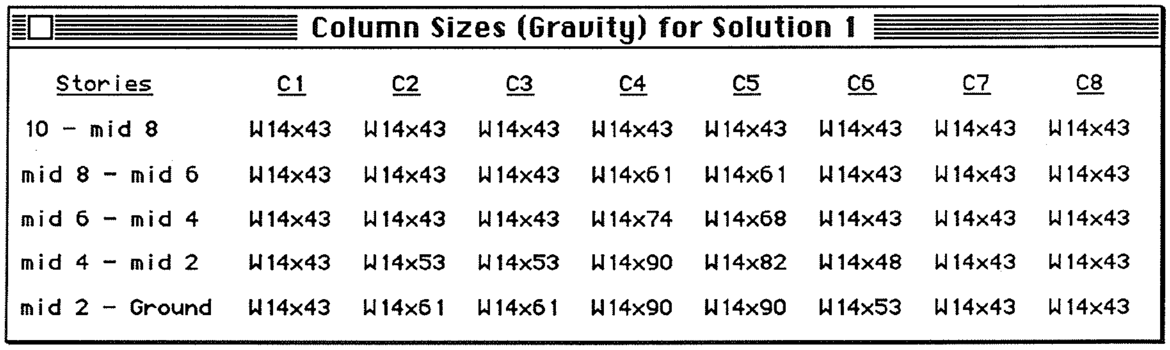


Figure 7.1: **Graphical Output of Framing Plans Generated by FFG.** Sizes of columns and flexural members of Framing Plan 1 are shown in subsequent figures.


Flexural Member Sizes (Gravity) for Solution 1

Member	Where	Span(ft)	Spacings(ft)	Section (#studs)	
B1	Exterior	28	0.0, 9.33	W 8x10	(14)
B2	Interior	28	9.33, 9.33	W12x14	(18)
B3	Interior	14	9.33, 9.33	W 8x10	(4)
G1	Exterior	28	0.0, 28.0	W12x19	(24)
G2	Interior	28	28.0, 28.0	W16x31	(32)
L1	Core-Boundary	14	0.0, 9.33	W 8x10	(4)
L2	Core-Boundary	28	0.0, 0.0	W10x12	(12)

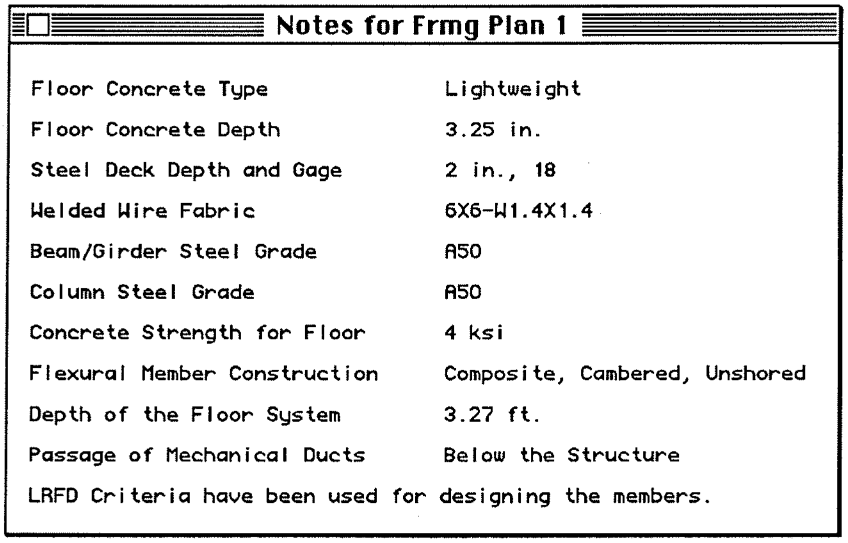
(a)


Column Sizes (Gravity) for Solution 1

Stories	C1	C2	C3	C4	C5	C6	C7	C8
10 - mid 8	W14x43	W14x43	W14x43	W14x43	W14x43	W14x43	W14x43	W14x43
mid 8 - mid 6	W14x43	W14x43	W14x43	W14x61	W14x61	W14x43	W14x43	W14x43
mid 6 - mid 4	W14x43	W14x43	W14x43	W14x74	W14x68	W14x43	W14x43	W14x43
mid 4 - mid 2	W14x43	W14x53	W14x53	W14x90	W14x82	W14x48	W14x43	W14x43
mid 2 - Ground	W14x43	W14x61	W14x61	W14x90	W14x90	W14x53	W14x43	W14x43

(b)

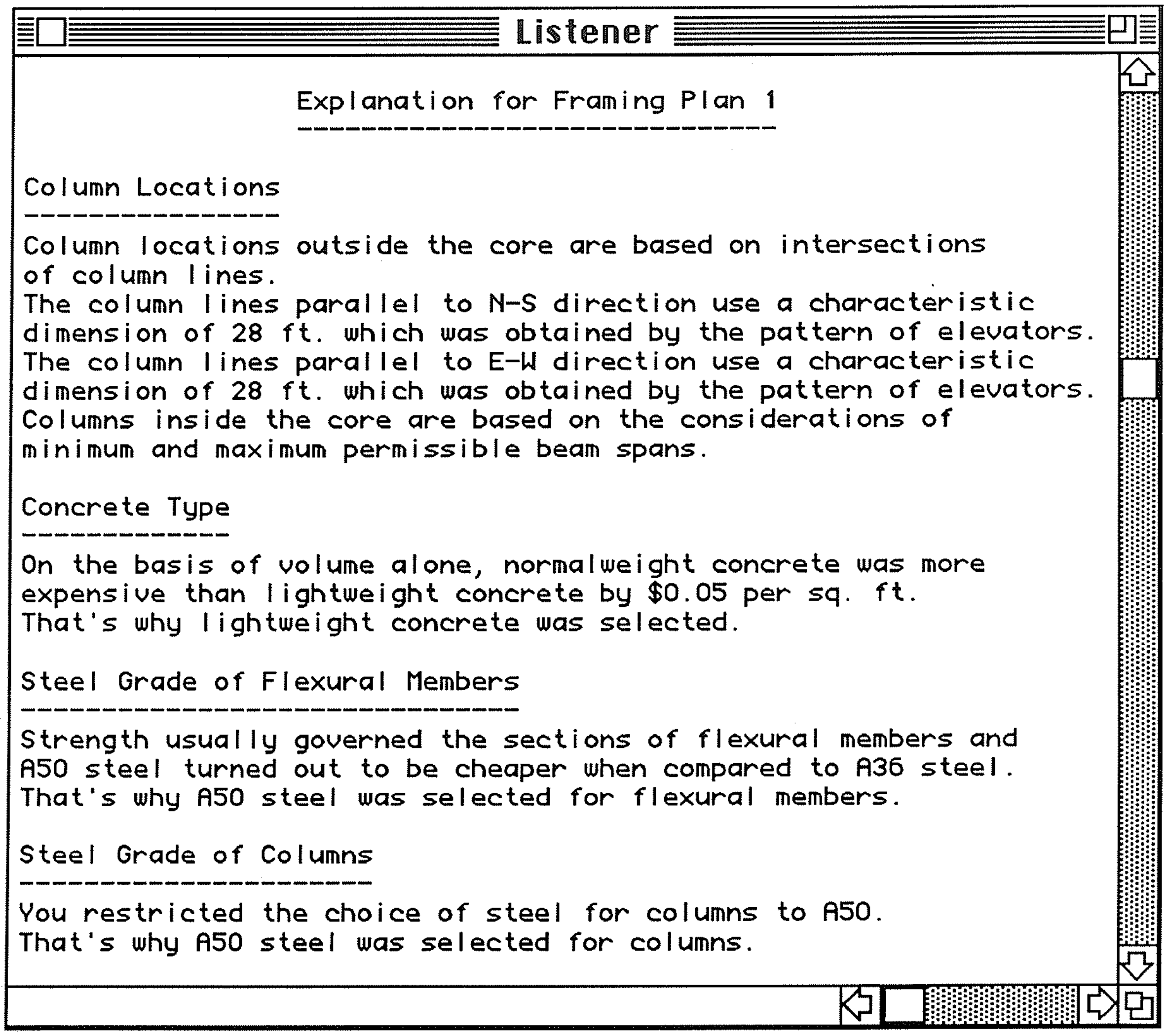
Figure 7.2: **Member Sizes for Framing Plan 1.** (a) Flexural member sizes. (b) Column sizes.


Notes for Frmg Plan 1

Floor Concrete Type	Lightweight
Floor Concrete Depth	3.25 in.
Steel Deck Depth and Gage	2 in., 18
Welded Wire Fabric	6X6-W1.4X1.4
Beam/Girder Steel Grade	A50
Column Steel Grade	A50
Concrete Strength for Floor	4 ksi
Flexural Member Construction	Composite, Cambered, Unshored
Depth of the Floor System	3.27 ft.
Passage of Mechanical Ducts	Below the Structure

LRFD Criteria have been used for designing the members.

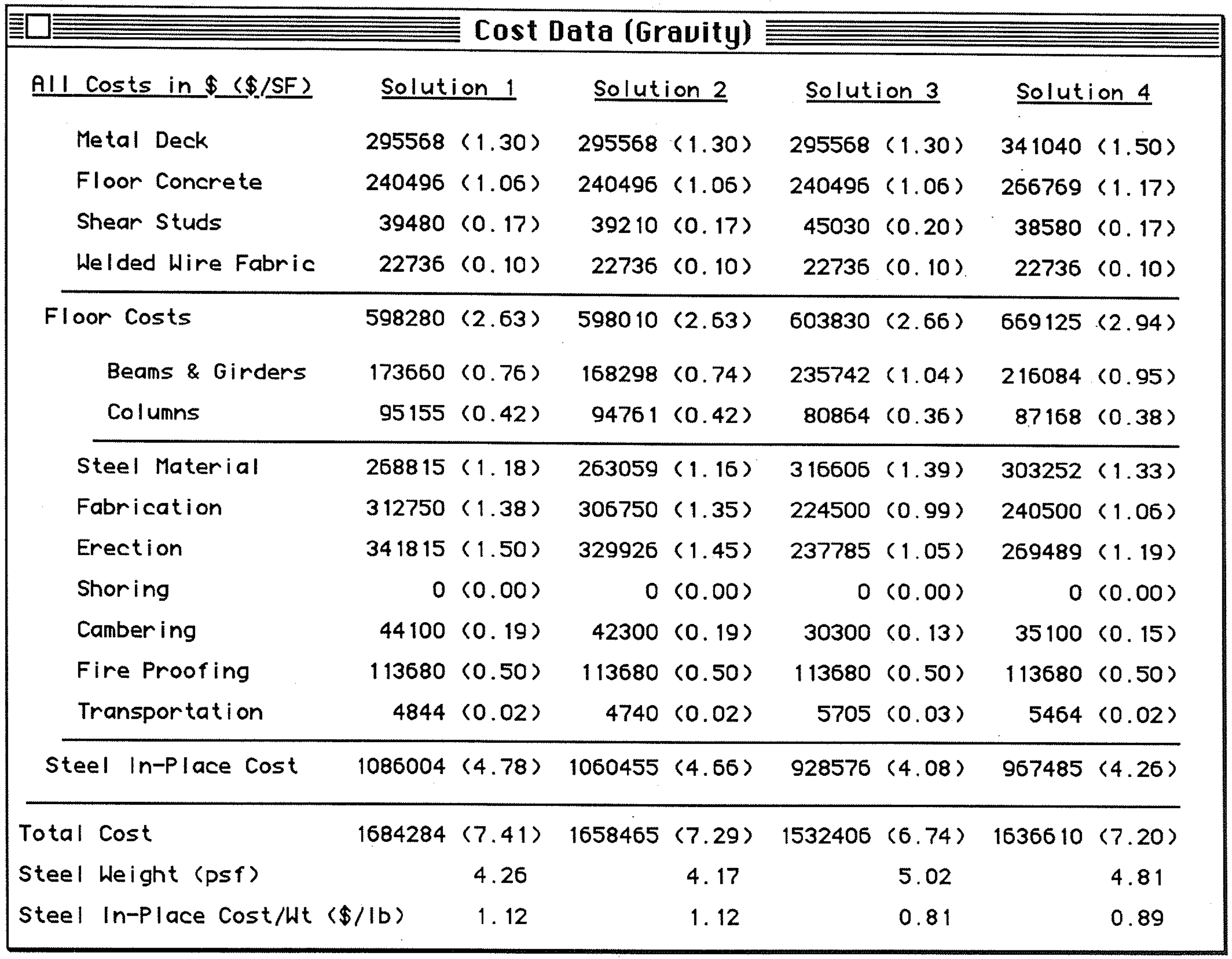
Figure 7.3: **Textual Notes for Framing Plan 1.** This auxiliary information accompanies Framing Plan 1.



```
                 Explanation for Framing Plan 1
                 ------------------------------

Column Locations
----------------
Column locations outside the core are based on intersections
of column lines.
The column lines parallel to N-S direction use a characteristic
dimension of 28 ft. which was obtained by the pattern of elevators.
The column lines parallel to E-W direction use a characteristic
dimension of 28 ft. which was obtained by the pattern of elevators.
Columns inside the core are based on the considerations of
minimum and maximum permissible beam spans.

Concrete Type
-------------
On the basis of volume alone, normalweight concrete was more
expensive than lightweight concrete by $0.05 per sq. ft.
That's why lightweight concrete was selected.

Steel Grade of Flexural Members
-------------------------------
Strength usually governed the sections of flexural members and
A50 steel turned out to be cheaper when compared to A36 steel.
That's why A50 steel was selected for flexural members.

Steel Grade of Columns
----------------------
You restricted the choice of steel for columns to A50.
That's why A50 steel was selected for columns.
```

Figure 7.4: **Explanation for Framing Plan 1.** FFG can explain its decisions in a vocabulary consistent with structural engineering terminology.

to obtain the lightest system, the grade of steel for all members was restricted to A50, composite behavior was chosen for beams and girders, and concrete was selected to be lightweight. The resulting schemes had unit steel weight ranging from 4.24 to 5.08 psf (pounds per sq. ft.) and costs ranging from 6.74 to 7.41 \$/sq. ft. For the other extreme, grade of steel for all members was restricted to be A36, noncomposite behavior was chosen for beams and girders, and concrete was selected to be normalweight, with everything else being the same. The resulting schemes this time had unit steel weight ranging from 7.05 to 8.39 psf and costs ranging from 7.62 to 8.19 \$/sq. ft.



Cost Data (Gravity)

All Costs in $ ($/SF)	Solution 1	Solution 2	Solution 3	Solution 4
Metal Deck	295568 (1.30)	295568 (1.30)	295568 (1.30)	341040 (1.50)
Floor Concrete	240496 (1.06)	240496 (1.06)	240496 (1.06)	266769 (1.17)
Shear Studs	39480 (0.17)	39210 (0.17)	45030 (0.20)	38580 (0.17)
Welded Wire Fabric	22736 (0.10)	22736 (0.10)	22736 (0.10)	22736 (0.10)
Floor Costs	598280 (2.63)	598010 (2.63)	603830 (2.66)	669125 (2.94)
Beams & Girders	173660 (0.76)	168298 (0.74)	235742 (1.04)	216084 (0.95)
Columns	95155 (0.42)	94761 (0.42)	80864 (0.36)	87168 (0.38)
Steel Material	268815 (1.18)	263059 (1.16)	316606 (1.39)	303252 (1.33)
Fabrication	312750 (1.38)	306750 (1.35)	224500 (0.99)	240500 (1.06)
Erection	341815 (1.50)	329926 (1.45)	237785 (1.05)	269489 (1.19)
Shoring	0 (0.00)	0 (0.00)	0 (0.00)	0 (0.00)
Cambering	44100 (0.19)	42300 (0.19)	30300 (0.13)	35100 (0.15)
Fire Proofing	113680 (0.50)	113680 (0.50)	113680 (0.50)	113680 (0.50)
Transportation	4844 (0.02)	4740 (0.02)	5705 (0.03)	5464 (0.02)
Steel In-Place Cost	1086004 (4.78)	1060455 (4.66)	928576 (4.08)	967485 (4.26)
Total Cost	1684284 (7.41)	1658465 (7.29)	1532406 (6.74)	1636610 (7.20)
Steel Weight (psf)	4.26	4.17	5.02	4.81
Steel In-Place Cost/Wt ($/lb)	1.12	1.12	0.81	0.89

Figure 7.5: **Evaluation and Comparison of Alternatives.** As the above example illustrates, a least weight solution need not be the least expensive one. (Structural members and the floor area inside the core are not included in the computations above.

7.2 Example II

This example illustrates several aspects of Galileo not discussed in the preceding section. These include use of the library of standard core configurations, design of the lateral system, and computation of cost premium for carrying the lateral loads.

In contrast with the example in the previous section in which the input was provided both interactively and through a file, all the input to Galileo for this example was supplied interactively. At the minimum, Galileo needs the extents of the perimeter and the number of stories in the building as input. Using the *Perimeter Coordinates* menu item in the *Input* menu, the coordinates of two opposite corners of the perimeter were set to (0, 0) and (110, 60). Through another menu item, the number of stories was set to thirteen.

Galileo includes a library of standard core configurations to assist in the preliminary exploration of framing alternatives. If only the plan dimensions and the number of stories in the building are known, Galileo can be used to generate a core configuration before proceeding with structural design. The configurations in the library are indexed by the number of elevators. There are fifteen configurations in all in the library, ranging in the provision of the number of elevators from two to sixteen. This approach of selecting a core configuration from a predetermined set has limitations, and accordingly the use of library is restricted to buildings between three and sixty stories. Buildings taller than sixty stories will usually need elevator experts to configure the vertical transportation system whereas buildings shorter than three stories will typically not need elevators or will have nonstandard installations. Other restrictions on the length and width of the plan also apply.

As mentioned earlier, the selection of a core configuration is based on the required number of elevators. An assessment of the required number of elevators can be made based on the gross plan area of the building, which in turn can be computed from plan dimensions and the number of stories. Once a standard configuration is chosen, the core is centered within the perimeter and the coordinates of various elements within the core are computed with respect to the plan dimensions. For the given perimeter dimensions and number of stories in the building for this problem, Galileo estimated the required number of elevators to be two, and the resulting floor plan is shown in Fig. 7.6.

