Technical Report 1043

Computation and Pre-Parametric Design

Karl T. Ulrich

MIT Artificial Intelligence Laboratory

Computation and Pre-Parametric Design

by

Karl Thatcher Ulrich

Submitted to the Department of Mechanical Engineering on July 29, 1988 in partial fulfillment of the requirements for the degree of Doctor of Science in Mechanical Engineering.

Abstract

My work is broadly concerned with the question *How can designs be synthesized computationally?* The project deals primarily with mechanical devices, and focuses on *pre-parametric design*: design at the level of detail of a blackboard sketch rather than at the level of detail of an engineering drawing. The primary goal of this research is to develop principles upon which future computational tools for pre-parametric design will be built, by developing research ideas to a level of detail where computer programs can be used as tools for validation. I explore the project ideas in the domain of single-input single-output dynamic systems, like pressure gages, accelerometers, and pneumatic cylinders.

Four ideas are at the core of the design procedures presented in this document: 1) A design problem should be broken into two subproblems— generating schematic descriptions and then generating physical descriptions. This strategy reduces the problem solving complexity and clarifies the problem solving procedures. 2) A formal schematic language is an important hallmark of a schematic synthesis problem. 3) A transformation that compresses many different schematic descriptions into a single minimal representation allows for a concise description of the domain knowledge, and a straightforward schematic synthesis procedure. 4) Efficient designs are ones in which many functions are provided by the same structural element. This property of design is called function sharing, and is a key component of my procedure for generating physical descriptions from schematic descriptions.

Acknowledgments

This technical report is a revised version of my doctoral thesis in mechanical engineering. Many people contributed to the work described in this report. In particular: my thesis advisor, Warren Seering; members of my thesis committee, Randy Davis, Dave Gossard, and Patrick Winston; and fellow students, Mike Caine, Andy Christian, Steve Eppinger, Steve Gordon, Peter Meckl, Ken Pasch, Neil Singer, Erik Vaaler, and Al Ward.

This research was performed at the MIT Artificial Intelligence Laboratory. The laboratory's research is funded in part by the Defense Advanced Research Projects Agency of the United States Department of Defense under contract number N00014-85-K-0124. I received specific support for this project from the National Science Foundation under grant number 8618776-DMC and from an IBM Pre-Doctoral Fellowship for Manufacturing Research.

I would also like to acknowledge that thanks to the faculty, staff, and students, the MIT Artificial Intelligence Laboratory is a very supportive research environment.

Contents

1 Introduction **1**

1.1 PROJECT SUMMARY 1

1.1.1 Motivating Scenario 1

1.1.2 Framework for Project 2

1.1.3 Major Ideas 5

1.2 Goals and Strategy 5

1.2.1 Objectives 5

1.2.2 Motivation 6

1.2.3 Approach 6

1.3 SOME PROPERTIES OF DESIGN 7

1.3.1 Design is a Transformation Between Descriptions 7

1.3.2 Design Is Combinatorial 7

1.3.3 Fundamental Trade-Off Between Expressiveness and Complexity 8

1.3.4 Design is Exactly Solvable at Best, Intractable at Worst . . . 10

1.3.5 Global Optimality is Usually Unrealistic 10

1.4 DESIDERATA FOR DESIGN RESEARCH 11

1.4.1 Definition of the Design Task 11

1.4.2 Representing Specifications and Designs 11

1.4.3 Identifying Techniques for Controlling Complexity 12

1.4.4 Computer Program Demonstration of Results 12

1.5 GUIDE TO THIS DOCUMENT 12

2 The Concept of Schematic Synthesis **15**

2.1 WHAT IS SCHEMATIC SYNTHESIS? 15

2.2 WELL-DEFINED PROBLEMS 16

2.2.1 Formality 16

2.2.2 Problem match 17

2.2.3 Derivability of Behavior 17

2.3 EXAMPLE SYNTHESIS PROBLEMS 18
2.4 SCHEMATIC SYNTHESIS AS FIRST STEP 22
2.4.1 Reduction in Problem Complexity 22
2.4.2 Decoupling Functional and Structural Design Issues 23

3 A Schematic Synthesis Problem and Solution 25
3.1 SUMMARY 25
3.2 DOMAIN DESCRIPTION 26
3.2.1 Single-Input Single-Output Dynamic Systems 27
3.2.2 Example Problems and Solutions 28
3.2.3 Representing Schematic Descriptions 28
3.2.4 Deriving Behavior From Schematic Descriptions 34
3.2.5 Specifying a Problem 36
3.3 SOLUTION TECHNIQUE 40
3.3.1 Generating Candidate Designs 40
3.3.2 Classifying Behavior 41
3.3.3 Modifying Candidate Schematic Descriptions 42
3.4 EXAMPLES 50
3.4.1 Current Meter 51
3.4.2 Speedometer 53
3.4.3 Rate-of-climb indicator 57
3.4.4 Accelerometer 59
3.5 DISCUSSION AND ANALYSIS 61
3.5.1 Utility of the Technique 61
3.5.2 Completeness of the Technique 63
3.5.3 Why the Technique Works 64
3.5.4 Extensibility 66
3.5.5 Lessons from the Dynamic Systems Example 68
3.5.6 Implementation 69

4 Function Sharing to Generate Efficient Physical Descriptions 73
4.1 INTRODUCTION TO FUNCTION SHARING 74
4.1.1 The Concept Of Function Sharing 74
4.1.2 Function Sharing is Important 74
4.1.3 Why Function Sharing is Possible 78
4.1.4 Summary of Function Sharing Procedure 78
4.2 DOMAIN DESCRIPTION AND REPRESENTATION 79
4.2.1 Dynamic Systems 79
4.2.2 Physical Representation 81

4.2.3 Nominal Functional Specification 82
4.2.4 Example Physical Design Description 83
4.2.5 Function Sharing Problem Definition in the Dynamic Systems Domain 87
4.3 FUNCTION-SHARING PROCEDURE 87
4.3.1 Example of Function Sharing Procedure 87
4.3.2 Deleting a Structural Element 88
4.3.3 Recognizing Alternative Features 92
4.3.4 Modifying Alternative Features 96
4.3.5 Multi-Function Elements 96
4.3.6 Control 97
4.4 IMPLEMENTATION AND RESULTS 98
4.4.1 Scope and Goals of the Implementation 98
4.4.2 Three Example Runs 99
4.4.3 Some Program Details 103
4.5 ANALYSIS OF FUNCTION SHARING PROCEDURE 105
4.5.1 Meaning of Physical Descriptions 105
4.5.2 Clobbering Existing Functions 106
4.5.3 Design Knowledge Level 106
4.5.4 Novelty in Mechanical Design 109
4.5.5 Extensibility 111
4.5.6 Miscellaneous Issues 112

5 Discussion **115**
5.1 CONTRIBUTIONS 115
5.1.1 Mechanical Design Concepts Can Be Generated Computationally 116
5.1.2 Separation of Schematic and Physical Descriptions 116
5.1.3 Synthesizing SISO Dynamic Systems 118
5.1.4 Function Sharing Can Be Explained and Automated 118
5.1.5 Novel Designs can be Generated by Unbiased Application of Design Operators 119
5.2 FUTURE WORK 119
5.2.1 Lessons for the Future 120
5.2.2 Future Projects 121

A Literature Review **125**
A.1 DIRECTLY RELEVANT PAPERS 126
A.2 INDIRECTLY RELEVANT PAPERS 130

B Invention as Novel Combination of Existing Device Features **133**
B.1 Summary . . . 133
B.2 INTRODUCTION . . . 134
B.2.1 Function, Structure and Conceptual Design . . . 134
B.2.2 Scope, Objectives and Approach . . . 135
B.3 CENTRAL CONCEPTS . . . 135
B.3.1 Knowledge Representation . . . 136
B.3.2 Design Procedure . . . 138
B.4 AN EXAMPLE . . . 139
B.4.1 The Task . . . 139
B.4.2 System Knowledge . . . 139
B.4.3 Methods . . . 142
B.4.4 Output . . . 144
B.5 DISCUSSION . . . 144
B.5.1 Feedback . . . 148
B.5.2 Conflict Resolution . . . 148
B.5.3 Context . . . 149
B.5.4 Abstraction and Analogy . . . 149
B.6 RELATED WORK . . . 150
B.7 REFERENCES . . . 150

C Achieving Multiple Goals in Conceptual Design **153**
C.1 ABSTRACT . . . 153
C.2 INTRODUCTION . . . 154
C.2.1 Conceptual Design Problems Have Multiple Goals . . . 154
C.2.2 Design And Debug Is One Solution Technique . . . 155
C.3 KEY CONCEPT . . . 156
C.3.1 Perspectives Allow Control of Debugging . . . 156
C.4 AN EXAMPLE . . . 158
C.4.1 A Domain With Two Distinct Design Perspectives . . . 158
C.4.2 Using Perspectives in Design and Debug . . . 159
C.4.3 Initial Design and Abstractions . . . 161
C.4.4 Debugging Procedures . . . 163
C.4.5 Instantiation procedures . . . 164
C.4.6 Evaluation . . . 164
C.4.7 Choosing the Next Design . . . 165
C.5 ANALYSIS . . . 165
C.5.1 This Approach is Successful in the Blocks Domain . . . 166
C.5.2 Perspectives Make Sense as an Organizational Tool for Enforcing Debugging Modularity . . . 166