Before proceeding with generation of structural schemes, several options were set to highlight the contrast between Examples I and II. Behavior of beams and girders was set to be noncomposite and their steel grade was restricted to be A36. Concrete type for the floor was set to be normalweight and the steel grade for columns was set to be A50. This usage of Galileo differs markedly from Example I in which the system was allowed to make

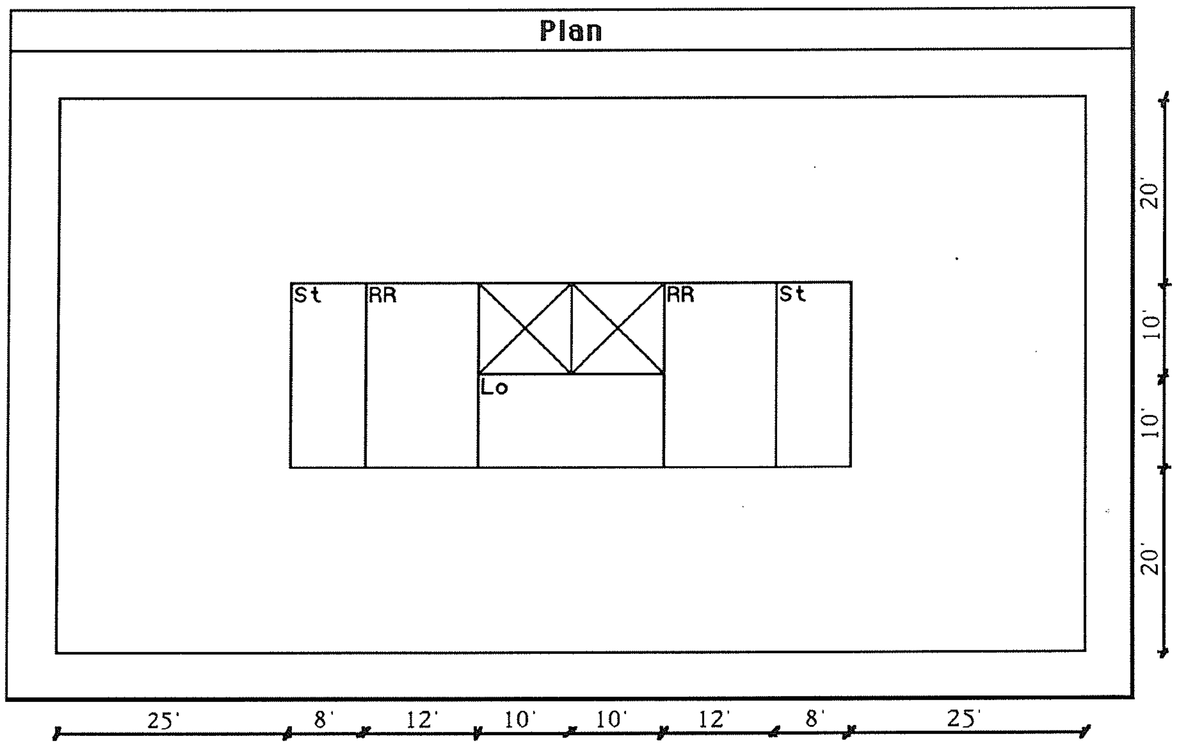


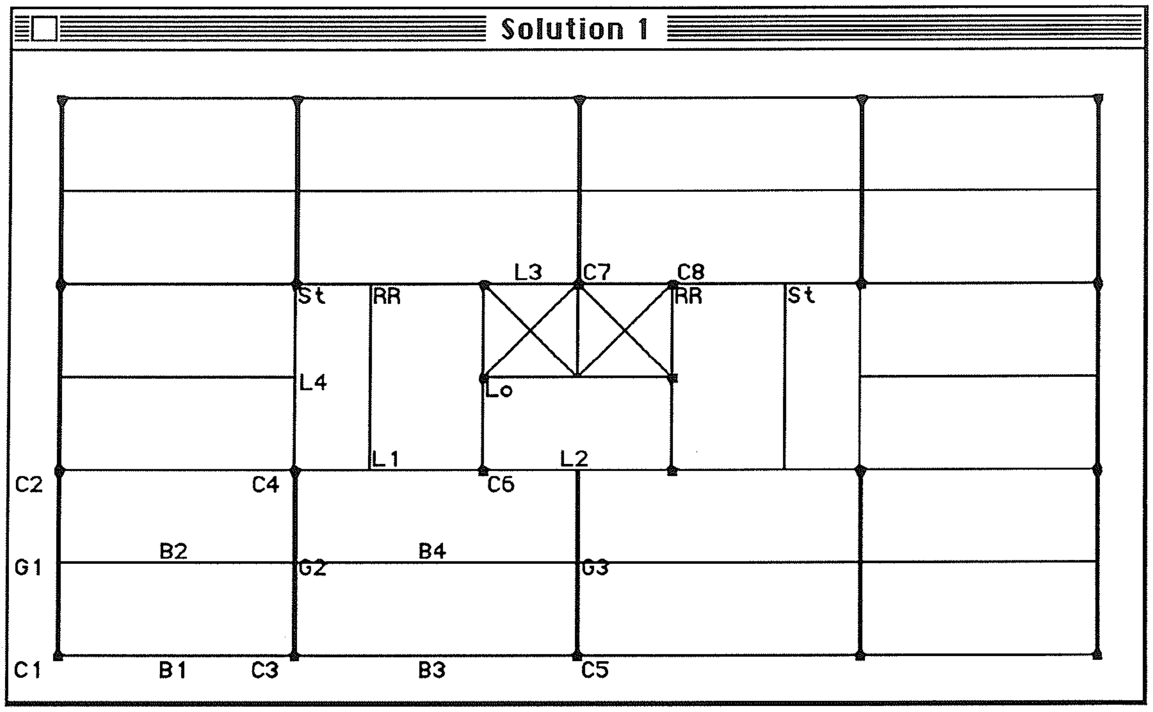
Figure 7.6: **Floor Plan for Example II.** The core configuration for this plan was generated by Galileo.

most of the decisions based on perceived economy; in Example II, a greater control over the functioning of the system was exercised.

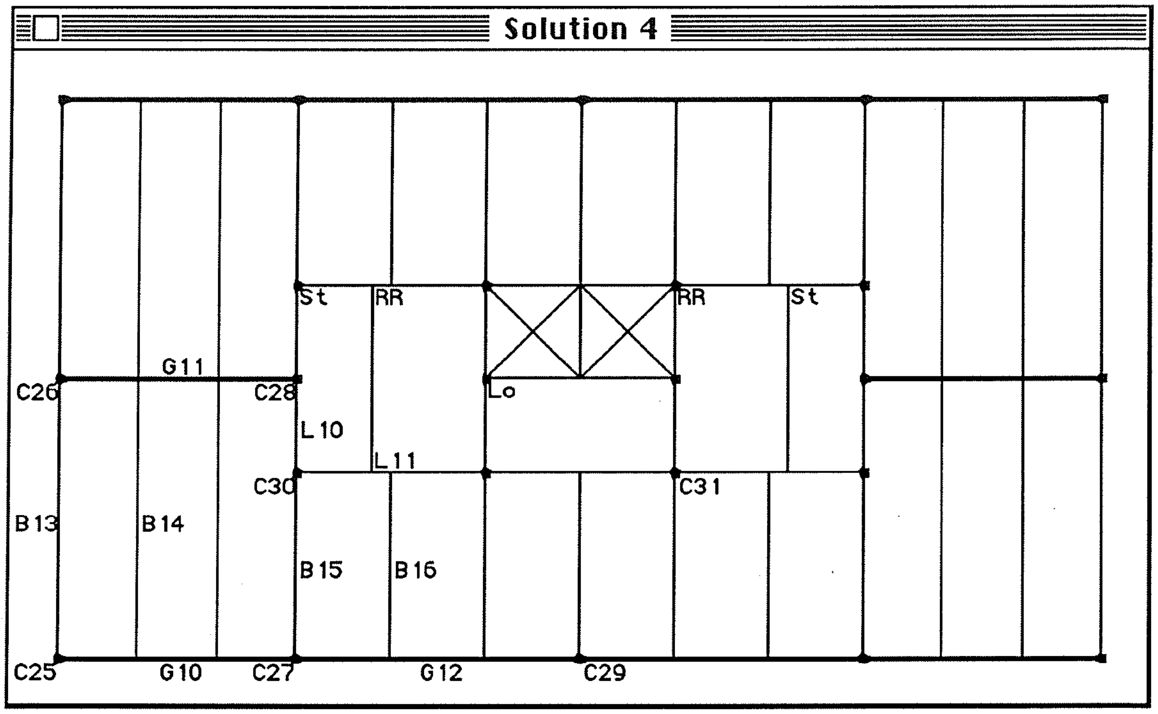
All other options, including the ones for loading, were left unchanged (see Appendix C for default values of options). In particular, the seismic zone for the building location was taken to be Zone 4. Furthermore, exposure type for wind loading was assumed to be B.

Galileo generated four structural schemes for this problem, two of which are shown in Fig. 7.7(a) and 7.7(b), respectively. Both gravity and lateral systems for each of the four alternatives were designed. Member sizes for Solution 1 corresponding to gravity loading alone are shown in Fig. 7.8. Member sizes corresponding to both gravity and lateral loading for the same solution are shown in Fig. 7.9. Galileo indicated that corner columns in all four solutions can be in tension due to lateral loads.

A comparison of the costs of gravity and total system follows in Fig. 7.10. As can be seen from Fig. 7.10(b), the premium for lateral loading using perimeter moment frames for this problem is close to 15% for each of the four alternative solutions. The premium can, of course, be different if a different lateral system (such as a braced frame) is employed.

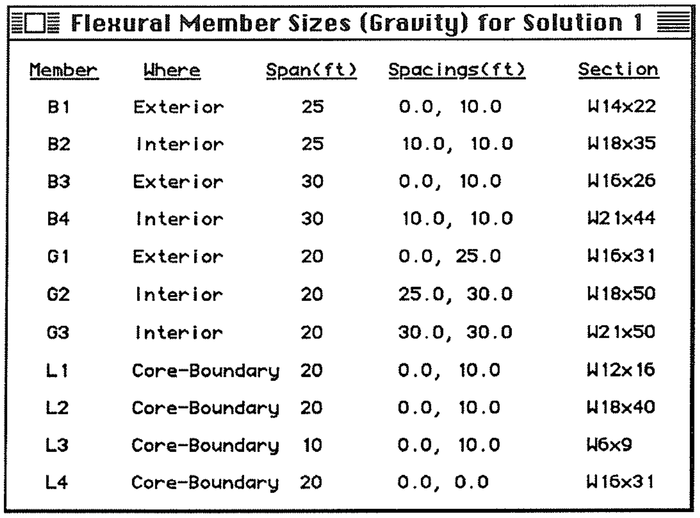


(a)



(b)

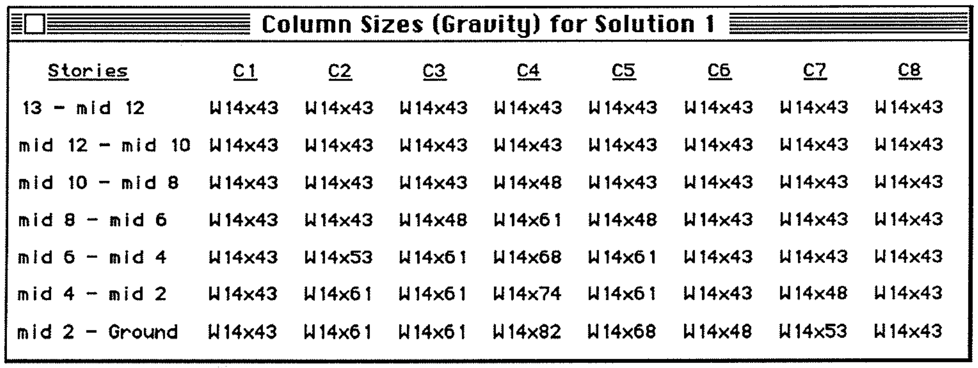
Figure 7.7: **Two Generated Framing Plans for Example II.** (a) Solution 1 (b) Solution 4.



Flexural Member Sizes (Gravity) for Solution 1

Member	Where	Span(ft)	Spacings(ft)	Section
B1	Exterior	25	0.0, 10.0	W14x22
B2	Interior	25	10.0, 10.0	W18x35
B3	Exterior	30	0.0, 10.0	W16x26
B4	Interior	30	10.0, 10.0	W21x44
G1	Exterior	20	0.0, 25.0	W16x31
G2	Interior	20	25.0, 30.0	W18x50
G3	Interior	20	30.0, 30.0	W21x50
L1	Core-Boundary	20	0.0, 10.0	W12x16
L2	Core-Boundary	20	0.0, 10.0	W18x40
L3	Core-Boundary	10	0.0, 10.0	W6x9
L4	Core-Boundary	20	0.0, 0.0	W16x31

(a)

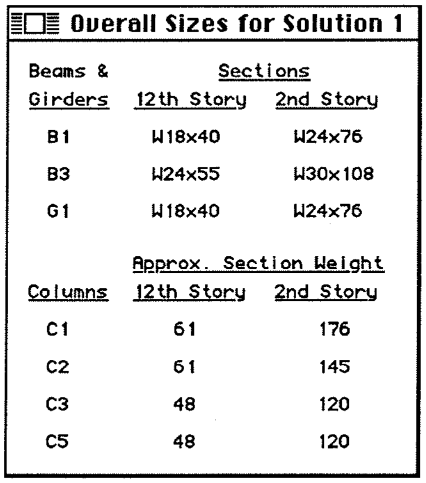


Column Sizes (Gravity) for Solution 1

Stories	C1	C2	C3	C4	C5	C6	C7	C8
13 - mid 12	W14x43	W14x43	W14x43	W14x43	W14x43	W14x43	W14x43	W14x43
mid 12 - mid 10	W14x43	W14x43	W14x43	W14x43	W14x43	W14x43	W14x43	W14x43
mid 10 - mid 8	W14x43	W14x43	W14x43	W14x48	W14x43	W14x43	W14x43	W14x43
mid 8 - mid 6	W14x43	W14x43	W14x48	W14x61	W14x48	W14x43	W14x43	W14x43
mid 6 - mid 4	W14x43	W14x53	W14x61	W14x68	W14x61	W14x43	W14x43	W14x43
mid 4 - mid 2	W14x43	W14x61	W14x61	W14x74	W14x61	W14x43	W14x48	W14x43
mid 2 - Ground	W14x43	W14x61	W14x61	W14x82	W14x68	W14x48	W14x53	W14x43

(b)

Figure 7.8: **Gravity System Sizes for Solution 1.** (a) Linear horizontal member sizes. (b) Column sizes.

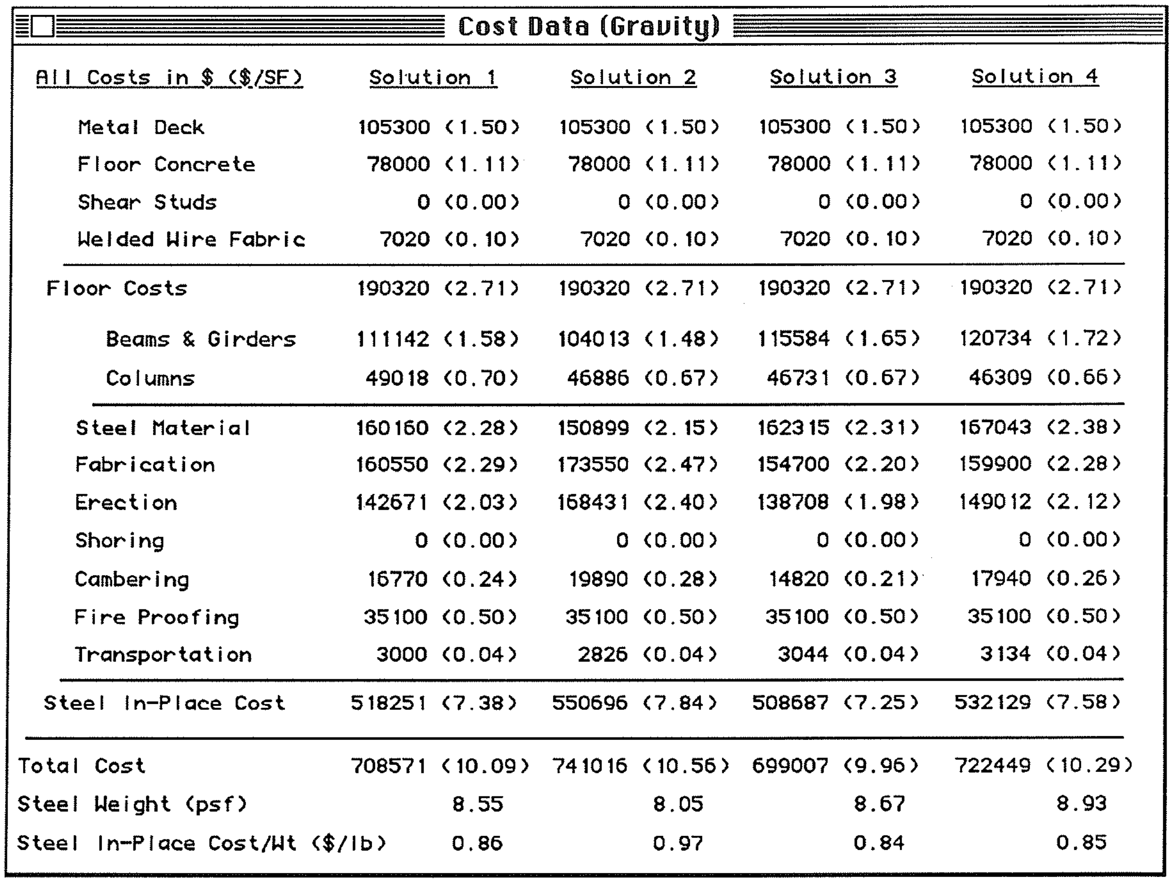


Overall Sizes for Solution 1

Beams &	Sections	
Girders	12th Story	2nd Story
B1	W18x40	W24x76
B3	W24x55	W30x108
G1	W18x40	W24x76

	Approx. Section Weight	
Columns	12th Story	2nd Story
C1	61	176
C2	61	145
C3	48	120
C5	48	120

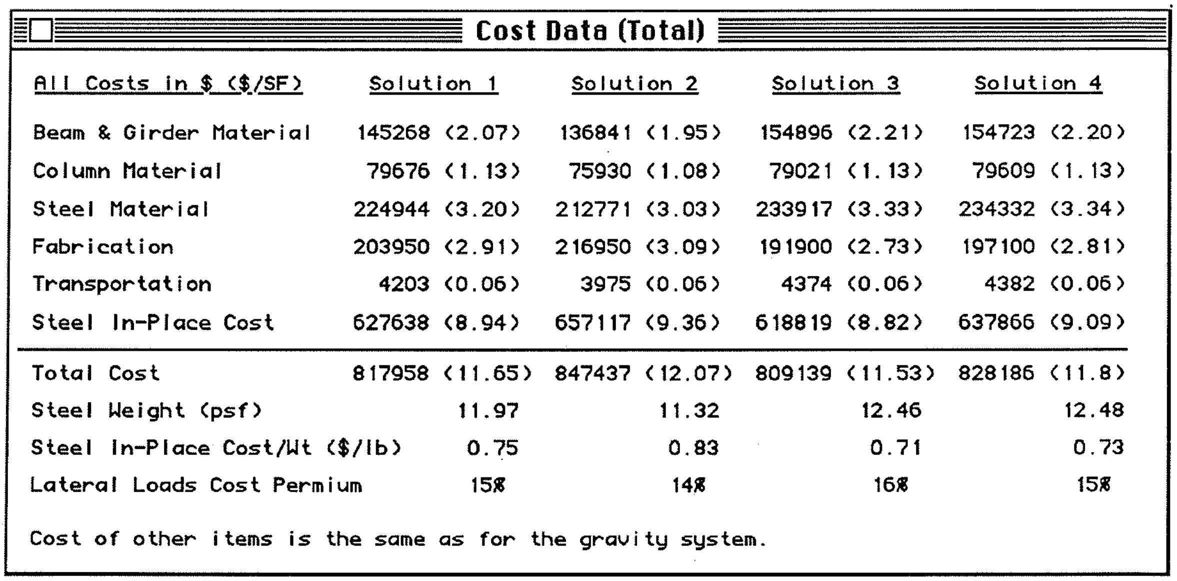
Figure 7.9: **Total System Sizes for Solution 1.** The steel weights of columns are only an estimate.