C.5.3 Criteria For Successful Application To Other Domains 167
C.5.4 The Cost of Using a Non-directed Control Strategy 168
C.6 EXTENSION TO OTHER DOMAINS 169
C.6.1 The Speedometer Revisited 169
C.6.2 Status of Research 170
C.7 RELATED RESEARCH 170
C.8 REFERENCES 172

D What Makes a Mechanical Design Interesting? 173
D.1 ORIGIN OF THESE CLASSIFICATIONS 174
D.2 FOUR CATEGORIES OF INTERESTING DESIGNS 174
D.2.1 Novel Combination 175
D.2.2 Geometrical Exploitation 178
D.2.3 Application of Natural Phenomena 181
D.2.4 Function Sharing 183
D.3 USING THESE IDEAS 185
D.3.1 Teaching Design 185
D.3.2 Research Directions 186

List of Figures

1.1 Scenario illustrating the design of an accelerometer 3
1.2 Problem decomposition 4
1.3 Trade-off between expressiveness and computational complexity . . . 9

2.1 Schematic description of a process plant. 19
2.2 Schematic description of an analog circuit 20
2.3 Schematic description of a kinematic linkage. 21

3.1 Example SISO dynamic systems problems and solutions. 29
3.2 Example Bondgraphs 30
3.3 Bondgraph notation 32
3.4 Example schematic synthesis problem specification and one solution. . 39
3.5 Graph-simplification rewrite rules 46
3.6 Individual bondgraph groups that isolate 1—C's and 0—I's 49
3.7 Current meter synthesis 54
3.8 Current meter synthesis (continued) 55
3.9 Current meter synthesis (continued) 56
3.10 Speedometer synthesis 58
3.11 Rate-of-climb meter example 60
3.12 Accelerometer synthesis 62

4.1 Example of function sharing 75
4.2 Illustration of function sharing procedure. 80
4.3 Example of structural element represented as a collection of orthogonally configured rectangular-prismatic sections. 82
4.4 Example of physical description of a design. 83
4.5 Complete physical description of pressure gage 84
4.6 Eliminating the fluid resistance. 89
4.7 Eliminating the fluid capacitance. 90

4.8 Organization of function sharing recognition and modification procedures. 93
4.9 Knowledge for other functions in dynamic systems domain. 94
4.10 Knowledge for other functions in dynamic systems domain (continued). 95
4.11 Accelerometer example 1. 100
4.12 Accelerometer example 2. 102
4.13 Rate-of-climb example 1. 104
4.14 Modification invalidates primary function of element. 107
4.15 Knowledge level for fluid resistance. 108
4.16 A surprising design. 110

B.1 Shaft coupling knowledge. 137
B.2 Fastener task situation. 140
B.3 Fasteners in the knowledge base. 140
B.4 Structural framework for a fastener. 141
B.5 The knowledge base description of a fastener 143
B.6 Example output. 145
B.7 Other example fasteners. 146

C.1 Debugging to meet two goals 157
C.2 Components available in the blocks domain. 159
C.3 A valid design. 160
C.4 Program architecture 162
C.5 Speedometer debugging 171

D.1 Easily insertable and removable fastener 176
D.2 Bicycle wheel 177
D.3 A threading tap 178
D.4 Novel ski boot sole 179
D.5 I-beam cross section 180
D.6 Stranded cable 181
D.7 Strain gage 182
D.8 Thermocouple 182
D.9 Fluorescent lights 183
D.10 Resistive heating crucible 184
D.11 Automobile body as electrical ground 185
D.12 Digital circuit chip leads 186

Introduction

Chapter 1

1.1 PROJECT SUMMARY

My work is broadly concerned with the question *How can designs be synthesized computationally?* This project deals primarily with mechanical devices, and focuses on *pre-parametric design*: design at the level of detail of a blackboard sketch rather than at the level of detail of an engineering drawing. This type of design is called pre-parametric because it is concerned with specifying gross design configurations rather than particular numerical values for pre-determined parameters. Throughout this document, I use the words *conceptual*, *preliminary*, and *pre-parametric* interchangeably.

1.1.1 Motivating Scenario

Design is a very broad class of problem solving, but this project is focused quite narrowly. In order to give a general idea of the problem solving task that I have addressed, this subsection describes a problem that can be solved by the computer program that implements the ideas in this document.