Cost Data (Gravity)

All Costs in $ ($/SF)	Solution 1	Solution 2	Solution 3	Solution 4
Metal Deck	105300 (1.50)	105300 (1.50)	105300 (1.50)	105300 (1.50)
Floor Concrete	78000 (1.11)	78000 (1.11)	78000 (1.11)	78000 (1.11)
Shear Studs	0 (0.00)	0 (0.00)	0 (0.00)	0 (0.00)
Welded Wire Fabric	7020 (0.10)	7020 (0.10)	7020 (0.10)	7020 (0.10)
Floor Costs	190320 (2.71)	190320 (2.71)	190320 (2.71)	190320 (2.71)
Beams & Girders	111142 (1.58)	104013 (1.48)	115584 (1.65)	120734 (1.72)
Columns	49018 (0.70)	46886 (0.67)	46731 (0.67)	46309 (0.66)
Steel Material	160160 (2.28)	150899 (2.15)	162315 (2.31)	167043 (2.38)
Fabrication	160550 (2.29)	173550 (2.47)	154700 (2.20)	159900 (2.28)
Erection	142671 (2.03)	168431 (2.40)	138708 (1.98)	149012 (2.12)
Shoring	0 (0.00)	0 (0.00)	0 (0.00)	0 (0.00)
Cambering	16770 (0.24)	19890 (0.28)	14820 (0.21)	17940 (0.26)
Fire Proofing	35100 (0.50)	35100 (0.50)	35100 (0.50)	35100 (0.50)
Transportation	3000 (0.04)	2826 (0.04)	3044 (0.04)	3134 (0.04)
Steel In-Place Cost	518251 (7.38)	550696 (7.84)	508687 (7.25)	532129 (7.58)
Total Cost	708571 (10.09)	741016 (10.56)	699007 (9.96)	722449 (10.29)
Steel Weight (psf)	8.55	8.05	8.67	8.93
Steel In-Place Cost/Wt ($/lb)	0.86	0.97	0.84	0.85

(a)



Cost Data (Total)

All Costs in $ ($/SF)	Solution 1	Solution 2	Solution 3	Solution 4
Beam & Girder Material	145268 (2.07)	136841 (1.95)	154896 (2.21)	154723 (2.20)
Column Material	79676 (1.13)	75930 (1.08)	79021 (1.13)	79609 (1.13)
Steel Material	224944 (3.20)	212771 (3.03)	233917 (3.33)	234332 (3.34)
Fabrication	203950 (2.91)	216950 (3.09)	191900 (2.73)	197100 (2.81)
Transportation	4203 (0.06)	3975 (0.06)	4374 (0.06)	4382 (0.06)
Steel In-Place Cost	627638 (8.94)	657117 (9.36)	618819 (8.82)	637866 (9.09)
Total Cost	817958 (11.65)	847437 (12.07)	809139 (11.53)	828186 (11.8)
Steel Weight (psf)	11.97	11.32	12.46	12.48
Steel In-Place Cost/Wt ($/lb)	0.75	0.83	0.71	0.73
Lateral Loads Cost Permium	15%	14%	16%	15%

Cost of other items is the same as for the gravity system.

(b)

Figure 7.10: **Costs for Example II Alternatives.** (a) Gravity system costs. (b) Total system costs.

Comparing the gravity and total cost data given in Fig. 7.10 provides some interesting insight into the cost impact of the lateral system. The total structural steel weight increased by about 40% and the fabrication cost increased by about 25%, leading to a total cost increase of about 15%. These increases have to be interpreted with due consideration of the assumptions made in the analysis. Two important issues need to be considered. Firstly, in the cost estimation process (see Section C.4) it is assumed that the cost difference between equally heavy "gravity" support elements (assumed to have only simple connections) and "frame" elements (assumed to have rigid connections) is reflected only in the fabrication cost; the erection cost is not affected. This heuristic knowledge is based on input provided by one cost estimation expert and is expected to vary from expert to expert. Secondly, the weight penalty for the lateral system is a very conservative estimate since, as was discussed in Chapter 5, all column sizes are limited to W14 sections, which are inefficient sections for columns in perimeter frames. As a consequence of these assumptions, the cost per pound of steel for the gravity and lateral system is less than that of the gravity system alone.

8 Observations on Logic

Because of its context-free and task-independent nature, declarative representation is a convenient means of representing knowledge, especially so because it avoids major revisions that must often be undertaken to accommodate even minor changes in knowledge in the case of procedural representation. Knowledge represented declaratively can be used even in situations that were not foreseen by the developer. This chapter summarizes some observations based on our experience with using formal logic for declaratively formulating knowledge in the structural engineering domain. The next section comments about the expressive power and inferential capabilities of first-order predicate calculus, a variant of which was employed in this investigation. Observations about knowledge transparency follow in Section 8.2. Section 8.3 discusses a few observations on KIF whereas Section 8.4 discusses efficiency in a logical environment. Shortcomings of logic and the proposed solutions to overcome them are discussed in the penultimate section. The final section concludes with an extended quotation of remarks made by developers of a large knowledge base.

8.1 Expressive Power and Inferential Capabilities

First-order predicate calculus (FOPC) is one of the most expressive and versatile languages available for representing knowledge declaratively. Alternative forms of declarative representation, such as frames, production systems, and semantic nets, can be defined in terms of FOPC. Charniak and McDermott [11] illustrated translation of semantic nets into FOPC while Hayes [36] has shown that frames can be defined using FOPC.

It is the expressiveness of first-order predicate calculus that gives it a significant advantage over other languages. For instance, if we were to express the fact that "IF a building has a moment resisting frame AND the lateral deflections are large THEN beams are too flexible OR columns are too flexible OR columns have too small an area" in a production system environment, the representation will be quite clumsy because of the OR's in the consequent part of the statement. On the other hand, such statements can be easily expressed in first-order logic using disjunctions. Also, as noted in Ref. [26], incomplete and negative information can be much more readily expressed in predicate logic when compared with other formalisms. An illustration of this can be observed when trying to state that a load has an eccentricity, without specifying what the eccentricity is (though it may become available later).

Perhaps the single most important source of the expressive power of FOPC is quantifiers. Quantifiers allow one to state properties of all the objects in a domain or of some objects in the domain without explicitly naming the objects. To give an idea of their power, consider a case where there are several beams of different depths. If we wish to find out the deepest beam(s) in a frame-based environment, it will not be possible to do so without explicitly naming all the beams (which will be quite a nuisance whenever there are additions and deletions to the list of beams). In FOPC, however, we just need to write a simple axiom like the following.

$\forall b\ [\texttt{Beam(b)} \land \neg\exists \texttt{b1}\ \texttt{Deeper(b1,b)} \Rightarrow \texttt{Deepest(b)}]$

When used in conjunction with unification, this axiom can be easily used to determine the deepest beam or to find out if a given beam is the deepest.

Another benefit of symbolic logic is the reasoning power it offers. Ordinary production systems, for example, perform only *modus ponens* (i.e., given evidence of the antecedent of a rule, they can infer the consequent). Besides *modus ponens*, a true logic system can perform several other kinds of sound reasoning. To give a simple example, suppose one knows that

$\texttt{Supports(column,beam)} \Rightarrow \texttt{SupportedBy(beam,column)}$

and that Beam1 is not supported by Column1 (or, in other words, $\neg$SupportedBy(Beam1, Column1)). From this knowledge one can still not infer $\neg$Supports(Column1,Beam1) in a production system environment. However, one can do so in a logic system with application of either *modus tollens* or resolution. We found logic to be inferentially adequate for our purpose of modeling conceptual structural design of steel office buildings.

8.2 Knowledge Transparency

A couple of example applications in Appendix A illustrate the utility of FOPC for solving engineering problems. As the application to structural analysis in the appendix shows, knowledge for solving the problem, which is hidden in the case of algorithmic solution, becomes transparent and natural in predicate calculus axioms. Similarly, processing design standards with predicate calculus is advantageous because, in contrast with other formalisms, the same representation can be used for both conformance checking (using demodulation) as well as design generation (using paramodulation).

We can illustrate transparency of the represented knowledge in FOPC using another simple example. Consider the case of computing the factorial of a nonnegative integer n in procedural and declarative paradigms. Fig. 8.1 shows the mathematical definition of the 'factorial' function and its computation using Fortran, FOPC, Prolog, and KIF respectively. As is evident from the figure, the knowledge is hidden in the case of Fortran since it captures the *how* of the problem. The predicate calculus and KIF representations, on the other hand, are very clean and elegant since they emphasize the *what*, leaving the *how* of the problem-solving process to be decided by the interpreter. It is this elegance and cleanliness of expression that is so dear to the proponents of logic.

The transparency of knowledge results in easier development as well as reduced effort for maintenance, modification, and extension of the system. Debugging a procedural program is a nontrivial chore and requires keeping track of how and when each piece of knowledge is being used; in contrast, only the truth of statements matters in logic. Also, one does not have to worry about such low-level details as indexing or memory allocation. (As an aside, a procedural representation equivalent to a declarative one will always be bigger because of the space needed for the *how* part.) When adding new knowledge to procedural programs, the impact and interaction of such knowledge with existing knowledge has to be considered; such concerns are minimal in the case of declaratively represented facts that have clean semantics. After implementing any well-sized system in a procedural language, the programmer often experiences a strong urge to redo the system differently; such is not a characteristic of logic.

8.3 Observations on KIF

Because of its philosophy of no short-cuts or compromises, KIF, an extension of FOPC and the language employed in this investigation, encourages much necessary discipline and rigor on

$$Factorial(n) = \begin{cases} 1, & \text{if } n = 0; \\ n * Factorial(n-1), & \text{if } n > 0. \end{cases}$$

(a)

```
      SUBROUTINE FACTORIAL(N, IFACT)
      IFACT = 1
      DO 10 J = 1, N
      IFACT = IFACT * J
10    CONTINUE
      RETURN
      END
```

(b)

$\texttt{Factorial(0)} = \texttt{1}$
$(\texttt{n} > \texttt{0}) \Rightarrow \texttt{Factorial(n)} = \texttt{n} * \texttt{Factorial(n} - \texttt{1)}$

(c)

```
factorial(0, 1).
factorial(N, F) :- N>0, N1 is N-1, factorial(N1, F1), F is N*F1.
```

(d)

```
(= (factorial 0) 1)
(<= (= (factorial $n) (* $n (factorial (- $n 1))))
    (> $n 0))
```

(e)

Figure 8.1: **Computing Factorial of a Nonnegative Integer n.** (a) Mathematical Definition, (b) Fortran Representation, (c) FOPC Axiomatization, (d) Prolog Representation, and (e) KIF Representation. Note the correspondence between (c) and (e).

---

the part of the programmer. It may take a little extra effort at the development stage to be quite precise in expression, but the effort is worthwhile from the long-term perspective. This is because every time someone twitches the code just to get it running (without adherence to rigorous declarative semantics), the person is creating a problem for the future. As an illustration, assume that one is writing a small expert system for some narrow task. In the small knowledge base (KB) the programmer may put in a "kludge" so that a chosen rule always fires immediately after the firing of another rule. Such an unprincipled approach may work for the small knowledge base, but some later day one may need to integrate that small KB with another KB developed for some related task. Now the programming trick that was working earlier may not work anymore because of the presence of additional rules that may be candidates for firing. However, a fact ought to be true irrespective of whether it is part of a small KB or a large one. But in our scenario the fact which was functioning well in the small system may produce unexpected behavior once it is put in a different context—hence the emphasis on semantics independent of the interpreter, though the process may seem cumbersome initially. (Section 8.6 quotes from a large-scale KB development effort that corroborates this observation.) One expresses a fact in a knowledge base and it will always be true irrespective of how it is used.

One advantage of expressing domain knowledge in KIF is that one can write alternative strategies (or meta-level knowledge) that operate on the same set of base-level statements. Thus, while selecting sections for columns in Galileo, one can opt for either a *least-weight strategy* to minimize the steel costs, or a *constant-depth strategy* to simplify splicing. There is only one set of base-level statements expressing properties of hot-rolled steel sections, but it can be used in multiple ways by separating the meta-level knowledge and providing for alternative strategies. On the other hand, the idea of mixing base-level and meta-level knowledge, or embedding the latter in the former (quite common in Prolog programs as enabled by constructs like `cut` and `fail`) is anathema to purists.

Mathematical equations, quite commonly used in structural engineering for expressing behavior and performance of components and systems, can be very conveniently expressed in KIF using *term simplification*, or *rewrite*, rules. The general form of these rules is the following.

$$\texttt{(<-}\ \tau_1\ \tau_2\texttt{)}$$

This statement signifies that an inference procedure is free to substitute the term τ_2 for all occurences of the term τ_1. For instance, bending moment at the center of a beam of span l carrying a uniformly distributed load w is expressed by the equation

$$M = \frac{wl^2}{8}.$$

The same can be precisely expressed in KIF as follows.

```
(<- (M udl center $w $l) (/ (* $w $l $l) 8))
```

8.4 Efficiency

There is always a trade-off between the generality of representation and efficiency [86]. Thus procedural representations, where the programmer encodes the knowledge about what is to be done each step, are usually quite efficient, though the represented knowledge is specific to a particular task. In contrast, the general purpose inference mechanisms in the case of formal logic must either perform a blind search or deliberate at each step as to what to do next; they are consequently slower [15].

The issue of efficiency must be seen in the proper perspective: if efficiency was to be the most important consideration, most of us would be programming in machine language. In other words, one has to consider how natural is the expression in a language and how clean is its semantics, in addition to efficiency. This is where FOPC's clear semantics combined with its expressive power becomes important. Moreover, given the current rate of improvement in hardware, excessive concerns about long run-times may not be entirely justifiable.

8.5 Shortcomings and Solutions

McDermott [64] raised questions about the suitability of logic for problem solving and pointed to the inadequacies of logical deduction for modeling human problem solving. He justifiably argued that humans do many other kinds of reasoning besides deduction (such as abduction, induction, etc.). To that end it must be pointed out that though deduction is an important form of reasoning, it need not be the only one. Some recent work in the field of nonmonotonic logic, such as circumscription [63,55], default logic [75], and modal operators [65,66], extends the reasoning power of classical logic. Nonmonotonic reasoning, which provides means for making unsound inferences (that may later be retracted), has interesting parallels in the structural design process. For instance, when designing high-rise buildings, designers typically assume that the lateral loads govern the design of vertical systems. Later on, they may discover that some individual components are governed by gravity loads, and this will result in a revision of design of affected components. This is an illustration of *default reasoning*, wherein the system makes certain inferences that are *normally* true but at a later stage retracts those inferences when additional information becomes available.

In addition to nonmonotonic logic, other nonclassical logics are actively being developed to overcome the shortcomings of classical logic. They include temporal logic (to reason about time), modal logic (to reason about, for instance, knowledge and belief), many-valued logics (to compensate for the inability of classical logic to account for anything other than binary truth values), and fuzzy logic (to handle uncertain information).

Stefik [83] points out that there are computational problems with the use of logical deduction as the core of problem solving. For instance, even if logic does provide a means of finding the result, it does not provide guidance for going about finding the result. Investigations into meta-level constraints and reflection [26] are intended to address such concerns.

One of the shortcomings of predicate calculus is that it is too cumbersome for simple, specific tasks. A specialized language may be more succint and natural than predicate calculus for many applications. Predicate calculus expressions also appear to be awkward to the uninitiated and it is only after some time and practice that one gains familiarity and begins to feel comfortable with the language. A solution to this problem of readability will be an automated front-end that can converse with the user in friendlier representations, while using predicate calculus as a representation base. Mackinlay [59] worked in this direction by building a program capable of displaying predicate calculus data in different modalities, such as tables, trees, bar graphs, and pie charts.

A problem with the inference mechanisms in logic is their indeterminacy. The most powerful of logical inference mechanisms, resolution, is only refutation complete; i.e., given a set of inconsistent facts, resolution is guarenteed to deduce the empty clause from it. However, if the set is consistent, there is no way of ascertaining it since inability of resolution to deduce empty clause in a given amount of time does not necessarily imply consistency. Perhaps the given set of sentences is inconsistent and a few additional inference steps would have proved it. A proposed (though not entirely satisfactory) solution to the problem is to limit the time allowed for deducing the desired clause. If a clause is not deduced within the specified time, it will be taken to mean that the clause does not follow from the original set.

In several cases, one has to determine all the objects in the domain that satisfy a particular condition. For instance, it may be desired to determine all the loads acting on a given beam. In yet other instances, the *number* of objects that meet some criterion may be needed. For example, when computing the material cost, one may need to determine the total number of beams of a particular type in a floor. Logic programming languages like Prolog address these problems by providing second-order predicates such as `setof` and `bagof`. (Epikit also extends KIF and provides functions such as `bagof`.) However, it must be borne in mind that such constructs are outside the realm of first-order logic and carry procedural semantics.

There are some tasks in the structural design process which are inherently iterative. An example is the design of beam-columns, wherein one starts by choosing a section based on some "equivalent" axial load and reiterates until the interaction equation is satisfied, while ensuring that the section is not too overdesigned. Modeling such procedures appears unnatural in logic. A process where the steps are repeated until some variable converges to within, say, 5% of its value in the previous iteration will be another instance. Moment distribution method is an example of such an iterative task.

8.6 Concluding Remarks

In this section we present the following extended quotation from a recent article [32] about Cyc, a decade-long, two person-century effort to develop a large common sense knowledge base intended to capture human consensus knowledge. Mid-way in the project, the authors report on their experiences and lessons learned over the past five years. Much of what they say about the representation language corroborates the observations in this chapter.

> CycL is the language in which Cyc knowledge base is encoded. In 1984, our representation was little more than frames. Although a significant fraction of knowledge can be conveniently handled using just frames, this approach soon proved awkward or downright inadequate for expressing various assertions we wanted to make: disjunctions, inequalities, existentially quantified statements, metalevel propositions about sentences, and so on. At least occasionally, therefore, we required a framework of greater expressive power. We were thus led to embed the frame system in a predicate calculus-like constraint language.
>
> Moreover, there were a number of downright ad hoc aspects of the 1984 frame language, such as how inheritance worked. We kept modifying and tweaking such mechanisms, and often, this method forced us to go back and redo parts of the knowledge base so that they corresponded to the new way the inference engine worked. As the size of the knowledge base increased, this process became intolerable. We came to realize that having a clean semantics for the knowledge base was vital, declaratively expressing the meaning of inheritance, TheSetOf, default rules, automatic classification, and so on, so that we wouldn't have to change the knowledge base when we altered the implementation of one of the mechanisms.
>
> ...
>
> We conclude this section by listing our desiderata for the representation language in which to build our large knowledge base. ... Our point here is that this list bears little resemblance to what we expected the language to be like five years ago.

First, the language should have a clear (and hopefully simple) semantics. The semantics should be declarative for two reasons: to facilitate communication with, and use by, many different problem solvers, be they human or machine and, as mentioned previously, to prevent having to discard or redo the knowledge base when the inference mechanisms change.

Second, it should provide certain inferential capabilities. ...

Third, it should provide some scheme for expressing and reasoning with default knowledge. ...

Fourth, it should have the expressiveness of all first-order predicate calculus with equality [67]. ...

Given this wish list, we could summarize the major changes in CycL over the past five years as follows: We started with a frame language whose emphasis was more on issues such as the data structures used, indexing, and interface. We have since realized the importance of a declarative semantics for the language, the need for expressive power, and the importance of making a clear distinction between what knowledge the knowledge base contains and how the knowledge base is implemented.

Logic, as it is generally understood, is the organ with which we philosophize. But just as it may be possible for a craftsman to excel in making organs and yet not know how to play them, so one might be a great logician and still be inexpert in making use of logic. Thus we have many people who theoretically understand the whole art of poetry and yet are inept at composing mere quatrains; others enjoy all the precepts of da Vinci and yet do not know how to paint a stool. Playing the organ is taught not by those who make organs, but by those who know how to play them; poetry is learned by continual reading of the poets; painting is acquired by continual painting and designing; the art of proof, by the reading of books filled with demonstrations—and these are exclusively mathematical ones, not logical ones.