Although some of the ideas are quite general, the details of this project are

centered on the domain of single-input single-output dynamic systems. Instruments, sensors, and actuators fall into this class of devices, and examples include pressure gages, pneumatic cylinders, voltmeters and speedometers. Figure 1.1 illustrates the way in which my system solves the problem of designing an accelerometer. The problem is specified as that of designing a device that will produce a deflection, x, that is proportional to a reference frame acceleration, a. The first design step is to generate a design description, in terms of idealized components, that will produce the desired behavior. In this case, the description includes a mass connected by a spring and a damper to ground. The second step is to transform this idealized description into an efficient physical description that represents the structure of the actual device to be built. In the figure, the final description shows a cantilever beam that is thick at one end surrounded by a viscous fluid. In this project, the final physical description can be thought of as a suggestion of a design configuration. That is, the program does not produce dimensions for the cantilever beam, but rather suggests that there be a cantilever beam that is thick at one end. The generation of design descriptions at this level of detail is what I mean by *pre-parametric* design.

1.1.2 Framework for Project

The scenario illustrated in figure 1.1 shows two distinct problem solving steps: first the system produces an idealized description of the device, then the system produces a description suggesting the structure of the device. These two steps correspond to my decomposition of the research task. I call the description produced by the first design step a *schematic description* and the description produced by the second design step a *physical description.* This distinction is shown in figure 1.2. My design procedure consists of two subprocedures: 1) generate a schematic description from a specification (chapters 2 and 3), and 2) generate a physical description from a schematic description (chapter 4).

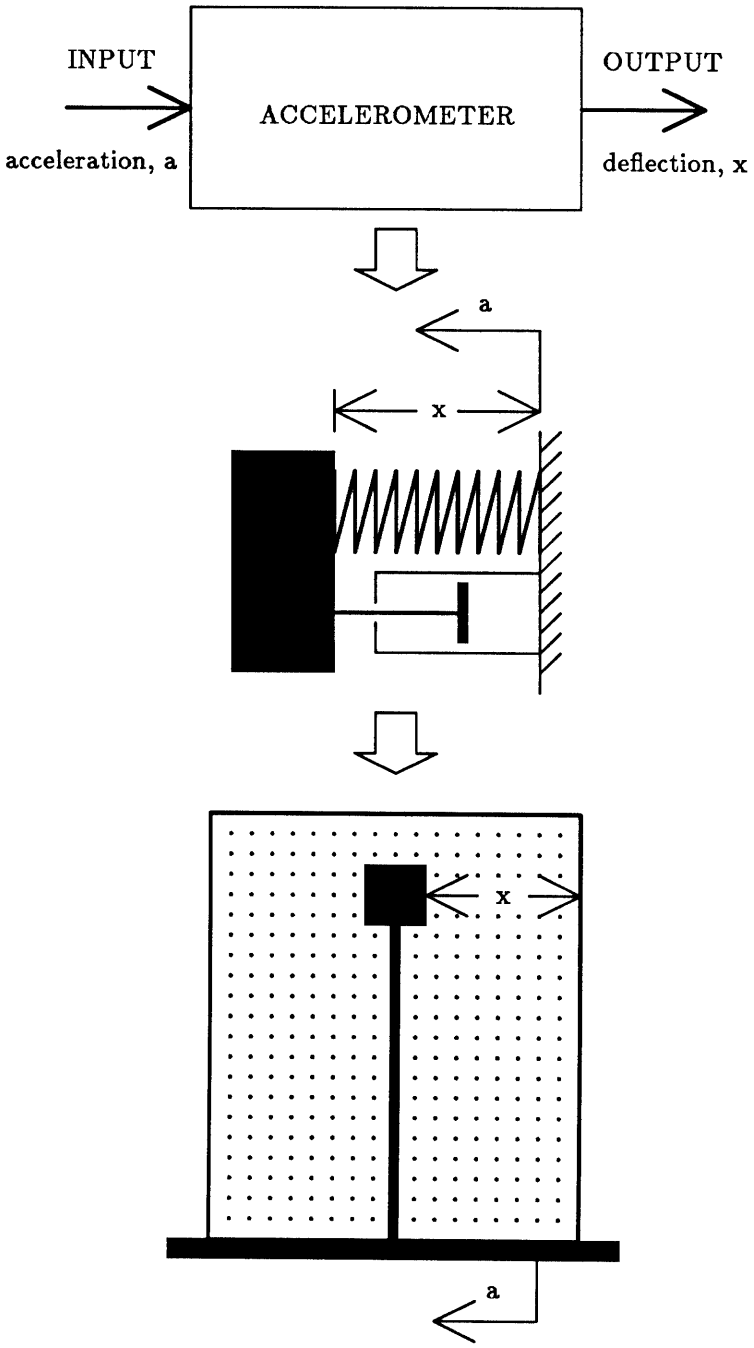

Figure 1.1: Scenario illustrating the design of an accelerometer

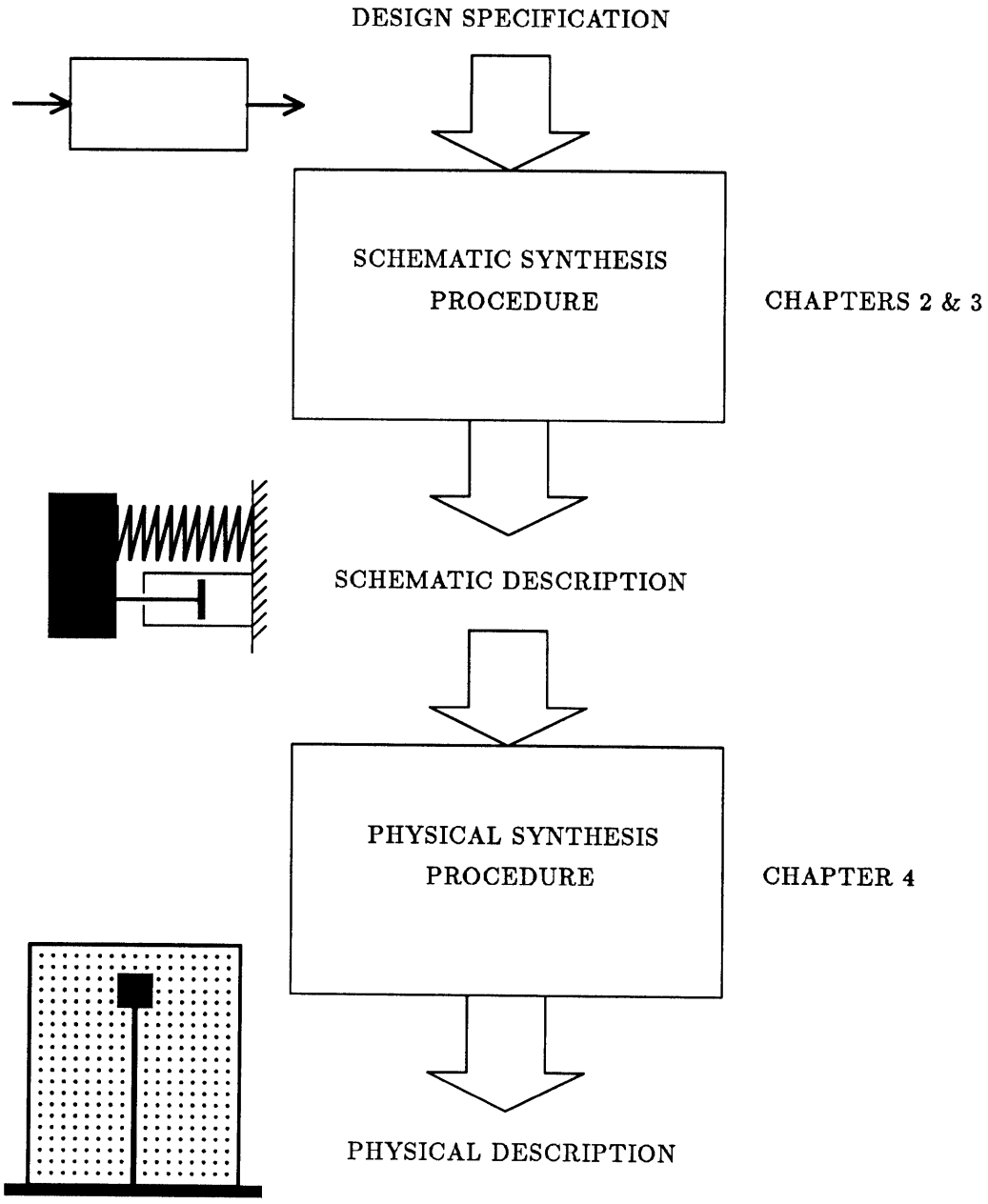

Figure 1.2: Problem decomposition

1.1.3 Major Ideas

This project uses the domain of single-input single-output dynamic systems to explore computation and pre-parametric design. Four ideas are at the core of the design procedures presented in this document.

1. Break a design problem into two subproblems— generating schematic descriptions and then generating physical descriptions. This reduces the problem solving complexity and clarifies the problem solving procedures.

2. A formal schematic language is an important hallmark of a well-defined schematic synthesis problem.

3. A transformation that compresses many different schematic descriptions into a single minimal representation allows for a concise description of the domain knowledge and a straightforward schematic synthesis procedure.

4. Efficient designs are ones in which many functions are provided by the same structural element. This property of design is called function sharing. Function sharing is a key component of my procedure for generating physical descriptions from schematic descriptions.

1.2 Goals and Strategy

1.2.1 Objectives

The primary goal of this research is to develop principles upon which future computational tools for pre-parametric design will be built. I have focused exclusively on the information processing tasks associated with conceptual design, and have ignored entirely the important issues related to how a computational tool might be designed to interact effectively with a human designer.