— *Galileo Galelei,* Dialogue Concerning the Two Chief World Systems *(translated by Stillman Drake) (1632)*

9 Closure

9.1 Conclusions

Several conclusions can be drawn from the study presented in this dissertation. As the investigation illustrates, reasoning based on function, with an emphasis on first principles (in place of heuristics), can be successfully applied for developing systems for conceptual structural design. Although heuristics are not absolutely dispensable in all aspects of conceptual design phase, minimizing their use even at the cost of speed can be worthwhile because of added generality in the represented knowledge.

A first-order predicate calculus based representation was found to be suitable for representation and reasoning in an integrated structural design environment. Especially noteworthy features of the representation include its expressive power, inferential capabilities, and cleanliness and elegance of expression resulting in knowledge transparency. Although there are concerns about issues like efficiency and syntax, the benefits far outweigh the drawbacks. Ongoing research in AI will hopefully address the concerns and remedy the shortcomings.

A sequential approach to the design of subsystems, in which a gravity load resisting system is synthesized independently of the lateral load resisting system, was adopted in this research. This helps one visualize the premium being paid (in terms of extra cost) for carrying the additional lateral loads incident on the structure.

This investigation also demonstrates that exogenous constraints can be advantageously incorporated in the structural design process. Realistic and practical designs must consider several other factors besides structural efficiency, and to that end, computer aids for structural design must also be able to handle exogenous constraints.

Effects of changing different parameters can best be measured in terms of the trade-off between *Cost* and *Value*. Because of the inherently subjective nature of *Value*, an automated environment is not well-suited for its computation. However, by estimating the *Cost* part of the *Cost/Value* ratio, and leaving the *Value* judgment to the user, a useful basis for evaluating the generated alternatives can be presented to the user.

9.2 Directions for Future Work

Although advancements such as more efficient search strategies or powerful representation schemes resulting from AI research will facilitate further developments, it is clear that even with existing techniques significant results can be achieved with knowledge-based conceptual structural design systems. Thus, as structural engineers, our energies will be spent more constructively in formalizing those parts of knowledge for which no widely acceptable theories exist. These include the problem-solving knowledge and the strategies used by expert designers. In this connection, application of general design methodologies to the conceptual structural design process should be explored.

Efforts should be devoted to the development of libraries or repositories of that body of structural engineering knowledge which is fairly stable. For this purpose, we will have to agree upon an ontology, i.e., evolve a vocabulary in which the definition of representational terms is standardized [31]. By encoding the knowledge in a declarative language (e.g., one whose syntax is based on first-order predicate calculus), such knowledge resources can be shared and reused, thus avoiding duplication of effort.

The problem of structural system elements layout is common to all civil engineering structures. This dissertation presented a partial formalization for solving such problems in the domain of building structures. Applications and extensions of the approach presented here to other types of structures can be explored.

9.2.1 Enhancements to Galileo

Several enhancements can be made to the system developed in this investigation. Allowing for non-rectangular geometries, non-prismatic buildings, and more than a single core are some of the obvious possible avenues. In addition, occupancies other than commercial office buildings and materials other than steel can also be considered.

The reasoning in Galileo can be made more intelligent by giving preference to symmetric framing plans, by adding capability for generating plans that have nonorthogonal flexural members, and by striving for uniformity in sections of the chosen members when doing sizing. A more sophisticated system will also be able to generate framing plans in which the orientation of floor beams (or girders) is not identical throughout the plan, but may vary in different zones. Some other useful extensions include the ability to handle varying load intensities in different parts of the floor plan and a more detailed treatment of the MEP ductwork. For the latter, the actual route of the ducts and the particular members affected by them can be used to gauge the influence of MEP constraints on the floor framing plan.

Since the unit erection costs for lower stories are usually smaller than those for higher stories (erecting higher stories involves more time and effort), ideally one may use different framing plans for higher and lower stories. The top stories may have fewer elements so as to cut down on erection costs, while the bottom stories can use more elements and save on material costs. Also, owing to the requirements of equipment for heating, ventilating, and air-conditioning, mechanical floors may require framing plans that are different from typical floors. The same may be true for the first floor since many columns may be discontinued to obtain a big lobby area free of columns. Galileo can be extended to include all these special-cases.

At the moment, the service core in the floor plan is treated as a weightless area during the design of the gravity system. An extended system should be able to consider each individual area within the core and design the structural system accordingly. Furthermore, the design of lateral systems can be extended significantly, including such systems as braced frames and tubes.

There are some possible extensions to the interface also. An interface with a computer-aided drafting (CAD) system as a front-end to Galileo will facilitate graphic communication of the input [42]. A customization module, implemented in the CAD system, will allow the user to modify and customize generated alternatives. For improving efficiency, application of automated transformation and compilation techniques can be explored to develop specialized inference procedures. Finally, Galileo can be made into a closed-loop system wherein results of the lateral system design are taken into account to modify the gravity system alternatives and the process continues until the two are reconciled.

9.3 Summary Assessment

A formal model of both the domain as well as the problem-solving process is an important ingredient for developing extensible and intelligent design systems. Reasoning in terms of the cause for the existence of a structure is a step towards that direction. An explicit representation of the function of structural elements and systems is essential for such reasoning. Also, if the model of the domain is based largely on first principles knowledge, the knowledge base will rest on a sound foundation and will be more general and applicable to many different types of structures. This work extends and applies a formal methodology consistent with such goals to the domain of structural design of multistory commercial steel buildings. A "proof-of-concept" implementation of a conceptual design system demonstrates the utility of the model.

By exploring the use of a FOPC-based representation for developing a medium-sized structural design application, strengths and limitations of FOPC as a representation language have become apparent. Despite a few minor concerns, predicate logic appears to be a suitable medium for representation and reasoning in conceptual structural design systems.

This work has also resulted in formalizing part of the problem-solving knowledge required for floor framing generation of steel office buildings. Although structural behavior is largely well-understood and codified in the theory of structures, the issue of formalization of problem-solving (or strategic) knowledge still needs attention. Partly, this involves formalizing the interaction between structural constraints and exogenous constraints. As was discussed earlier, recognizing the importance of exogenous considerations is very important for developing practical and useful designs. As the implementation in this work shows, such considerations can be successfully incorporated in the reasoning process at the conceptual phase of structural design.

It is important to provide flexibility to the user to control the actions of the design system. The goal of flexibility has been realized in several different ways in Galileo. To give some specific examples, the user can (i) change the default values of design parameters and prices, (ii) force the system to choose a particular type of concrete or a particular strength of steel, (iii) select options related to construction, such as shoring and cambering, and (iv) specify constraints on the products to be used, such as constraints on the depths of beams and type of shear studs.

Whenever the system has to make a decision amongst alternatives, the selection is based on a comparison between the respective economies of various alternatives, instead of heuristics like "If the speed of construction is crucial then eliminate concrete from consideration." We believe that estimation of cost is the most useful indicator for discriminating between alternatives at a localized scale (for instance, while choosing between lightweight and normalweight concrete when developing a solution) as well as global scale (for instance, while comparing fully developed structural schemes).

If you cannot understand my argument, and declare "It's Greek to me," you are quoting Shakespeare; . . . if you have ever refused to budge an inch or suffered from green-eyed jealousy, if you have played fast and loose, if you have been tongue-tied, a tower of strength, hoodwinked or in a pickle, if you have knitted your brows, made a virtue of necessity, insisted on fair play, slept not one wink, stood on ceremony, danced attendance (on your lord and master), laughed yourself into stitches, had short shrift, cold comfort or too much of a good thing, if you have seen better days or lived in a fool's paradise—why, be that as it may, the more fool you, for it is a foregone conclusion that you are (as good luck would have it) quoting Shakespeare; . . . even if you bid me good riddance and send me packing, if you wish I was dead as a door-nail, if you think I am an eyesore, a laughing stock, the devil incarnate, a stony-hearted villian, bloody-minded or a blinking idiot, then—by Jove! O Lord! Tut, tut! for goodness' sake! what the dickens! but me no buts—it is all one to me, for you are quoting Shakespeare.

— Bernard Levin

Appendices

A

A Propositional and Predicate Calculus Primer

In this appendix we discuss two logical languages, namely, propositional calculus and predicate calculus, that can be used to represent knowledge in a given domain in a declarative fashion. Their use is illustrated through applications to structural engineering problems. Our emphasis in this appendix is less on mathematical rigor than on informally conveying the essence of how logic can be usefully employed for creating knowledge systems in engineering domains. Thus we do not strive for completeness or mathematical precision; however, references to appropriate material are provided throughout this appendix for the inquisitive reader.

A representation language has two components: syntax and semantics. The *syntax* establishes the rules of grammar that enable one to formulate legal sentences in the language. For example, the sentence "updated Paris times word london below problem" is syntactically unacceptable in the English language since it violates the rules of English grammar. The *semantics* of the language, on the other hand, defines the meanings of the expressions in the language. That is how, for example, we humans understand the essence of spoken words. Thus, the symbol `Paris` is normally taken to denote the city that is the capital of France. In a manner analogous to the English language, both propositional calculus and predicate calculus have well-defined syntactic and semantic rules.

This appendix is divided into five sections. First, we discuss the syntactical (pertaining to the organization of symbols) and semantical (pertaining to the meaning of the symbols) aspects of propositional and predicate calculus in detail. The following section illustrates a prototypical application of logic for structural analysis. Another application, this time to standards processing for detailed design, is the subject of the penultimate section. The final section presents some concluding remarks and suggestions for further reading.

A.1 Propositional Calculus

A.1.1 Syntax

Propositional calculus is a language of abstract statements that consist of propositions and connectives. *Propositions* can either be *truth symbols*, `True` and `False`, or *propositional symbols*, such as `P`, `ConstructionMaterialIsSteel`, `BldgIsInSeismicZone`, and `R2D2`. In the notational convention adopted here, propositional symbols begin with an uppercase letter and can consist of alphanumeric characters. The *connectives* can be *and* ($\wedge$), *or* ($\vee$), *not* ($\neg$), *implies* ($\Rightarrow$), or *equivalent* ($\equiv$).[1] Each proposition in itself is a sentence; thus `P` and `True` are both legal sentences in propositional calculus. More complex sentences can be formed using connectives. The types of sentences that can be formed are:

$\neg\alpha$
$\alpha \wedge \beta$
$\alpha \vee \beta$
$\alpha \Rightarrow \beta$
$\alpha \equiv \beta$

where α and β can themselves be simple or complex sentences. Parentheses, ( ), and brackets, [ ], are used to clarify the structure of a sentence when there is a possibility of ambiguity. Using these rules, it can be seen that the following is a syntactically correct sentence.

$[(\neg \mathtt{P} \wedge \mathtt{Q}) \Rightarrow (\mathtt{Q} \vee \mathtt{False})] \equiv \mathtt{True}$

A.1.2 Semantics

Having discussed the syntax of propositional calculus, we move on to study the semantical issues. Using semantical rules one can determine the truth value of a sentence—that is, a value of either *true* or *false* can be assigned to a sentence based on the truth of its propositions and the connections among the propositions. The following rules are used for this purpose.

- The sentence `True` is always *true*.

[1]Some authors (for instance, [70]) prefer alternative symbols for connectives, e.g., & instead of $\wedge$; $\sim$ or $-$ instead of $\neg$; $\rightarrow$ or $\supset$ instead of $\Rightarrow$; and $\leftrightarrow$ or $\Leftrightarrow$ instead of $\equiv$.

- The sentence `False` is always *false*. (Note the distinction between the truth symbols `True` and `False` and the truth values *true* and *false*.)
- A sentence consisting of a single proposition and no connectives is *true* if the proposition is *true*, and *false* if the proposition is *false*. Thus the sentence P will be *true* if and only if the proposition P itself is *true*.
- The sentence $\neg\alpha$ is *true* if α is *false*, and *false* if α is *true*.
- The sentence $\alpha \wedge \beta$ is *true* if both α and β are *true*, and *false* otherwise.
- The sentence $\alpha \vee \beta$ is *false* if both α and β are *false*, and *true* otherwise.
- The sentence $\alpha \Rightarrow \beta$ is *false* if α is *true* and β is *false*, and *true* otherwise.
- The sentence $\alpha \equiv \beta$ is *true* if both α and β have the same truth value (i.e., they are either both *true* or both *false*), and *false* otherwise.

Thus, for the case when P is *true* and Q is *false*, the sentence

$(\mathtt{P} \wedge \neg \mathtt{Q}) \Rightarrow \mathtt{False}$

will evaluate to *false* since according to the rules:

Q will evaluate to *false*,
$\neg$*false* will evaluate to *true*,
P will evaluate to *true*,
true $\wedge$ *true* will evaluate to *true*,
`False` will evaluate to *false*, and
true $\Rightarrow$*false* will evaluate to *false*.

Once knowledge is expressed as a set of sentences in propositional calculus, new sentences can be deduced using *rules of inference*. Some example rules of inference are given in Fig. A.1. One of these rules is *modus ponens*, which, from two sentences of the form $\alpha \Rightarrow \beta$ and α, permits us to deduce β. This is a commonly employed rule in our daily reasoning processes.

Besides the rules of inference, one can also make use of the *laws of propositional algebra*. These laws are of the form $\alpha \equiv \beta$, meaning that the expressions on both sides of the '$\equiv$' symbol are equivalent. Thus one expression can be freely substituted in place of the other and new sentences can be formulated. A partial list of such laws is given in Fig. A.2. (A fuller list can be found in Ref. [4].)

1. *Modus Ponens*: From two sentences of the form

$$\alpha \Rightarrow \beta$$

and

$$\alpha$$

one can infer the sentence

$$\beta$$

2. *Modus Tollens*: From two sentences of the form

$$\alpha \Rightarrow \beta$$

and

$$\neg\beta$$

one can infer the sentence

$$\neg\alpha$$

3. *And Introduction*: From two sentences of the form

$$\alpha$$

and

$$\beta$$

one can infer the sentence

$$\alpha \wedge \beta$$

4. *Modus Tollens*: From a sentence of the form

$$\alpha \wedge \beta$$

one can infer the two sentences

$$\alpha$$

and

$$\beta$$

Figure A.1: **Sample Rules of Inference.** New sentences can be deduced from a set of sentences using rules of inference.

A.1.3 Example

The use of propositional calculus for reasoning is illustrated in this section with a simple, though contrived, engineering example. We start with the following two axioms.

- If the building has a moment resisting frame and the lateral deflection at the top under the lateral loads is too large, then increase the stiffness of columns or increase the stiffness of beams.

- If the flexural stiffness of columns is much larger than the flexural stiffness of beams, then do not increase the stiffness of columns.

Also, let us assume that for a given building the following is true.

- The building has a moment resisting frame.

- The lateral deflection at the top under the lateral loads is too large.

- The flexural stiffness of columns is much larger than the flexural stiffness of beams.

1. *Commutativity Laws*

$$\alpha \vee \beta \equiv \beta \vee \alpha$$

$$\alpha \wedge \beta \equiv \beta \wedge \alpha$$

2. *Distributivity Laws*

$$\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$$

$$\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$$

3. *Idempotence Laws*

$$\alpha \vee \alpha \equiv \alpha$$

$$\alpha \wedge \alpha \equiv \alpha$$

4. *Complementation Laws*

$$\alpha \vee \neg\alpha \equiv \texttt{True}$$

$$\alpha \wedge \neg\alpha \equiv \texttt{False}$$

5. *Annihilator Laws*

$$\texttt{True} \vee \alpha \equiv \texttt{True}$$

$$\texttt{False} \wedge \alpha \equiv \texttt{False}$$

6. *Identity Elements*

$$\alpha \vee \texttt{False} \equiv \alpha$$

$$\alpha \wedge \texttt{True} \equiv \alpha$$

7. *DeMorgan's Laws*

$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha) \vee (\neg\beta)$$

$$\neg(\alpha \vee \beta) \equiv (\neg\alpha) \wedge (\neg\beta)$$

8. *Cancellation of Negations*

$$\neg(\neg\alpha) \equiv \alpha$$

Figure A.2: **Sample Laws of Propositional Algebra.** By substituting expressions that are equivalent to other expressions, new sentences can be formulated from given sentences.[2]

From the preceding statements, the reader can derive that the recourse for reducing the lateral deflection in the given case is to increase the stiffness of the beams. We can reach the same conclusion through syntactic manipulations with propositional calculus. In the reasoning process we make use of the rules of inference and the laws of propositional algebra. The following symbols are used for the propositions in the proof.

M	The building has a moment resisting frame.
L	The lateral deflection at the top under the lateral loads is too large.
F	The flexural stiffness of columns is much larger than the flexural stiffness of beams.
B	Increase the stiffness of the beams.

[2]The distinction between the rules of inference and the laws of algebra of propositions is rather arbitrary. For instance, the cancellation of negations law can be formulated as the double negation rule which permits one to deduce α from a sentence of the form $\neg(\neg\alpha)$ and vice versa. Similar statements can be made about DeMorgan's and other laws.

C Increase the stiffness of the columns.

Using these symbols, the given statements can be expressed as follows.

1. $(M \land L) \Rightarrow (B \lor C)$
2. $F \Rightarrow \neg C$
3. M
4. L
5. F

From these statements, the following can be deduced with the aid of the rules of inference and the laws of propositional algebra (mentioned within parentheses for each statement).

6.	$M \land L$	(And Introduction: 3, 4)
7.	$B \lor C$	(Modus Ponens: 1, 6)
8.	$\neg C$	(Modus Ponens: 2, 5)
9.	$(B \lor C) \land (\neg C)$	(And Introduction: 7, 8)
10.	$(B \land \neg C) \lor (C \land \neg C)$	(Distributivity: 9)
11.	$(B \land \neg C) \lor \texttt{False}$	(Complementation: 10)
12.	$B \land \neg C$	(Identity: 11)
13.	B	(And Elimination: 12)

Thus, the desired conclusion (increase the stiffness of the beams) has been reached. The proof may seem cumbersome, but we have employed only the most primitive of algebraic manipulations. Higher level manipulations can also be formulated and shown to be valid using the basic laws. Thus, for instance, we could have used the *Disjunctive Argument* rule, a rule which allows us to infer α from $\alpha \lor \beta$ and $\neg\beta$. Using this particular rule, the result B is obtained in a single step from statements 7 and 8.

A.2 Predicate Calculus

Although propositional calculus is a useful language for expressing simple concepts, it is often inadequate for expressing many facts about the real world. The language is too coarse and primitive to express the concept of an object, properties of an object, or relationships among several objects [61]. These shortcomings arise because the propositions in propositional calculus are a unit as a whole and not decomposable entities. Thus, if the symbol P1 represents the proposition "Frame A has 5 bays" and P2 represents "Frame B has 3 bays," one cannot infer that Frame A has more bays than Frame B. In predicate calculus, the number of bays can be expressed as an attribute of each of the objects Frame A and Frame B and it is possible to reason about such an attribute. Predicate calculus enhances the expressive power of propositional calculus by introducing additional concepts such as predicates, functions, variables, and quantifiers.

A.2.1 Syntax

There is a greater variety of symbols available in predicate calculus as compared to propositional calculus. The basic *truth symbols* remain the same: `True` and `False`. In addition, there are *constant symbols* or *constants* (e.g., `A5`, `Red`, `123`, `FrameB`, `Delaware`), *variable symbols* or *variables* (e.g., `x`, `pt`, `y1`), *function symbols* or *functions* (e.g., `Sin`, `Distance`, `Age`, `Log`), and *predicate symbols* or *relations* (e.g., `Even`, `Taller`, `Neighbor`). Constants, variables, and functions correspond well with the similar notions in algebra. As the name implies, constants denote those elements whose value never changes. Variables, on the other hand, denote the elements that can assume different values at different instances. Function symbols denote operators or functions that operate on a fixed number of arguments and evaluate to a legal value. Predicate symbols denote relations that hold among a fixed number of arguments.