1.2.2 Motivation

Pre-parametric design is the stage of design receiving perhaps the least attention both in research and in practice. Yet it is at this stage that the basic direction of a project is determined. As a design project progresses in time, the cost of changing the fundamental design concept escalates roughly exponentially. In the first days of a project, changing the design strategy costs practically nothing. Once a fabrication facility has been built for the design, a change in the concept may cost millions of dollars. Because of this characteristic of design, engineers should be very confident that they have selected a design concept wisely. One approach to this selection process is to generate tens or hundreds of alternative concepts early in the project, rather than one or two (as is current industrial practice). An important application of computation is in aiding this concept generation activity.

1.2.3 Approach

The general approach of this project has been to develop research ideas to a level of detail where computer programs can be used as tools for validating the ideas and for exploring their consequences. Specifically, I have approached the problem of synthesizing pre-parametric design concepts computationally by selecting a bounded problem domain, by decomposing the problem into subproblems, by devising representation languages for describing the designs, and by developing techniques for guiding the synthesis process within the space of possible designs defined by the representation languages. Finally I have used computer implementations of these ideas to clarify my thinking and to test the results.

1.3 SOME PROPERTIES OF DESIGN

Design has traditionally been viewed as the art of engineering, yet most of this document is devoted to illustrating *procedures* for performing design tasks. To place these procedures in the proper perspective and to help demystify the design task, this section describes some very general properties of all engineering design.

1.3.1 Design is a Transformation Between Descriptions

Design can be thought of as a transformation between descriptions of devices. Generally, design transforms a functional description into a structural description. Structural descriptions may be communicated with drawings, models, text, or tables of numbers. The medium of description constitutes a design language. These languages are often informal in the case of blackboard sketches by human designers, or may adhere to formal definitions in the case of an engineering drawing or a computer-language description. In either case, designs are constructed from primitive building blocks that might be lines, features, components, or systems.

1.3.2 Design Is Combinatorial

The design process is combinatorial— that is, designs are collections of primitive elements, and many different elements can be combined in many different ways. Consider for example the relatively simple design problem of selecting a shaft, two bearings, two grease seals, and two bearing retaining devices for a specified spindle configuration. If there are 10 possible shaft dimensions and materials, 500 possible bearings, 50 possible seals and 15 possible retaining devices, then the unbridled complexity of the design problem is 3,750,000 potential designs. As with many other information processing tasks, combinatorial complexity makes design difficult.

1.3.3 Fundamental Trade-Off Between Expressiveness and Complexity

The primitive elements of a design language determine the space of possible solutions to a design problem. A designer can only generate structural descriptions that can be formed from these elements. A simple example illustrates this point. Imagine an arch designer. If the designer can work with only rectangular blocks then arch 1 can be generated but not arch 2 or arch 3 (figure 1.3).

The constraints imposed by the choice of design primitives are the cause of a fundamental trade-off between the expressiveness of the design language and the complexity of the design problem. As design languages become more expressive, the space of possible designs increases. Again, consider the arch example. If we allow arch designs to contain three rectangular blocks and specify that blocks can be joined at any of 10 points along the circumference of a block, there are about 20,000 ways the blocks can be put together. If we increase the expressiveness of the design language by using triangular blocks that can be joined at any of 3 points along their circumference, the 24 triangular blocks (equivalent in area to the 3 rectangular blocks) can be put together in about 10^{44} ways. Notice that this increase in expressiveness allows the system to design many more kinds of arches, but at a large cost in the complexity of the design problem. This trade-off between the expressiveness of the design language and the complexity of the resulting design space is the primary reason novel conceptual design is difficult. Although expressiveness is necessary to allow for the generation of a wide variety of designs, simply increasing the expressiveness of a design language swamps the designer with alternatives. So, any increase in expressiveness must be accompanied by an increase in the designer's ability to control the complexity of the design space.