We adopt the following notational convention: A constant symbol is a sequence of alphanumeric characters in which the first character is either numeric or uppercase alphabetic. A variable symbol is a sequence of alphanumeric characters in which the first character is lowercase alphabetic. Function symbols can either be a sequence of alphanumeric characters in which the first character is uppercase alphabetic, or one of the following functional operators: $+$, $-$, $*$, and $/$. Predicate symbols can either be a sequence of alphanumeric characters in which the first character is uppercase alphabetic, or one of the following mathematical operators: $=$, $<$, $>$, $\leq$, and $\geq$. In predicate calculus, two punctuation marks are also employed: (i) commas to separate the multiple arguments of a function or a predicate, and (ii) parentheses to enclose such arguments.

Functions and relations have associated arities (the number of arguments the function or the relation requires) which are always positive integers. The arguments that functions and relations take are called terms. A *term* can be either a constant, a variable, or a functional expression, where a *functional expression* consists of a function symbol of arity n followed by a list of n terms enclosed in parentheses and separated by commas. The following are legal functional expressions:

```
Sqrt(Fy)
```

`/(640,Sqrt(Fy))`[3]

`MomentOfInertia(section,YY)`

(Note that the functional expressions themselves can serve as arguments to other functions.) These functional expressions and the constants and variables such as `A5`, `123`, and `y1` are all legal examples of terms.

From these terms, sentences can be formed. A *sentence* can be of one of three types: atomic sentence, logical sentence, or quantified sentence. An *atomic sentence* can either be a truth symbol or a predicate symbol of arity n followed by a list of n terms enclosed in parentheses and separated by commas. Examples of atomic sentences are:

`NumberOfBays(FrameA,5)`

$>$ `(x,0)`[4]

A *logical sentence* is formed by combining sentences using logical connectives such as *and* ($\wedge$), *or* ($\vee$), *not* ($\neg$), *implies* ($\Rightarrow$), and *equivalent* ($\equiv$). The types of sentences that can be formed with these connectives are the same as those for propositional calculus. An example of a logical sentence is:

`NumberOfBays(FrameA,5)` $\wedge$ `NumberOfBays(FrameB,3)`

A *quantified sentence* can either be a universally quantified or an existentially quantified sentence. A *universally quantified* sentence consists of the universal quantifier, *for all* ($\forall$), a variable, followed by a sentence. For instance, the sentence

$\forall$`y Partition(y)` $\Rightarrow$ `PotentialColLine(y)`

is a universally quantified sentence. The intended meaning of the sentence is that every partition is a potential column line. An *existentially quantified* sentence consists of the existential quantifier, *for some*[5] ($\exists$), a variable, followed by a sentence. For instance, the sentence

$\exists$`x Eccentricity(Load,x)` $\wedge\ \neg$`(x = 0)`

is an existentially quantified sentence. The intended meaning of the sentence is that there is a load with non-zero eccentricity.

For clarity, parentheses and brackets are used to specify the *scope* of quantifiers. A sentence that uses both universal and existential quantifiers is the following.

[3]For readability purposes, the infix notation is sometimes preferred in the case of common mathematical functions; for instance, (x/y), or x/y, may be used in place of $/(x,y)$.

[4]For readability purposes, the infix notation is sometimes preferred in the case of common mathematical relations; for instance, $(x > y)$, or $x > y$, may be used in place of $> (x,y)$.

[5]Also termed as *there exists.*

$$\forall \texttt{line}\ [\texttt{PotentialColLine(line)} \wedge \neg\exists \texttt{otherLine}\ (\texttt{ColLine(otherLine)} \wedge \neg(\texttt{line} = \texttt{otherLine}) \wedge \texttt{Distance(otherLine, line)} < \texttt{MinDistance})] \Rightarrow \texttt{ColLine(line)}$$

The intended meaning of the sentence is that all potential column lines which are no closer than a certain minimum distance to any other column line, become column lines themselves.

A language in which functions and predicates are constants is called a *first-order* language. A language in which functions and predicates also vary is called *second-order*. In a second-order language, functions and relations can serve as quantifying variables to quantifiers. In this appendix, however, we will restrict ourselves to first-order predicate calculus.

A.2.2 Semantics

Now that the formulation of sentences in predicate calculus has been discussed, let us examine how their truth values can be evaluated. For this purpose, introduction of some additional notions—such as interpretation and variable assignment—is necessary. We first examine the truth or falsehood of an atomic sentence. Later in this section we discuss evaluation of truth values of logical and quantified sentences.

The motivation for formulating sentences in logic is to conceptualize a world about which one is trying to reason. In the process, certain symbols are used to represent the elements of the world, since it is often not possible to substitute the actual elements—which can be physical and conceptual entities—in the text. Thus, in the example in Section A.1.3, we expressed certain conceptual ideas about the world in terms of single letter propositional symbols. An *interpretation* maps the elements of the language to the elements of the world that one is trying to conceptualize. Interpretation is similar to dereferencing the symbols in a programming language—bringing into consideration the object the symbol denotes rather than the symbol itself. Indeed, the idea of interpretation is so natural to us humans that we do it frequently without even being aware of it. Thus, when we say that an I-beam has three parts (two flanges and one web), we mean that the members of the category of objects represented by the word `I-beam` each have three parts, rather than that the word `I-beam` has three parts.

With reference to predicate calculus, an interpretation over a domain (a set of objects, functions, and relations) provides a mapping for the constant symbols, the function symbols, and the predicate symbols. To each constant symbol in the language, the interpretation assigns an object in the domain. For each function symbol of arity n, the interpretation assigns an n-ary function defined over the objects in the domain. *A functional expression* *evaluates to a value which itself is an object of the domain.* For instance, the function symbol Abs of arity 1 can be assigned the unary mathematical function 'absolute' over the set of real numbers. The value of Abs(x) will then always be a nonnegative real number for any value of x. For each predicate symbol of arity n, the interpretation assigns an n-ary relation defined over the objects in the domain such that *the value of an atomic sentence composed of the relation and n terms always evaluates to either true or false* depending on whether the relation holds for the specified objects or not. For instance, the predicate symbol Gtz of arity 1 can be assigned, over the set of real numbers, a relation 'greater-than-zero' which is true for all positive numbers and false for nonpositive numbers. The value of Gtz(x) will always evaluate to either *true* or *false* for all values of x.

Analogous to the concept of interpretation is the concept of variable assignment. A *variable assignment* maps the free variables in the language to the elements of the domain. *Free variables* in a sentence are those variables that are not within the scope of any quantifier.

When we write sentences, we have a certain *intended interpretation* in mind, that is, we associate the elements of the language with specific elements in the world being conceptualized. For instance, in the intended interpretation in the example of Section A.1.3, the symbol B represented the concept that the stiffness of the beams should be increased. But it is equally acceptable to assign a totally different interpretation (as well as variable assignment) over an entirely different domain to the same set of sentences. To illustrate, let us consider the sentence

$\forall$y P(y) $\Rightarrow$ Q(F(y),z)

under two different interpretations. The sentence has two predicate symbols (P and Q), one function symbol (F), and one free variable (z). Let us first consider the following interpretation and variable assignment over the domain of steel sections and real numbers.

P is 'rolled-section' relation,
Q is 'greater-than' relation,
F is 'cross-sectional-area' function, and
z is '0'.

The given sentence intuitively transforms to:

the cross-sectional area of all rolled-sections is greater than zero.

This statement turns out to be *true*. However, consider another interpretation and variable assignment, this time over the domain of beams.

P is 'simply-supported-beam' relation,

Q is 'is-a' relation,

F is 'left-support' function, and

z is 'roller'.

The intuitive meaning of the sentence becomes:

the left support of all simply supported beams is a roller

which certainly is *false*.

In general, the truth of a sentence depends on the particular interpretation and the particular variable assignment being employed. A sentence may be *true* under some interpretation and variable assignment, and *false* under a different interpretation and variable assignment. Normally, symbols for predicates and functions are so chosen that they correspond well with what they represent in the intended interpretation. (We rely on such practice throughout the appendix. Our prime motive, however, is brevity—instead of stating every time "consider the interpretation ... over the domain ... where predicate and function symbols ... and ... represent ... and ... respectively, and consider the variable assignment ...," we instead appeal to intuition to convey the meaning of symbols involved.) However, as the previous example illustrates, it is important to distinguish between the symbols and the referents.

The preceding paragraphs define the semantics of atomic sentences. The truth values of logical sentences can be determined by following the rules for logical connectives presented earlier in Section A.1.2. The truth values of quantified sentences can be ascertained as follows. A universally quantified sentence $\forall\mu\,\alpha$ is *true* under a given interpretation if and only if the sentence α is *true* for all variable assignments of μ. An existentially quantified sentence $\exists\mu\,\alpha$ is *true* if and only if there exists a variable assignment for μ for which the sentence α is *true*. In essence, the universal quantifier enables us to express some property which is true of all the objects in the domain while the existential quantifier enables us to express some property of an individual object without specifying the object. For example, consider the sentence

$\forall$x TrussMember(x) $\Rightarrow$ BendingMoment(x, 0)

With this sentence we have expressed that the bending moment in all the truss members is always zero, without enumerating any of the truss members. Similarly, we used existential quantifier earlier to express that a load has certain eccentricity without specifying the eccentricity.

A sentence is termed *satisfiable* if and only if there exists an interpretation and variable assignment for which the sentence is *true*. Otherwise, the sentence is *unsatisfiable* (or *contradictory*). If a sentence is satisfied by every interpretation and variable assignment, it is termed as a *valid* sentence. For example, the propositional sentence $\mathtt{P} \vee \neg\mathtt{P}$ is a valid sentence (it will always evaluate to *true* irrespective of the truth value of P). In a manner analogous to individual sentences, these definitions can be extended to sets of sentences. In particular, a set of sentences is *satisfiable* or *consistent* if and only if all the sentences in the set are *true* for some interpretation and variable assignment. Otherwise, the set is *unsatisfiable* or *inconsistent*.

A.2.3 Inference

The rules of inference listed in Fig. A.1 are also applicable to predicate calculus. In addition, there are some rules applicable to predicate calculus alone. One such rule is the *universal instantiation rule*, which is quite intuitive. According to this rule, from a universally quantified sentence one can infer any other sentence that has been obtained by replacing the universally quantified variable with a suitable term. For example, from

$\forall\mathtt{x}\ \mathtt{TrussMember(x)} \Rightarrow \mathtt{BendingMoment(x,0)}$

the following (and many other such sentences) can be inferred.

$\mathtt{TrussMember(AB)} \Rightarrow \mathtt{BendingMoment(AB,0)}$

$\mathtt{TrussMember(TopChord(Truss1))} \Rightarrow \mathtt{BendingMoment(TopChord(Truss1),0)}$

The term used for replacing the universally quantified variable can be a constant, a functional expression, or a variable. However, if the variable is a free variable or the functional expression contains free variables, it has to be ensured that the names of such free variables do not match with the names of other variables already present in the sentence.

The *existential instantiation rule* is similar to the universal instantiation rule and is applicable to existentially quantified sentences. The restrictions on the terms that can be used for replacement are, however, quite different.

Another rule of inference is resolution. To understand resolution, it is necessary to first understand two other concepts: clausal form and unification. A detailed description of clausal form and conversion of ordinary sentences to clausal form can be found in Ref. [26]. For our purposes, it suffices to say that *any* sentence formed by following the rules of syntax for predicate calculus described earlier can be equivalently represented in a simpler form called *clausal form* which consists of clauses. A *clause* is a set of literals (atomic sentences or their negations) that represents the disjunction of such literals. For example, the clause

{¬`TrussMember(x)`, `BendingMoment(x,0)`}

is equivalent to the sentence

¬`TrussMember(x)` ∨ `BendingMoment(x,0)`

Unification in some sense is like pattern matching. We say that two expressions can be unified if they can be made identical by replacing some or all of their variables with other terms. For instance, the two expressions

`BeamSupport(Beam1,Left,supportingBeam)` and
`BeamSupport(someBeam,Left,Beam5)`

can be unified by replacing `supportingBeam` by `Beam5` in the first expression and `someBeam` by `Beam1` in the second. Such replacements, or *substitutions*, for variables (written, for example, as [`supportingBeam` ← `Beam5`, `someBeam` ← `Beam1`]) that unify two expressions are called *unifiers*. There may be more than one unifier for two expressions that are unifiable. In such a case, a *most general unifier* (mgu) is one from which other unifiers can be obtained after appropriate substitutions.

After this introductory background, we can now proceed to discuss the resolution principle.[6] The *resolution principle* can be applied to any two clauses, Θ and Ω, if one clause contains a positive literal (an unnegated atomic sentence) α, whereas the other contains a negative literal (a negated atomic sentence) $\neg\beta$, such that α and β can be unified by a most general unifier ν. If these conditions are met, then from the two clauses Θ and Ω, we can infer another clause which is obtained by applying the substitution ν to the union of Θ and Ω minus α and $\neg\beta$. Mathematically, this can be written as

$$\begin{array}{lll} \text{From} & \Theta & (\alpha \in \Theta) \\ \text{and} & \Omega & (\neg\beta \in \Omega) \\ \text{infer} & [(\Theta - \{\alpha\}) \cup (\Omega - \{\neg\beta\})] \leftarrow \nu & \text{where } \alpha \leftarrow \nu \equiv \neg\beta \leftarrow \nu \end{array}$$

(The symbol '←' represents the substitution operator.)

As an illustration of application of resolution, consider the following two clauses.

{¬`TrussMember(x)`, `BendingMoment(x,0)`}
{`TrussMember(TopChord(Truss1))`}

The mgu [`x`←`TopChord(Truss1)`] will unify the two literals `TrussMember(x)` and `TrussMember(TopChord(Truss1))`. Thus, these two clauses will resolve to produce

[6]The definition here does not include the notion of factors. For a more general definition see Ref. [26].

{`BendingMoment(TopChord(Truss1),0)`}

The resolution principle is quite powerful. Logic programming languages exploit the generality and the power of resolution to deduce facts and, in the process, instantiate variables (using the coupled notion of unification). In many cases, unifiers are of prime importance as one may be interested in the values of variables that will satisfy certain conditions. In other cases, one may be interested in finding out if a statement logically follows from a consistent set of statements. The procedure for these latter cases is similar to the method of contradiction employed for proving mathematical theorems. The statement to be proved is first negated and is added to the existing fact base of clauses. If the empty clause (equivalent to a *false* statement), {}, can then be deduced from this new fact base using resolution, it will mean that the set of sentences has become inconsistent because of the addition of the negated statement—implying that the original statement does indeed follow from the original fact base. This process is called *resolution refutation*.

How can one be sure that the conclusions derived from a set of sentences using a particular rule of inference will indeed be correct (or will logically be implied by the set of sentences)? How can one be sure that *all* the conclusions that logically follow from a given set of sentences can be derived using a particular rule of inference? In logicians' parlance, these properties of inference rules are known as *soundness* and *completeness*, respectively. All the rules of inference given in this appendix are sound but none is complete. Resolution, though, has an important property of being *refutation complete*, i.e., if a given set of sentences is unsatisfiable, the empty clause can always be deduced using resolution alone.

A.3 Application to Structural Analysis

In this section, an example of structural analysis is described to illustrate the concepts presented earlier. The declarative approach is contrasted with the procedural approach of solving a problem. The problem is part of a larger problem discussed by Fenves [18] and involves determining reactions at the ends of simply-supported beams that comprise an idealized floor system. The information about the floor geometry and loading is provided to the program as input. The floor itself is composed of various rectangular areas that are loaded with uniformly distributed area loads and behave as one-way slabs, transferring their accumulated loads to two parallel supporting beams (specified by the engineer in the input) at the edges. The beams lie on a rectangular grid and form a hierarchy, transferring loads through beams lower in the hierarchy ultimately to the columns. One sample floor layout

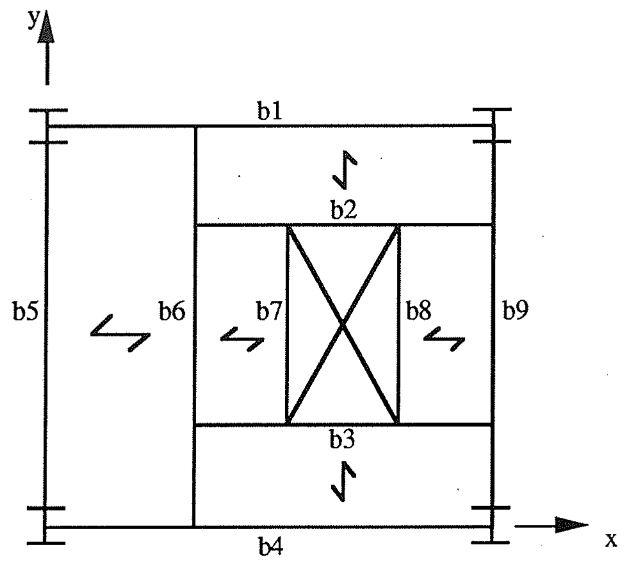


Figure A.3: **A Sample Floor Layout.** Solving for end reactions of a beam requires that end reactions of all beams supported by the beam under consideration have been determined.

is shown in Fig. A.3. The problem appears to be trivial, but its solution requires knowledge about beam hierarchy since the computation of end reactions of a beam can proceed only if the reactions of all other beams supported by the beam under consideration have been determined.

To appreciate the difference between the procedural approach and the declarative approach, first consider the following abstract procedure (adapted from Ref. [18]) that can be used to solve the problem.

```
Convert Area Loads
 For each area load Do
  locate edge beams under the area from the grid
  assign a line load to the supporting beams

Carry Down Loads
 For each beam Do
  initialize counter to 0
 For each beam Do
  increase the counters of the left and right supports by 1
 Repeat
```

```
  For each beam Do
    If counter = 0 Then
      place beam designation into the next location of a stack
      decrease the counters of the left and right supports by 1
Until all beams are in stack
```

Solve for End Reactions

```
While stack is not empty
  pop a beam from the top of the stack
  compute its end reactions
  assign the end reactions as point loads to underlying beams (or columns)
```

Effectively, the algorithm first explicitly determines the sequence in which the end reactions for the beams should be determined, and then proceeds to solve for the reactions. However, in solving problems of this type, a structural engineer does not typically determine such a sequence beforehand but relies on the following fact:

- If the end reactions of all the beams that a beam under consideration supports have been computed then the supporting beam can be solved.

Other knowledge that is needed for the solution to this problem follows.

- Each of the two beams acting as supports to an area carries a uniformly distributed line load (udll) of intensity equal to half of the area's dimension in the direction perpendicular to supporting beams times the loading intensity on the area.
- The extent (portion of the span) of loading on the beam resulting from such area loads coincides with that portion of the beam which corresponds to the edge of the area.
- For purposes of computing end reactions on a simply supported beam, a udll can be treated as an equivalent point load acting at the midpoint of the udll and having a magnitude equal to intensity times the extent of udll.

In addition, laws of static equilibrium are also needed to determine the end reactions of a beam.

In contrast to the procedural approach, we will encode these pieces of knowledge as declarative statements in predicate calculus and reason from them. However, before the statements can be formulated in predicate calculus syntax, we must decide on a conceptualization; for instance, we must choose what relations and functions will be used for presenting the information about the floor geometry and loading. One possibility is the following.

```
AreaLoad(areaName,loadIntensity)
AreaSupport(areaName,Left,suptgBeamOnLeft)
AreaSupport(areaName,Right,suptgBeamOnRight)
AreaEdges(areaName,xLeft,yLeft,xRight,yRight)
BeamSupport(beamName,Left,leftSupport)
BeamSupport(beamName,Right,rightSupport)
BeamEnds(beamName,xLeft,yLeft,xRight,yRight)
UDLLoad(beamName,from,to,loadIntensity)
PtLoad(beamName,location,magnitude)
```

The importance of proper conceptualization cannot be overemphasized. The one shown here is not unique; there are many others. However, with experience, one develops intuition and judgement as to which ones are natural as well as efficient for reasoning. For the area support information, for instance, we could have chosen relations like `AreaSupportLeft` and `AreaSupportRight`, eliminating one of the arguments. These relations, however, are undesirable because they embed the semantics of arguments in the predicate names where they are not amenable to manipulation.

Using the proposed choice of relations for presenting the floor geometry and loading information, the items of knowledge presented earlier can be formulated as the following predicate calculus sentences. Note that there is a direct correspondence between the English sentences and the predicate calculus sentences.

A beam is solvable if it has not already been solved and there does not exist another unsolved beam that is supported by this beam.

$[\neg \texttt{Solved(beam)} \wedge$
$\neg \exists \texttt{higher}\ (\texttt{BeamSupport(higher, direction, beam)} \wedge \neg \texttt{Solved(higher)})]$
$\Rightarrow \texttt{Solvable(beam)}$

If a beam supports an area on which there is certain loading, and the extent and intensity of loading resulting on the supporting beam have been determined, then the beam can be assigned a udll of such intensity and extent.

$[\texttt{AreaSupport(area, side, beam)} \wedge \texttt{AreaLoad(area, load)} \wedge$
$\texttt{LoadingExtents(area, beam, from, to, leftEdge, rightEdge)} \wedge$

$(\texttt{intensity} = \texttt{load} * \texttt{Abs}(\texttt{rightEdge} - \texttt{leftEdge})/2)]$
$\quad \Rightarrow \texttt{UDLLoad}(\texttt{beam}, \texttt{from}, \texttt{to}, \texttt{intensity})$

If the y coordinates of the two ends of a beam are the same (or, in other words, the beam is parallel to the x-axis) then the x coordinates of the two edges of the area that are perpendicular to the beam can be used to determine the extent of the loading.

$[\texttt{BeamEnds}(\texttt{beam}, \texttt{xLeft}, \texttt{y}, \texttt{xRight}, \texttt{y}) \land \texttt{AreaEdges}(\texttt{area}, \texttt{x1}, \texttt{y1}, \texttt{x2}, \texttt{y2})]$
$\quad \Rightarrow \texttt{LoadingExtents}(\texttt{area}, \texttt{beam}, \texttt{x1}, \texttt{x2}, \texttt{y1}, \texttt{y2})$

If the x coordinates of the two ends of a beam are the same (or, in other words, the beam is parallel to the y-axis) then the y coordinates of the two edges of the area that are perpendicular to the beam can be used to determine the extent of the loading.

$[\texttt{BeamEnds}(\texttt{beam}, \texttt{x}, \texttt{yLeft}, \texttt{x}, \texttt{yRight}) \land \texttt{AreaEdges}(\texttt{area}, \texttt{x1}, \texttt{y1}, \texttt{x2}, \texttt{y2})]$
$\quad \Rightarrow \texttt{LoadingExtents}(\texttt{area}, \texttt{beam}, \texttt{y1}, \texttt{y2}, \texttt{x1}, \texttt{x2})$

A udll is equivalent to a point load of magnitude equal to intensity times span and location at the midpoint of the extent of udll.

$\texttt{UDLLoad}(\texttt{beam}, \texttt{from}, \texttt{to}, \texttt{intensity})$
$\quad \Rightarrow \texttt{PtLoad}(\texttt{beam}, (\texttt{from} + \texttt{to})/2, \texttt{intensity} * \texttt{Abs}(\texttt{to} - \texttt{from}))$

These statements can be translated into any logic programming language in a fairly straightforward manner. (A Prolog implementation, for instance, is given in Ref. [43].)

A.4 Application to Standards Processing

In the domain of structural design, designed entities need to meet certain established specifications or standards. Previous research works have emphasized the need for treating design standards as data for application programs (or in other words, processing declaratively represented standards) instead of hard-coding them in the program structure [74,82]. One representation scheme that has been employed previously for this purpose is logic decision tables (DT's) [19,23,1]. Table A.1 shows a sample decision table for deciding on whether a section is compact or partially compact (based on Section 1.5.1.4.1 of AISC Specifications [5]) for the purpose of determining the allowable stress for bending about the major axis. (See Ref. [38] for a description of decision tables.) Only I-shaped sections are considered in the sample DT.

In this section we describe application of predicate calculus to standards processing. The description is based on Ref. [44].

	S_1	S_2	S_3	S_4
C_1: $\frac{b_f}{2t_f} \leq \frac{65}{\sqrt{F_y}}$	Y	Y	N	N
C_2: $\frac{b_f}{2t_f} \leq \frac{95}{\sqrt{F_y}}$	Y*	Y*	Y	Y
C_3: $\frac{f_a}{F_y} \leq 0.16$	Y	N	Y	N
C_4: $\frac{d}{t_w} \leq \frac{640}{\sqrt{F_y}}(1 - 3.74\frac{f_a}{F_y})$	Y	I	Y	I
C_5: $\frac{d}{t_w} \leq \frac{257}{\sqrt{F_y}}$	I	Y	I	Y
Section is Compact	X	X		
Section is Partially Compact			X	X

Table A.1: A Sample Decision Table

A.4.1 Representation of Design Standards

Although decision tables have their foundations in logic, they do not have all the expressive power of logical languages like first-order predicate calculus. For example, if a rule prescribes a disjunction of actions, it cannot be expressed in a DT format unless the actions themselves are not simple actions but composite actions. Converting a limited-entry DT into predicate calculus sentences is a straightforward operation. The conditions and actions can be replaced by appropriate relations. The upper part (consisting of condition entries) of the rules is stated as the antecedent (composed of positive literals corresponding to `Y` in the rule and negated literals corresponding to `N` in the rule) and the lower part (consisting of action entries) as the consequent of an implication. The immaterials (`I`) can be simply ignored when converting to predicate calculus format. For instance, the conversion of the decision table in Table A.1 follows.

S_1: $C1 \wedge C2 \wedge C3 \wedge C4 \Rightarrow$ `Compact(section)`
S_2: $C1 \wedge C2 \wedge \neg C3 \wedge C5 \Rightarrow$ `Compact(section)`
S_3: $\neg C1 \wedge C2 \wedge C3 \wedge C4 \Rightarrow$ `PartiallyCompact(section)`
S_4: $\neg C1 \wedge C2 \wedge \neg C3 \wedge C5 \Rightarrow$ `PartiallyCompact(section)`

In many cases, the condition entries of the rules can be simplified to render them syntactically more natural and more efficient for processing. For instance, the condition entries for implied `Yes` and implied `No` can be completely ignored. In fact, by building and storing a fact base of implied truth values in advance, it is possible to resolve and eliminate certain redundant clauses in the converted rules. Hence, with a statement of the form $C1 \Rightarrow C2$ (which makes explicit the interdependency between the two relations) the statements S_1 and S_2 can be resolved, yielding:

S_1': $C1 \wedge C3 \wedge C4 \Rightarrow$ `Compact(section)`

S_2': $C1 \wedge \neg C3 \wedge C5 \Rightarrow$ `Compact(section)`

Moreover, we can combine the statements that result in the same action to arrive at a more compact representation. Thus, Table A.1 can be equivalently expressed by the following two sentences.

S_{12}: $C1 \wedge [(C3 \wedge C4) \vee (\neg C3 \wedge C5)] \Rightarrow$ `Compact(section)`

S_{34}: $\neg C1 \wedge C2 \wedge [(C3 \wedge C4) \vee (\neg C3 \wedge C5)] \Rightarrow$ `PartiallyCompact(section)`

An interesting problem arises when an `else` rule is present in the decision table. One way to handle the `else` rule is the following. Left hand sides of all the other rules is negated and their conjunction is included as the left hand side of the `else` rule. An alternative is to take advantage of the inference mechanism of the particular implementation tool being employed. For example, for Prolog language, the `else` rule can be a rule without any antecedent conditions, placed at the end of the set of rules representing a particular specification. (The feature of `cuts` can also be used in the earlier rules to prevent instantiation of the `else` rule when other rules are applicable.) In Epikit, one can use the modal operator **Provable** to express that the action of `else` rule should be executed when no other action can be proved to be applicable. Thus, if Table A.1 included an `else` rule leading to the action `Noncompact(section)`, we could have represented it as

$\neg$**Provable**(`Compact(section)`) $\wedge$ $\neg$**Provable**(`Partiallycompact(section)`)
$\Rightarrow$ `Noncompact(section)`

This has the benefit of improved efficiency but also has the disadvantage of being implementation dependent. Depending upon the relative importances of clarity and efficiency, the appropriate method of handling the `else` rule can be chosen.

A.4.2 Checking Properties of Design Standards

Design standards must meet the three requisite properties: completeness, uniqueness (absence of redundancy and contradiction), and correctness [20]. While the first two properties relate to the syntax of representation, the third one is essentially a semantical issue. Given here are formal tests for checking completeness, lack of redundancy, and lack of contradiction in the predicate calculus representation of design standards. These tests can be performed once and for all when a predicate calculus formalization of a design standard is developed. We represent specifications as groups of statements of the form S_i: $L_i \Rightarrow R_i$, where L_i represents the part to the left of the implication while R_i represents the part to the right of the implication. Each group represents rules for a single specification; e.g., a group may represent the specification for deciding on compactness of a steel section. The total number of statements in a group is denoted by n.

1. *Completeness*: A set of specifications is complete if at least one rule in each specification group is applicable for any given combination of logical variables. Completeness can be verified by proving the validity of the following sentence for each group.

   $$\bigvee_{i=1}^{n} L_i$$

   When the `else` rule is present in the equivalent DT representation for the group, the specification will always be complete (since at least the `else` rule will be applicable if no other rule applies).

2. *Lack of Redundancy*: A set of specifications is said to be redundant if more than one rule is applicable in some specification group for a given combination of logical variables. The lack of redundancy in a set of specifications can be verified by proving the validity of the following sentence for each group.

   $$\bigwedge_{i=1,j=i+1}^{i=n-1,j=n} (\neg L_i \vee \neg L_j)$$

3. *Lack of Contradiction*: Contradiction is a special case of redundancy. A set of specifications is said to be in contradiction if different actions are suggested by multiple rules (in the same specification group) that are applicable for a given combination of logical variables. The lack of contradiction in a set of specifications can be verified by proving the validity of the following sentence for each group.

   $$\bigwedge_{i=1,j=i+1}^{i=n-1,j=n} [\neg L_i \vee \neg L_j \vee (R_i = R_j)]$$

As an illustration, we shall check the completeness of the criteria regarding compactness of steel sections; the criteria have been given as statements S_{12} and S_{34} previously. Thus, if we can prove the validity of the sentence

$$[C1 \wedge [(C3 \wedge C4) \vee (\neg C3 \wedge C5)]] \vee [\neg C1 \wedge C2 \wedge [(C3 \wedge C4) \vee (\neg C3 \wedge C5)]]$$

we will be assured of the completeness of specification regarding compactness of sections. In order to prove the validity of this sentence, we first negate it, convert it to the clausal form, and then perform resolution on the resulting clauses to deduce the empty clause ($\{\}$). The clausal form of the negated sentences is shown in statements 1–4 that follow. Statements 5 and 6 are deduced from these using resolution while statement 7 is deduced from statements 5 and 6.

1. $\{\neg C1, \neg C3, \neg C4\}$
2. $\{\neg C1, C3, \neg C5\}$
3. $\{C1, \neg C2, \neg C3, \neg C4\}$
4. $\{C1, \neg C2, C3, \neg C5\}$
5. $\{\neg C2, \neg C3, \neg C4\}$ (Resolution: 1, 3)
6. $\{\neg C2, C3, \neg C5\}$ (Resolution: 2, 4)
7. $\{\neg C2, \neg C4, \neg C5\}$ (Resolution: 5, 6)

Since there are no positive literals for any of $C2$, $C4$, and $C5$, statement 7 can not be resolved any further. Therefore, it is not possible to deduce the empty clause from this set of statements. The statement can thus not be proven valid. That is, the specification regarding the compactness of sections as coded in Table A.1 and rules S_{12} and S_{34} is incomplete. This process also suggests that simultaneous falsity of conditions $C2$, $C4$, and $C5$ can not be accounted for in the given set of rules. Thus, in order to make the specification complete, additional rule(s) must be introduced that take care of the situations currently unaccounted for, including when $C2$, $C4$, and $C5$ are false. In this particular example, an `else` rule with the resultant action of `Noncompact(section)` on the right hand side will obviously suffice. In some other instances, the specification can be made complete by an `else` rule with the action `NotApplicable` to signify that the specification is applicable only for the given cases.

A.4.3 Discussion

As mentioned earlier, decision tables have their basis in logic but lack some of the expressive power which predicate calculus possesses. Any decision table can be converted into a limited-entry decision table, which in turn can be converted into predicate calculus format. One significant advantage of predicate calculus is that, without any extra effort, the same representation of design standards can be used not only for conformance checking of a given design but also to deduce the design descriptions themselves. On one hand, the unification and resolution processes can instantiate the design parameters such that they satisfy the specifications which are stored as axioms. On the other hand, through truth evaluation by resolution refutation, one can ascertain the conformance of a given design with the specifications. Rasdorf and Lakmazaheri [73] have illustrated this thorugh application to AISC allowable stress design specifications pertaining to axially loaded components.

In their work, Garrett and Fenves [23] treated functions as degenerate decision tables that have no conditions, only one possible action, and only one rule that specifies the single action. There is a direct correspondence between predicate calculus functions and the functions in mathematics as commonly used in standards, and therefore the need for an artificial construct to handle functions can be avoided. Just as a standard can be viewed as a network of decision tables, it can also be viewed as a network of logical axioms. It has been claimed that of all the modeling tools (DT's, information network, and organization system) employed for standards, decision tables provide the most complete and precise description of provisions [72]. Predicate calculus maintains all the benefits of the DT representation and provides some additional benefits as discussed earlier.

A.5 Closing Remarks

In this appendix we have deliberately emphasized informality rather than strict exactness in our treatment of logic. The reader may find minor variations in the description of logic in some other works. For example, many authors exclude the concept of truth values (*true* and *false*) when describing formal logic systems and proof techniques. Despite the adequacy of such an approach, we have chosen to include the notion of truth values to impart an intuitive flavor to the syntactic manipulations.

We should also note that propositional and predicate calculi are not the only logical languages. The governing characteristic of a logical language is that it must possess formal semantics; i.e., one should be able to evaluate the truth or falsehood of statements in the language for a given interpretation of the symbols. Decision tables, semantic nets, even frames and rules can also be regarded as logical languages. All these specialized languages, however, provide only a subset of the expressive power and inferencing capabilities of predicate calculus, and can be defined in terms of predicate calculus.

A.5.1 Further Reading

In addition to the rules of inference and laws of propositional algebra described in Section A.1.2, there are several other techniques for proving validity (and invalidity) of propositional calculus sentences. These include truth tables, semantic trees, and falsification. A good description of these techniques can be found in Ref. [61]. Refs. [70,71] are other excellent book-length treatments of the two languages discussed in this appendix. We have described clausal resolution in this appendix. Description of non-clausal resolution can be found in Ref. [62].

Ref. [4] provides a good introduction to the field of logic programming and knowledge engineering. For a wider perspective on applications of logic to the field of Artificial Intelligence, the reader is encouraged to refer to Ref. [26]. Proceedings of the International Conferences on Logic Programming [52,48] are a good source of the latest developments in the area of logic programming.

In this appendix, we confined ourselves to describing monotonic reasoning. In monotonic reasoning, sound rules of inference are used that lead to conclusions that will be true under all circumstances. As it turns out, this is not enough to model common-sense reasoning. In nonmonotonic reasoning one makes certain provisional inferences using rules of inference that are not sound. These provisional inferences are not guaranteed to be correct and may have to be later retracted in the presence of additional evidence. This may lead to a nonmonotonic growth of the fact base. The field of nonmonotonic reasoning has attracted significant attention recently and there is a growing body of work in this area. There are several approaches being pursued, including modal operators [65,66], default theories [75], and circumscription [63,55].

The deductive power offered by logic comes at a price—it is computationally expensive. To overcome the drawback of slow execution and consequent long run-times with increasing size of the fact base, *semantic attachment*, wherein appropriate computational structures are used in place of rules of inference, can be employed. The idea of semantic attachment was first proposed by Green [30] and later expounded by Weyhrauch [87]. The primary concept behind semantic attachment is the following. Using sound rules of inference ensures that any sentence derived from a given fact base will be consistent for *all* the possible interpretations for which the database is consistent. However, we may be equally well off with conclusions of less strength for most of our purposes, i.e., even when the derived sentence does not hold true for all the possible interpretations as long as it holds true in the intended interpretation. Engineering systems of significant size will have to rely on semantic attachment extensively.

Some researchers have expressed reservations about the adequacy of logic. In particular, they point to the inability of logic to model other types of reasoning besides deduction, and inability to provide guidance regarding which deductions to perform out of the possible candidates. Nonmonotonic reasoning and meta-level architecture [24] are some of the proposed solutions in response to such reservations. One of the more extensive discussions about the suitability of logic appears in a special volume of *Computational Intelligence* which includes a critique of logic by McDermott [64] and commentary by leading researchers on both sides of the issue.

Through naming comes knowing; we grasp an object mentally, by giving it a name—hension, prehension, apprehension. And thus through language create a whole world, corresponding to the other world out there. Or we trust that it corresponds. Or perhaps, like a German poet, we cease to care, becoming more concerned with the naming than with the things named: the former becomes more real than the latter. And so in the end the world is lost again. No, the world remains—those unique, particular, incorrigibly individual junipers and sandstone monoliths—and it is we who are lost. Again. Round and round, through the endless labyrinth of thought—the maze.

— *Edward Abbey,* Desert Solitaire *(1971)*

B
Knowledge Interchange Format

Because symbols like ∀, ∃, ≡, ¬, etc., are not available on a typical keyboard, and for other reasons of implementation, the syntax of logic programming languages is usually not identical to the one described in Section A.2.1. The syntax may differ in such details as placement of commas and parentheses, case insensitivity, etc. KIF, or Knowledge Interchange Format, is no exception; the correspondence between KIF's syntax and the one in Section A.2.1, however, is still instructive. This appendix briefly describes the syntax of KIF with suitable illustrative examples from FFG's knowledge base. The description is not intended to be comprehensive—indeed the description is minimally sufficient just to give a flavor of KIF as well as FFG's knowledge base. (All examples in this appendix are taken from FFG's knowledge base.) For a complete definition of KIF, the reader should consult Ref. [27].

Since KIF is meant to be a knowledge representation formalism for intercommunicating Lisp programs, and since Lisp does not distinguish between uppercase and lowercase symbols, the convention of using symbols starting with an uppercase letter for constants and those with a lowercase letter for variables is not applicable in KIF. Instead, *variables* in KIF are distinguished by placing a $ sign before them.[1] Thus, `$span` and `$conc-depth` are both variables in KIF. The connectives and quantifiers are *sentence operators* in KIF. Hence, there are operators like `not`, `and`, `=>`, `forall`, and `exists`. In addition to the sentence operators, there is one more type of operators, namely *term operators*. Examples of term operators are `if` and `quote`.