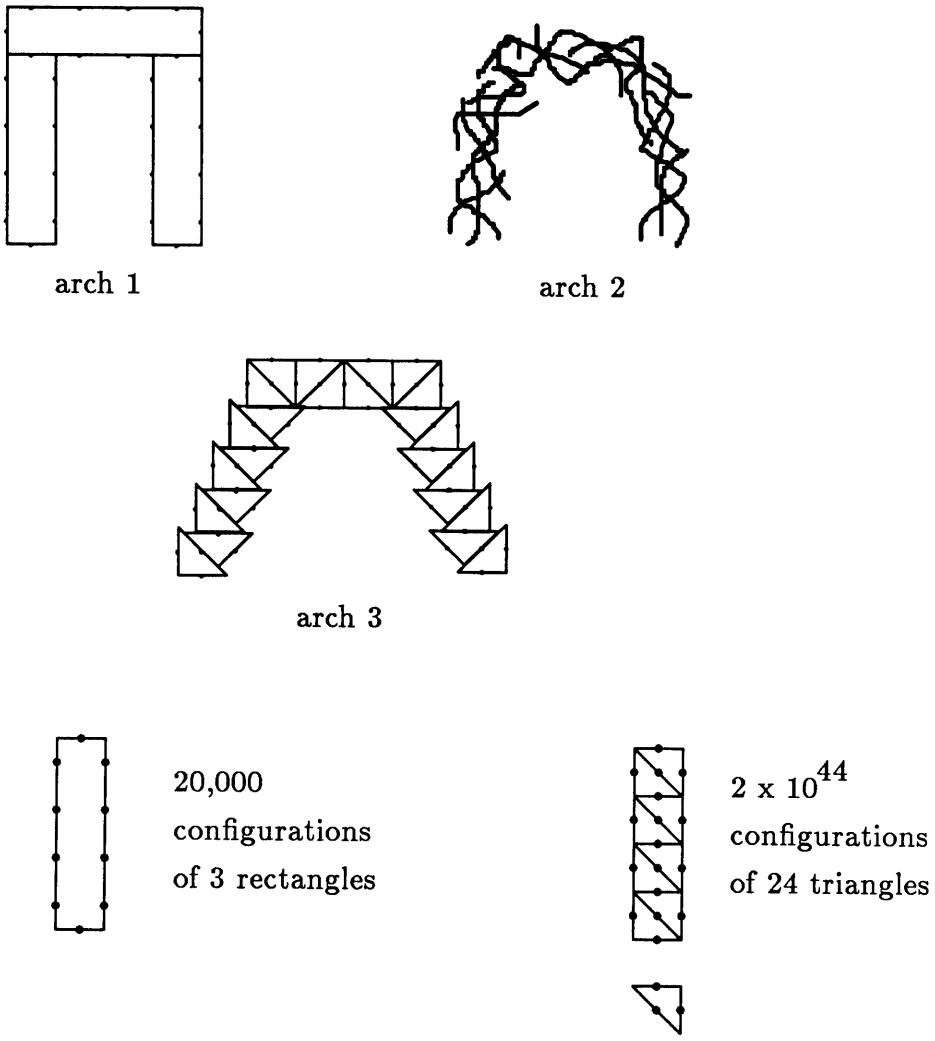

Figure 1.3: Trade-off between expressiveness and computational complexity

1.3.4 Design is Exactly Solvable at Best, Intractable at Worst

Design is a very broad category of intelligent behavior. Some design problems have exact, globally optimal solutions. Some design problems are intractable.

Consider the problem of designing a constant cross-section rectangular box beam 1 M long. The design can specify either a particular aluminum alloy or a particular steel alloy. The beam wall thickness must be at least .002 M. The design problem consists of specifying a height, width and thickness for the beam to provide the greatest bending strength per unit mass. Mathematical optimization techniques can find the global optimum for this problem exactly and in a reasonable (polynomial) amount of time.

Now consider the (somewhat contrived) problem of designing a key for a door lock. The lock has 12 tumblers and the key should have notches with an integer depth from 1 to 10. The designer is provided with a *black box* computer simulation of the lock that will reply whether or not a key design will open it. There are 10^{12} alternative designs, and the designer has no choice but to try them exhaustively. This problem is not solvable in a reasonable amount of time.

In practice most design problems fall between these extremes. It is rare to be able to solve a problem exactly. But, it is also rare that problems are posed in a way that a designer is forced to blindly search for solutions.

1.3.5 Global Optimality is Usually Unrealistic

Except in some mathematically parameterized design problems, design spaces are very sparse and riddled with local optima. In these cases, the notion of global optimality is not relevant. There is usually no way of placing a bound on the best solutions. There is also typically an infinity of solutions.

Designers will usually accept designs that meet most of their specifications. In

better cases, the designer will have several acceptable designs to choose from. In general there is no way of knowing if other, better designs exist.

1.4 DESIDERATA FOR COMPUTATIONAL DESIGN RESEARCH

Design is a complex, vague, human activity involving information processing of many different types. I believe that this attribute of *the design problem* suggests some desiderata for design automation research. In particular, a completed piece of research ideally addresses four subtasks: 1) definition of the specific design task being addressed; 2) development of a representation for specifications and designs; 3) identification of the techniques used by the system for controlling design space complexity; and 4) demonstration of the design techniques with a computer program implementation.

1.4.1 Definition of the Design Task

The first important step in computational design research is defining the design task. What is the domain? What is the transformation we would like to perform? What are the inputs, what are the outputs? What knowledge is required to perform this transformation? The explicit identification of the design task makes the problem accessible to other design researchers. Without this identification, the research contribution can never be clearly identified.