Any symbol that is neither a variable nor an operator is a *constant* in KIF. Both `A36`

[1]Symbols whose first character is a $ sign are called *individual variables* in KIF. KIF also has what are known as *sequence variables*—they are denoted by symbols that begin with an @ sign.

and 8 are examples of constants. There are certain *basic* (or predefined) constants in KIF. These include:

- function constants such as `*`, `sin`, `ceiling`, and `min`;
- relation constants such as `>=`, `<-`, and `integerp`; and
- logical constants such as `true` and `false`.

With this vocabulary, one can formulate terms in a manner analogous to Section A.2.1. However, the syntax for functional terms differs in one respect: instead of the function name followed by list of arguments (separated by commas) in parentheses, functional terms in KIF are lists whose first member is a function constant and other members are arguments separated by spaces. Thus, if `conc-volume` is a function of the deck depth and concrete depth, we can formulate an example term as `(conc-volume $deck-depth $conc-depth)`. Some other sample terms are: `A50`, `$direction`, and `(phi bending composite)`. KIF also provides an additional type of terms known as *special terms*. The following simple conditional, where ϕ is a sentence and τ_1 and τ_2 are terms is an example.

$$(\texttt{if}\ \phi\ \tau_1\ \tau_2)$$

(The value of the above conditional is the value of τ_1 if ϕ is true; otherwise it is the value of τ_2.)

Formulation of sentences in KIF is also similar to that described in Section A.2.1 except for differences in a few details. For one, the relation's name goes within the parentheses along with the arguments (which are separated by spaces instead of commas). Thus, `(primary-behavior column axial-compression)` is a syntactically acceptable KIF sentence. (Here `primary-behavior` is a relation that holds between two object constants, namely, `column` and `axial-compression`.) To illustrate sentence formulation with another example, recall that the yield strength, f_y, of A36 steel is 36 ksi. Ordinarily we write this as:

$$f_y = 36 \text{ ksi}.$$

To be more precise, f_y is a function of the grade of steel, and

$$f_y(\text{A36}) = 36 \text{ ksi}.$$

We can convert this into KIF syntax as

```
(= (fy A36) 36)
```

Note that the representation is completely declarative and the statements do not have any control information embedded in them.

Similar to the `=` relation, KIF has another basic relation `<-` which is used for writing *term simplification* rules. Thus, a rule of the form `(<-` τ_1 τ_2`)` means that instances of the term τ_1 can be replaced by the term τ_2: This can be quite useful as the following simple rules show. The first one declaratively states that a quantity can be converted from feet to inches by multiplying it by 12. The next rule states that area of a circle of diameter `$dia` is π `$dia`$^2/4$. (`pi` is a basic constant in KIF.) The last rule states that perpendicular of `XX` is `YY`.

```
(<- (ft-to-in $qty) (* $qty 12))
(<- (A $dia) (/ (* pi $dia $dia) 4))
(<- (perpendicular XX) YY)
```

Proceeding to more complex examples, consider the case of estimating cost of cambering of linear horizontal elements (lhe's), such as beams and girders, in a floor. We can express the relationship for cost as shown here.

$$\text{Cost(Cambering)} = \begin{cases} \text{Unit-Price(Cambering)} * \text{Total-Pieces(lhe)}, & \text{if lhe's cambered;} \\ 0, & \text{otherwise.} \end{cases}$$

This relationship can be expressed in KIF syntax as follows.

```
(<- (cost cambering)
    (if (known (is (lhe-cambering) cambered))
        (* (unit-price cambering) (total-pieces lhe))
        0))
```

We conclude this appendix by giving two axioms from FFG's knowledge base that are used for reasoning based on function.

```
(=> (building-occupancy commercial)
    (load vertical-distributed))

(=> (need-to-perform $function)      ; If a function has to be performed,
    (function $e $function)          ; and there exists an element which performs that fn,
    (not (provable (unusable $e)))   ; and it can't be proved that the element is unusable,
    (usable-material $e $matl)       ; and there is a usable material for the problem,
    (possible-material $e $matl)     ; from which it is possible to construct the element,
    (use $e))                        ; then use the element.
```

But surpassing all stupendous inventions, what sublimity of mind was his who dreamed of finding means to communicate his deepest thoughts to any other person, though distant by mighty intervals of place and time! Of talking with those who are in India; of speaking to those who are not yet born and will not be born for a thousand or ten thousand years; and with what facility, by the different arrangements of twenty characters upon a page!

— *Galileo Galelei,* Dialogue Concerning the Two Chief World Systems *(translated by Stillman Drake) (1632)*

A given representation language can be implemented in all manner of ways: predicate calculus assertions may be implemented as lists, as character sequences, as trees, as networks, as patterns in associative memory, etc.: all giving different computational properties but all encoding the same representational language. . . . Similarly, any one of these computational techniques can be used to implement many essentially different representational languages. Thus, circuit diagrams, perspective line drawings, and predicate calculus assertions, three entirely distinct formal languages, can be all implemented in terms of list structures. Were it not so, every application of computers would require the development of a new specialised programming language.

— *P. J. Hayes, in* Frame Conceptions and Text Understanding *(1980)*

C
System Details

This appendix presents low-level details of Galileo not described elsewhere in this dissertation. These include particulars of data organization, input format, interface, and reasoning. The details and the rationale behind them are the subject of discussion for the remainder of this appendix.

C.1 Data Organization

Galileo needs to keep track of two types of data: (i) problem-independent data and (ii) problem-dependent data. Examples of the former include information about properties of noncomposite and composite steel sections, whereas examples of the latter include information about the topology and design of members of generated solutions. The data is organized in terms of predefined predicates that have fixed associated arities. Conceptually, this amounts to what can be termed as a database schema. Details of the schema, for both problem-independent as well as problem-dependent data, follow in subsequent subsections. (The reader is referred to Appendix B for a description of the KIF syntax.)

C.1.1 Problem-Independent Data

For load and resistance factor design, properties such as resistance capacities and moment of inertia of different sections are needed. In the case of composite sections, the properties depend upon the location of the plastic neutral axis (PNA) and thickness of the concrete slab, as measured by two parameters `Y1` and `Y2` defined in Part 4 of the LRFD manual. Accordingly, Galileo stores data about noncomposite and composite sections in the following format.

```
(noncomposite <shape> <depth> <wt> <A36-mom-capacity> <A50-mom-capacity>
              <Ixx>)
(composite-A36 <shape> <depth> <wt> <Y1> <sigmaQn> A36 moment capacity for
               <Y2 = 4.0> <y2 = 4.25> <Y2 = 4.5> <Y2 = 5.0> <Y2 = 5.25>
               <Y2 = 5.5> <Y2 = 6.0> <Y2 = 6.5> <Y2 = 7.0>)
(composite-A50 <shape> <depth> <wt> <Y1> <sigmaQn> A50 moment capacity for
               <Y2 = 4.0> <Y2 = 4.25> <Y2 = 4.5> <Y2 = 5.0> <Y2 = 5.25>
               <Y2 = 5.5> <Y2 = 6.0> <Y2 = 6.5> <Y2 = 7.0>)
(composite-Ilb <shape> <depth> <wt> <Y1> Lower bound moment of inertia for
               <Y2 = 4.0> <Y2 = 4.25> <Y2 = 4.5> <Y2 = 5.0> <Y2 = 5.25>
               <Y2 = 5.5> <Y2 = 6.0> <Y2 = 6.5> <Y2 = 7.0>)
```

where

sigmaQn is the horizontal shear force at the interface between the steel section and the concrete slab,
Y1 is the distance from PNA to beam top flange, and
Y2 is the distance from concrete flange force to beam top flange.

(Please see Part 4 of the LRFD manual for more details.) Additionally, the shear and axial capacities of noncomposite sections are stored in the form shown here.

```
(shear-capacity <shape> <depth> <wt> <A36-shear-capacity>
                <A50-shear-capacity>)
(axial-capacity <shape> <depth> <wt> <eff-length> <A36-axial-capacity>
                <A50-axial-capacity>)
```

Given the section properties and steel grade, shear capacity of a section can be determined. Similarly, axial capacity of a given section of a particular steel grade is a function of its effective length. Tables of these capacities can be prepared beforehand as a function of relevant variables (they can also be extracted from the LRFD manual) and stored electronically to save run time during actual design. In the case of Galileo, shear and axial capacities are used for design of beams and columns, respectively, through comparison with the pertinent demand.

Data about the metal decks is represented in the following format.

```
(unshored-clear-span <deck-depth> <gauge> <conc-type> <conc-depth>
                     <span>)
```

```
(= (conc-volume <deck-depth> <conc-depth>) <volume>)
```

Properties of the decks are typically manufacturer-dependent; the ones in Galileo correspond to the decks produced by Vulcraft. The representation, however, would be applicable to decks produced by any manufacturer. In order to design using decks produced by some other manufacturer, one only has to replace the current deck properties database in Galileo with one corresponding to the chosen manufacturer.

C.1.2 Problem-Dependent Data

Framing plans typically contain a few different kinds of elements that are repeated several times (e.g., see Fig. 7.1). Thus, there may be several instances of a particular *type* of beam in a plan, all identical in every respect except location. Data which is the same for all such beams (or, in other words, is independent of the location) is stored once, defining a designation. In addition, a fact corresponding to each instance of the designation is also stored. This is illustrated through examples in the passages that follow.

Beams, girders, etc., in the floor are collectively referred to as linear horizontal elements (or lhe's for short) in Galileo. This is to facilitate the same code to perform operations like sizing on all one-dimensional bending elements irrespective of their kind. Accordingly, the representation of lhe's in Galileo is as follows.

```
(lhe-desig <type> <id> <where> <span> <spcg1> <spcg2>)
```

In this template, `<type>` can be one of (i) `beam`, (ii) `girder`, (iii) `core-lhe`, and (iv) `redundant-lhe` (see Chapter 7 for examples of each type). Although core linear horizontal elements can also be classified as beams and girders, they are assigned a separate category to underscore the tentative nature of their design. This is because loads due to the service core are not considered in the design of members for gravity loading. The `<id>` argument is a name tag to label the element. Although Galileo assigns unique tags to all the elements, the representation and reasoning in the system is not dependent on this property. The field `<where>` can take either `exterior` or `interior` as a value. (Design considerations are different for the two; e.g., cladding loads are applicable only to exterior lhe's.) Finally, `<span>` is the span of the lhe whereas `<spcg1>` and `<spcg2>` are the respective spacings on either side.

Individual instances of the lhe's are represented as follows.

```
(beam <id> <direction> <level> <start-coord> <end-coord>)
```

```
(girder <id> <direction> <level> <start-coord> <end-coord>)
(core-lhe <id> <direction> <level> <start-coord> <end-coord>)
(redundant-lhe <id> <direction> <level> <start-coord> <end-coord>)
```

Since Galileo considers only orthogonal lhe's, `<dir>` can be either `XX` or `YY`. If the direction is `XX`, three values—namely, the x coordinates of the start and end points and the y coordinate (i.e., the `<level>`) of either point—are sufficient to describe the location of the lhe. Alternatively, if the direction is `YY`, y coordinates of the start and end points will be relevant and the `<level>` will correspond to x coordinate.

The organization of data about columns is similar and self-explanatory.

```
(column-desig <id> <where>)
(column <id> <x> <y>)
```

The design information of the members is represented in the following manner.

```
(lhe-section <loading> <id> <story> <shape> <depth> <wt>)
(column-section <loading> <id> <starting-story> <ending-story>
                <shape> <depth> <wt>)
```

In this representation, `<loading>` can be either `gravity` or `total`. (Recall that Galileo sizes members purely for gravity loads also to estimate the overhead for carrying the lateral loads.) In the case of columns, which may be spliced every second or so story, the starting story and the end story in which a particular section is used are recorded. For lhe's, if `<story>` is a variable (e.g., `$n`), it will imply that the section for that lhe designation is the same for all stories since any story can unify with the variable. Alternatively, one fact per story per designation will be stored. Section properties `<shape>`, `<depth>`, and `<wt>` are self-evident. For example, for a W14 $\times$ 34 section, the `<shape>`, `<depth>`, and `<wt>` will be W, 14, and 34 respectively. For composite lhe's, the number of shear studs also needs to be stored. Statements of the following form are used for the purpose.

```
(shear-studs <lhe-id> <no-of-studs>)
```

Besides the generated data, input to Galileo is also problem-dependent. The input largely consists of the description of the geometry of the plan. Since entities in the plan are restricted to be rectangular, coordinates of any two opposite corners are sufficient to express their location. Thus, most input facts contain a system-recognized relation (i.e., a keyword) followed by two sets of coordinates. A list of system recognized relations with their arguments follows.

```
(perimeter <x1> <y1> <x2> <y2>)
(core <x1> <y1> <x2> <y2>)
(lobby <x1> <y1> <x2> <y2>)
(staircase <x1> <y1> <x2> <y2>)
(restroom <x1> <y1> <x2> <y2>)
(shaft <x1> <y1> <x2> <y2>)
(hallway <x1> <y1> <x2> <y2>)
(misc <x1> <y1> <x2> <y2>)
(elevator-bank <x1> <y1> <x2> <y2> <n>)
```

The last relation in the preceding list, `elevator-bank`, takes an additional argument that specifies the number of elevators in the bank.

C.2 Parameters

As stated in Chapter 6, the knowledge in Galileo is structured in terms of certain parameters. They are:

- Planning Module,
- Minimum Office Space,
- Minimum Beam Spacing,
- Maximum Beam Spacing,
- Minimum Beam Span,
- Maximum Beam Span,
- Minimum Girder Span, and
- Maximum Girder Span.

Default values of all these parameters are provided. The values for planning module, minimum beam spacing, and maximum beam spacing are shown in Table C.1. Default values of other parameters depend on the value of the planning module and are taken to be an integral multiple of the planning module closest to a specified value as shown in Table C.2.

When the user alters any of the default values, certain checks are applied on the new value. First is a syntactic check, which ensures that the new value is a positive number. Semantic checks ensure that the values are consistent; e.g., the entered value of maximum

Parameter	*Default Value (ft.)*
Planning Module	5
Minimum Beam Spacing	8
Maximum Beam Spacing	12

Table C.1: Default Values of Some Parameters

Parameter	*Default Value (ft.) is a Multiple of the Planning Module Closest to*
Minimum Office Space	20
Minimum Beam Span	10
Maximum Beam Span	45
Minimum Girder Span	15
Maximum Girder Span	45

Table C.2: Default Values of Planning Module-Dependent Parameters

beam spacing should not be less than the minimum beam spacing. Plausibility checks see if the entered value is plausible. For example, the user-defined values of parameters mentioned in Table C.2 should preferably be integral multiples of planning module. In a similar vein, a value outside the range of 3–12 ft. for planning module is implausible. Violation of syntactic and semantic checks results in an error and nonacceptance of the offered value. If a plausibility check is not satisfied, a warning is issued; however, the offered value is accepted.

C.3 Options

Galileo provides several different options to the user as briefly illustrated earlier in this dissertation. A comprehensive account of the various options is given in Tables C.3–C.11.

Item	*Legal Values*	*Default Value*
Concrete Type	One of ○ Lightweight ○ Normalweight ● Explore Both	Explore Both
f'_c	Any positive number	4 ksi
Mechanical Duct Depth	Any nonnegative number	14 in.
Light Fixtures Depth	Any nonnegative number	7 in.
Floor Finish Depth	Any nonnegative number	1 in.

Table C.3: Floor Design Options

Item	*Legal Values*	*Default Value*	*Remarks*
Steel Grade	One of ○ A36 ○ A50 ● Explore Both	Explore Both	
Behavior	One of ○ Noncomposite ● Composite	Composite	Shear studs and shoring options are disabled if behavior is chosen to be noncomposite.
Shear Studs	One of ○ $1/2'' \times 5''$ ● $3/4'' \times 5''$	$3/4'' \times 5''$	
Shoring	Yes or No	No	
Cambering	Yes or No	Yes	
Deflection Factor (α)	Any positive number	360	$\delta(\text{liveload}) \leq \frac{l}{\alpha}$
Minimum Beam Depth	Any positive number less than or equal to Maximum Beam Depth and greater than or equal to 6 in.	6 in.	
Maximum Beam Depth	Any positive number greater than or equal to Minimum Beam Depth and less than or equal to 36 in.	36 in.	
Minimum Girder Depth	Any positive number less than or equal to Maximum Girder Depth and greater than or equal to 6 in.	6 in.	
Maximum Girder Depth	Any positive number greater than or equal to Minimum Girder Depth and less than or equal to 36 in.	36 in.	

Table C.4: LHE Design Options

Item	*Legal Values*	*Default Value*	*Remarks*
Steel Grade	One of o A36 o A50 ● Explore Both	Explore Both	
Strategy	One of o Minimum Weight ● Constant Depth	Constant Depth with $d = 14$ in.	d can be set by the user. Plausible value for d is 14 in.
Effective Length Factor (Gravity)	Any positive number	1.0	Plausible range is 0.5–5.0
Splicing Frequency	Any positive integer	2 stories	Plausible values are 1, 2, and 3 stories.

Table C.5: Column Design Options

Item	*Legal Values*	*Default Value*
Office Live Load	Any positive number	50 psf
Partitions	Any positive number	20 psf
Ceiling	Any positive number	10 psf
Cladding	Any positive number	25 psf
Lightweight Concrete Unit Weight	Any positive number	110 psf
Normalweight Concrete Unit Weight	Any positive number	145 psf
Construction Live Load	Any positive number	20 psf

Table C.6: Gravity Loading Options

Item	*Legal Values*	*Default Value*	*Remarks*
Seismic Zone	One of ○ Zone 0 ○ Zone 1 ○ Zone 2 ○ Zone 3 ● Zone 4	Zone 4	1988 UBC [40]
Soil Profile	One of ○ S1 ● S2 ○ S3 ○ S4	S2	
Importance Factor (I)	Any positive number	1.0	Plausible value is 1.0
Fundamental Period (T)	One of ● $C_t h_n^{3/4}$ ○ User-supplied value	$T = C_t h_n^{3/4}$	$C_t = 0.035$ for moment resisting frames. Plausible value of T is less than or equal to 10 sec.

Table C.7: Seismic Loading Options

Item	*Legal Values*	*Default Value*	*Remarks*
Exposure	One of ● B—Irregular Terrain ○ C—Flat and Open Terrain	B	
V_{30}	Any positive number	70 mph	Plausible value is below 200 mph

Table C.8: Wind Loading Options

Item	*Legal Values*	*Default Value*
Steel—A36	Any positive number	525 \$/ton
Steel—A50	Any positive number	555 \$/ton
Concrete—Lightweight	Any positive number	80 \$/cu. yd.
Concrete—Normalweight	Any positive number	60 \$/cu. yd.
Metal Deck—2" deep	Any positive number	1.30 \$/sq. ft.
Metal Deck—3" deep	Any positive number	1.50 \$/sq. ft.
Shear Studs—1/2″ × 5″	Any positive number	1.20 \$/ea.
Shear Studs—3/4″ × 5″	Any positive number	1.50 \$/ea.
Welded Wire Fabric	Any positive number	0.10 \$/sq. ft.
Fabrication—Gravity LHE	Any positive number	100 \$/ea.
Fabrication—Gravity Column	Any positive number	650 \$/ea.
Fabrication—Frame LHE	Any positive number	150 \$/ea.
Fabrication—Frame Column	Any positive number	1000 \$/ea.
Erection	Any positive number	560 \$/person-day
Cambering	Any positive number	30 \$/ea.
Shoring	Any positive number	0.25 \$/sq. ft.
Fire Proofing	Any positive number	0.50 \$/sq. ft.
Transportation	Any positive number	10 \$/ton

Table C.9: Price Options

Item	*Legal Values*	*Default Value*
Average Column Stress—A36	Any positive number	19 ksi
Average Column Stress—A50	Any positive number	26 ksi
These stresses are used only for estimating savings in column material when comparing lightweight and normalweight concretes.		

Table C.10: Stress Options

Project Size (tons)	*Item*	*Legal Values*	*Default Value*
0–1000	Erectable Pieces/Day	Any positive integer	65
	Crew Size	Any positive integer	23
1001–4000	Erectable Pieces/Day	Any positive integer	60
	Crew Size	Any positive integer	26
4001–10000	Erectable Pieces/Day	Any positive integer	55
	Crew Size	Any positive integer	29
> 10000	Erectable Pieces/Day	Any positive integer	50
	Crew Size	Any positive integer	32

Table C.11: Erection Options

C.4 Cost Estimation

Cost of various items is computed through the following formulas.

$$\begin{aligned}
\text{Deck cost} &= \text{Price per sq. ft. of the deck (for the chosen depth and gauge)}\\
&\quad \times \text{Deck area}\\
\text{Concrete cost} &= \text{Price per cu. yd. of concrete (for the chosen type)}\\
&\quad \times \text{Concrete volume}\\
\text{Shear studs cost} &= \begin{cases} \text{Unit price} \times \#\text{studs}, & \text{if lhe's are composite;}\\ 0, & \text{if lhe's are noncomposite.}\end{cases}\\
\text{Welded wire}&\\
\text{fabric cost} &= \text{Price per sq. ft. (for the chosen type)} \times \text{Deck area}\\
\text{Steel cost} &= \text{Price per ton of steel (for the chosen grade)} \times \text{Steel tonnage}\\
\text{Fabrication cost}&\\
\text{for gravity system} &= (\text{Fabrication price for gravity support columns}\\
&\quad \times \text{Number of gravity support columns})\\
&\quad + (\text{Fabrication price for gravity support lhe's}\\
&\quad \times \text{Number of gravity support lhe's})
\end{aligned}$$

$$\begin{aligned}
\text{Fabrication cost} & \\
\text{for total system} &= (\text{Fabrication price for gravity support columns} \\
&\quad \times \text{Number of gravity support columns}) \\
&\quad + (\text{Fabrication price for frame columns} \\
&\quad \times \text{Number of frame columns}) \\
&\quad + (\text{Fabrication price for gravity support lhe}'\text{s} \\
&\quad \times \text{Number of gravity support lhe}'\text{s}) \\
&\quad + (\text{Fabrication price for frame lhe}'\text{s} \\
&\quad \times \text{Number of frame lhe}'\text{s})
\end{aligned}$$

$$\begin{aligned}
\text{Erection cost} &= \frac{\text{Total number of pieces to be erected}}{\text{Erectable pieces per day (for the project size)}} \\
&\quad \times \text{Persons required per day (for the project size)} \\
&\quad \times \text{Labor price per person-hour}
\end{aligned}$$

$$\begin{aligned}
\text{Shoring cost} &= \begin{cases} \text{Price per sq. ft.} \times \text{Deck area,} & \text{if lhe's are shored;} \\ 0, & \text{if lhe's are unshored.} \end{cases} \\
\text{Cambering cost} &= \begin{cases} \text{Unit price} \times \#\text{lhe}'\text{s to be cambered,} & \text{if lhe's are cambered;} \\ 0, & \text{otherwise.} \end{cases} \\
\text{Fire proofing cost} &= \text{Price per sq. ft.} \times \text{Deck area} \\
\text{Transportation cost} &= \text{Price per ton of steel} \times \text{Steel tonnage}
\end{aligned}$$

For every programming language, no matter how clean, elegant, and high level, one can find programmers who will use it to write dirty, contorted, and unreadable programs.

— *Leon Sterling and Ehud Shapiro,* The Art of Prolog *(1986)*

The value of knowledge lies not in its accumulation, but in its utilization.

— *E. Green*

Bibliography

[1] M. H. Ackroyd, S. J. Fenves, and W. McGuire. Computerized LRFD specification. In *Proceedings of the National Steel Construction Conference*, Miami Beach, FA, 1988.

[2] H. Adeli and K. V. Balasubramanyam, editors. *Expert Systems in Construction and Structural Engineering: A New Generation.* Chapman and Hall, New York, NY, 1988.

[3] H. Allison. Steel design—Special considerations. In R. N. White and C. G. Salmon, editors, *Building Structural Design Handbook*, chapter 19, pages 566–632. John Wiley & Sons, New York, NY, 1987.

[4] T. Amble. *Logic Programming and Knowledge Engineering.* Addison-Wesley, Reading, MA, 1987.

[5] American Institute of Steel Construction (AISC), Chicago, IL. *Manual of Steel Construction*, 1980.

[6] American Institute of Steel Construction (AISC), Chicago, IL. *Load and Resistance Factor Design Specification for Structural Steel Buildings*, 1986.

[7] N. C. Baker. Towards a spatial and functional building design system. Technical Report EDRC 02-08-89, Carnegie Mellon University, Pittsburgh, PA, 1989.

[8] J. S. Bennet and R. S. Englemore. SACON: A knowledge-based consultant for structural analysis. In *Proceedings of Sixth International Joint Conference on Artificial Intelligence*, pages 47–49, 1979.

[9] J. A. Bowen, T. C. Cornick, and S. P. Bull. BERT—An expert system for brickwork design. In M.A. Bramer, editor, *Proceedings of Expert Systems '86, the Sixth Annual Technical Conference of the British Computer Society Specialist Group on Expert*

Systems, pages 207–216, 1986.

[10] W.-t. Chan and B. C. Paulson, Jr. Logic programming to manage constraint-based design. *Microcomputer Knowledge-Based Expert Systems in Civil Engineering*, pages 188–202, 1988.

[11] E. Charniak and D. McDermott. *Introduction to Artificial Intelligence.* Addison-Wesley, Reading, MA, 1985.

[12] R. A. Coleman. *Structural System Design.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1983.

[13] Council on Tall Buildings & Urban Habitat, ASCE, New York, NY. *Tall Building Systems and Concepts*, 1978.

[14] R. Coyne. *Logic Models of Design.* Pitman, London, UK, 1988.

[15] R. Davis. Form and content in model based reasoning. In E. Scarl, editor, *Proceedings, Workshop on Model-Based Reasoning* (held in conjunction with the Eleventh Joint Conference on Artificial Intelligence), pages 11–27, 1989.

[16] C. L. Dym and R. E. Levitt. *Knowledge-Based Systems in Engineering.* McGraw-Hill, Inc., New York, NY, 1991.

[17] Epistemics, Inc., Palo Alto, CA. *The Epikit Manual*, 1988.

[18] S. J. Fenves. Computer applications in structural engineering. In E. H. Gaylord, Jr. and C. N. Gaylord, editors, *Structural Engineering Handbook*, pages 2–1 – 2–17. McGraw Hill, New York, NY, 2nd edition, 1979.

[19] S. J. Fenves, E. H. Gaylord, and S. K. Goel. Decision table formulation of the 1969 AISC specification. Civil Engineering Studies SRS 347, Department of Civil Engineering, University of Illinois, Urbana, IL, 1969.

[20] S. J. Fenves and R. N. Wright. The representation and use of design specifications. Technical Note 940, National Bureau of Standards, 1977.

[21] S. J. Fenves, U. Flemming, C. Hendrickson, M. L. Maher, and G. Schmitt. A prototype environment for integrated design and construction planning of buildings. In *Proceedings of the First Symposium of the Center for Integrated Facility Engineering*, Stanford University, Stanford, CA, 1989.

[22] M. Fischer. *Constructibility Improvement during Preliminary Design of Reinforced Concrete Structures.* PhD thesis, Department of Civil Engineering, Stanford University, Stanford, CA, (under preparation).

[23] J. H. Garrett, Jr. and S. J. Fenves. A knowledge-based standards processor for structural component design. *Engineering With Computers*, 2(4):219–238, 1987.

[24] M. R. Genesereth. An overview of meta-level architecture. In *Proceedings of the National Conference on Artificial Intelligence*, pages 119–124, AAAI, Washington, DC, 1983.

[25] M. R. Genesereth. The Designworld project. Technical Report Logic-89-3, Logic Group, Computer Science Department, Stanford University, Stanford, CA, 1989.

[26] M. R. Genesereth and N. J. Nilsson. *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann, Los Altos, CA, 1987.

[27] M. R. Genesereth, R. Letsinger, and N. P. Singh. Basic Knowledge Representation. Technical Report Logic-89-15, Logic Group, Department of Computer Science, Stanford University, Stanford, CA, 1990.

[28] J. S. Gero. Design prototypes: A knowledge representation schema for design. *AI Magazine*, 11(4):26–36, 1990.

[29] J. S. Gero, M. L. Maher, and W. Zhang. Chunking structural design knowledge as prototypes. In J. S. Gero, editor, *Artificial Intelligence in Engineering: Design*, pages 3–21. Computational Mechanics Publications, Southampton, UK, 1988.

[30] C. Green. Application of theorem proving to problem solving. In *Proceedings of the First International Joint Conference on Artificial Intelligence*, pages 219–239, Washington, DC, 1969.

[31] T. Gruber and Y. Iwasaki. How things work: Knowledge-based modeling of physical devices. Technical Report KSL 90-51, Knowledge Systems Laboratory, Computer Science Department, Stanford University, Stanford, CA, 1990.

[32] R. V. Guha and D. B. Lenat. Cyc: A midterm report. *AI Magazine*, 11(3):32–59, 1990.

[33] D. Haber and S. Karshenas. CONCEPTUAL: An expert system for conceptual structural design. *Microcomputers in Civil Engineering*, 5(2):119–127, 1990.

[34] F. Hart, W. Henn, and H. Sontag. *Multi-Storey Buildings in Steel.* Nichols Publishing Company, New York, NY, 1985.

[35] P. J. Hayes. In defence of logic. In *Proceedings of Fifth International Joint Conference on Artificial Intelligence*, Cambridge, MA, 1977.

[36] P. J. Hayes. The logic of frames. In D. Metzing, editor, *Frame Conceptions and Text Understanding*, pages 46–61. de Gruyter, Berlin, 1980.

[37] H. C. Howard, J. Wang, F. Daube, and T. Rafiq. Applying design-dependent knowledge in structural engineering design. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 3(2):111–123, 1989.

[38] R. B. Hurley. *Decision Tables in Software Engineering.* Van Nostrand Reinhold Company, Inc., New York, NY, 1983.

[39] P. J. Hutchinson. An expert system for the selection of earth retaining structures. Master's thesis, Department of Architectural Science, University of Sydney, Sydney, Australia, 1985.

[40] International Conference of Building Officials, Whittier, CA. *Uniform Building Code Standards*, 1988.

[41] S. H. Iyengar and M. Iqbal. Composite construction. In R. N. White and C. G. Salmon, editors, *Building Structural Design Handbook*, chapter 23, pages 787–820. John Wiley & Sons, New York, NY, 1987.

[42] D. Jain and M. L. Maher. Combining expert systems and CAD techniques. *Microcomputers in Civil Engineering*, 3(4):321–331, 1988. (Also in Gero, J. and Stanton, R., editors, *Artificial Intelligence Developments and Applications*, Elsevier Science Publishers, pp. 65–81, 1988).

[43] D. Jain, K. H. Law, and H. Krawinkler. Knowledge Representation with Logic. Technical Report 13, Center for Integrated Facility Engineering (CIFE), Stanford University, Stanford, CA, 1989.

[44] D. Jain, K. H. Law, and H. Krawinkler. On processing standards with predicate calculus. In *Proceedings of the Sixth Conference on Computing in Civil Engineering*, pages 259–266, Atlanta, GA, 1989. ASCE.

[45] P. Jayachandran and N. Tsapatsaris. A knowledge based expert system for the selection of structural systems for tall buildings. *Microcomputer Knowledge-Based Expert Systems in Civil Engineering*, pages 88–99, 1988.

[46] A. Karakatsanis. FLODER: A floor designer expert system. Master's thesis, Department of Civil Engineering, Carnegie-Mellon University, Pittsburgh, PA, 1985.

[47] S. H. Kim and N. P. Suh. Application of symbolic logic to the design axioms. *Robotics and Computer-Integrated Manufacturing*, 2(1):55–64, 1985.

[48] R. A. Kowalski and K. A. Bowen, editors. *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, volume 1 and 2, University of Washington, Seattle, 1988. The MIT Press, Cambridge, MA.

[49] B. Kumar and B. H. V. Topping. An integrated rule-based system for industrial building design. *Microcomputer Knowledge-Based Expert Systems in Civil Engineering*, pages 53–72, 1988.

[50] J. C. Kunz, M. J. Stelzner, and M. D. Williams. From classic expert systems to models: Introduction to a methodology for building model-based systems. In G. Guida and C. Tasso, editors, *Topics in Expert Systems Design: Methodologies and Tools*, pages 87–110. Elsevier Science Publishers B.V. (North-Holland), New York, NY, 1989.

[51] S. Lakmazaheri. *A Study on the Constraint Logic Approach for Structural Design Automation*. PhD thesis, Department of Civil Engineering, North Carolina State University, Raleigh, NC, 1990.

[52] J. Lassez, editor. *Proceedings of the Fourth International Conference on Logic Programming*, volume 1 and 2, University of Melbourne, Australia, 1987. The MIT Press, Cambridge, MA.

[53] D. B. Lenat and E. A. Feigenbaum. On the thresholds of knowledge. In *Proceedings of Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, 1987.

[54] R. E. Levitt. HOWSAFE: A microcomputer-based expert system to evaluate the safety of a construction firm. In C. N. Kostem and M. L. Maher, editors, *Expert Systems in Civil Engineering*, pages 55–66. ASCE, 1986.

[55] V. Lifschitz. Computing circumscription. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, 1985.

[56] T. Y. Lin and S. D. Stotesbury. *Structural Concepts and Systems for Architects and Engineers.* John Wiley & Sons, Inc., New York, NY, 1981.

[57] G. P. Luth. *Representation and Reasoning for Integrated Structural Design of High-Rise Commercial Office Buildings.* PhD thesis, Department of Civil Engineering, Stanford University, Stanford, CA, (under preparation).

[58] G. P. Luth, D. Jain, H. Krawinkler, and K. H. Law. A formal approach to automating conceptual structural design: Part I—Methodology. To appear in *Engineering With Computers*, 1991.

[59] J. D. Mackinlay. *Automatic Design of Graphical Presentations.* PhD thesis, Department of Computer Science, Stanford University, Stanford, CA, 1987.

[60] M. L. Maher and S. J. Fenves. HI-RISE: A knowledge-based expert system for the preliminary structural design of high rise buildings. Technical Report R-85-146, Department of Civil Engineering, Carnegie-Mellon University, Pittsburgh, PA, 1985.

[61] Z. Manna and R. Waldinger. *The Logical Basis for Computer Programming*, volume 1. Addison Wesley, Menlo Park, CA, 1985.

[62] Z. Manna and R. Waldinger. *The Logical Basis for Computer Programming*, volume 2. Addison Wesley, Menlo Park, CA, 1990.

[63] J. McCarthy. Circumscription—A form of non-monotonic reasoning. *Artificial Intelligence*, 13(1–2):27–39, 1980.

[64] D. McDermott. A critique of pure reason. *Computational Intelligence*, 3(3):151–160, 1987.

[65] D. McDermott and J. Doyle. Non-monotonic logic I. *Artificial Intelligence*, 13(1–2):41–72, 1980.

[66] R. C. Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25(1):75–94, 1985.

[67] R. C. Moore. The role of logic in knowledge representation and commonsense reasoning. In H. Levesque and R. Brachman, editors, *Readings in Knowledge Representation*, pages 335–343. Morgan Kaufmann, San Mateo, CA, 1986.

[68] M. Musen. Automated generation of model–based knowledge–acquisition tools. SIGLunch Presentation, Stanford University, Stanford, CA, December 1989.

[69] N. J. Nilsson. Logic and artificial intelligence. In *Proceedings of MIT Workshop on Artificial Intelligence*, Cambridge, MA, (to appear).

[70] H. Pospesel. *Introduction to Logic: Propositional Calculus.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1974.

[71] H. Pospesel. *Introduction to Logic: Predicate Calculus.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1976.

[72] W. J. Rasdorf and T. E. Wang. Generic design standards processing in an expert system environment. *Journal of Computing in Civil Engineering*, 2(1):68–87, 1988.

[73] W. J. Rasdorf and S. Lakmazaheri. A logic-based approach for processing design standards. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, (to appear).

[74] D. R. Rehak and L. A. Lopez. Computer-aided engineering: Problems and prospects. Civil Engineering Systems Laboratory Research Series (CESLRS) 8, Department of Civil Engineering, University of Illinois, Urbana-Champaign, IL, 1981.

[75] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1–2):81–132, 1980.

[76] T. J. Ross, H. C. Sorensen, S. J. Savage, and J. M. Carson. DAPS: Expert system for structural damage assessment. *Journal of Computing in Civil Engineering*, 4(4):327–348, 1990.

[77] R. Sause and G. H. Powell. A design process model for computer integrated structural engineering. *Engineering with Computers*, 6(3):129–143, 1990.

[78] E. Scarl, editor. *Proceedings, Workshop on Model-Based Reasoning* (held in conjunction with the Eleventh Joint Conference on Artificial Intelligence), AAAI, Detroit, MI, 1989.

[79] D. L. Schodek. *Structures.* Prentice-Hall Inc., Englewood Cliffs, NJ, 1980.

[80] D. Sriram. *Knowledge-Based Approaches for Structural Design.* PhD thesis, Department of Civil Engineering, Carnegie-Mellon University, Pittsburgh, PA, 1986.

[81] D. Sriram, G. Stephanopoulos, R. Logcher, D. Gossard, N. Groleau, D. Serrano, and D. Navinchandra. Knowledge-based systems applications in engineering design: Research at MIT. *AI Magazine*, 10(3):79–96, 1989.

[82] F. I. Stahl, R. N. Wright, S. J. Fenves, and J. R. Harris. Expressing standards for computer-aided building design. *Computer-Aided Design*, 15(6):329–334, 1983.

[83] M. Stefik. *Introduction to Knowledge Systems.* Xerox Palo Alto Research Center, Palo Alto, CA, (under preparation).

[84] B. S. Taranath. *Structural Analysis & Design of Tall Buildings.* McGraw-Hill, New York, NY, 1988.

[85] J. A. Venegas Pabon. *A Model for Design/Construction Integration During the Initial Phases of Design for Building Construction Projects.* PhD thesis, Department of Civil Engineering, Stanford University, Stanford, CA, 1987.

[86] B. L. Webber and N. J. Nilsson, editors. *Readings in Artificial Intelligence.* Tioga Publishing Company, Palo Alto, CA, 1981.

[87] R. Weyhrauch. Prolegomena to a theory of mechanized formal reasoning. *Artificial Intelligence*, 13(1–2):133–170, 1980.

[88] G. Winter. Cold-formed steel construction. In R. N. White and C. G. Salmon, editors, *Building Structural Design Handbook*, chapter 20, pages 633–659. John Wiley & Sons, New York, NY, 1987.

[89] F. Zhao and M. L. Maher. Using analogical reasoning to design buildings. *Engineering With Computers*, 4(3):107–119, 1988.

This dissertation was produced using the $\LaTeX$ macro package of Leslie Lamport under the $\TeX$ typesetting system developed by Donald E. Knuth. The $\TeX$tures^{TM} implementation of these systems for the Macintosh computer was employed. The *Computer Modern* family of fonts, generated in METAFONT by Donald Knuth was used. Customized $\LaTeX$ `.sty` files, based on the `suthesis` style, were used for the design of the format of this dissertation.