1.4.2 Representing Specifications and Designs

Coupled to the identification of the design task is the development of representations for the specification of the design problem, the description of the design, and the

knowledge that will be required to synthesize the design. The representation languages must be carefully chosen since they constrain what can be expressed about the design. Any computational system that performs design tasks must have some representation, but the explicit identification of the representation helps to clarify exactly what design tasks the system can perform.

1.4.3 Identifying Techniques for Controlling Complexity

All successful design systems must incorporate some means of controlling the combinatorial complexity of the design problem. Some systems will use design rules culled from human experience; some will use constraint propagation, branch and bound search, or gradient techniques. The techniques that a system uses can usually be explained independently of the particular domain of application. An explanation of the relative importance of multiple techniques is also very useful in assessing why a particular approach is or is not successful.

1.4.4 Computer Program Demonstration of Results

Writing computer programs helps to eliminate fuzzy conceptual thinking, and helps to prove the validity of design techniques. The process of formalizing a representation and design procedure to the point where it can be encoded in a computer program unearths conceptual bugs. Computer programs may also be necessary to empirically validate results, since in many cases, design techniques will be heuristic.

1.5 GUIDE TO THIS DOCUMENT

This document describes several different branches of a single project aimed at understanding how mechanical design concepts can be synthesized computationally. The most coherent story is told by the five main chapters, while some of the peripheral

work is described in the appendices. In total there are nine chapters— five in the body of the document, four in the appendices.

MAIN CHAPTERS:

- Chapter 1 is the introduction to the document. It contains a description of the objectives, motivation and approach of the project, as well as some background on computation and design. Finally, it outlines four desiderata for research in computation and design.

- I break the global problem of generating design concepts into two parts: generating schematic descriptions and generating physical descriptions. Chapter 2 describes the concept of a schematic description and gives several examples of well-defined schematic synthesis problems.

- Chapter 3 is an example of a solution to a well-defined schematic synthesis problem in the domain of mechanical devices that can be described by networks of idealized elements. It is presented as an example illustrating some of the desirable properties of a solution technique for schematic synthesis problems in general.

- Chapter 4 highlights an important property of mechanical devices— function sharing. I explain function sharing as a general idea and then present a specific procedure and a computer program for performing function sharing in the dynamic systems domain. Function sharing is one way to perform the second part of my design problem decomposition— generating physical descriptions from schematic descriptions.

- Chapter 5 delineates the contributions made by this research and discusses future projects that follow naturally from this work.

APPENDICES:

- Appendix A is the literature review associated with this project. In this chapter, I give one paragraph descriptions of the publications that have contributed directly to this project, and one or two sentence descriptions of those that have contributed in secondary ways. The objective of this chapter is to give researchers in the field of computation and conceptual design a headstart on finding relevant literature.

- Appendix B describes an early chunk of work I did on novel combination—the combination of several structural attributes of different known devices in order to design a new device. The novel combination ideas were developed in the context of designing new kinds of fasteners.

- Appendix C explains a way of organizing design reasoning in order to meet several design goals simultaneously. I call this reasoning *debugging based on perspectives.* The major idea is to organize design reasoning into several explicitly separated problem solving perspectives (as if forming a committee of experts from different fields). This organization allows the program and programmer to control the focus of problem solving effort in a design problem.

- Appendix D outlines some of my preliminary thoughts on what makes a mechanical design concept interesting, creative or elegant. This thinking lead to the work on function sharing in chapter 4, but could also lead to several other fruitful areas of research.

- The literature references are listed at the end of the document.

The Concept of Schematic Synthesis

Chapter 2

Overview

The strategy of this project has been to devise techniques for producing schematic descriptions from specifications and then for producing physical descriptions from schematic descriptions. Schematic descriptions are graphs of functional elements. Examples include digital circuit diagrams, analog circuit diagrams, and process plant flow diagrams. This chapter outlines some desiderata for schematic synthesis problems, gives some example schematic languages and synthesis problems, and explains why schematic synthesis is a good first step in solving pre-parametric design problems.

2.1 WHAT IS SCHEMATIC SYNTHESIS?

In this project, schematic descriptions are defined as graphs of functional elements. A design described schematically consists of a specification of its constituent functional elements and their interconnections. The key distinction between schematic

descriptions and other types of design descriptions is that schematic descriptions generally contain no information about the design's geometrical and material properties. Generating a schematic description in response to a specification of desired device behavior is *schematic synthesis.*

2.2 HALLMARKS OF WELL-DEFINED SCHEMATIC SYNTHESIS PROBLEMS

There are three major hallmarks of well-defined schematic synthesis problems: formality, problem match, and derivability of behavior.

2.2.1 Formality

Formality refers to the degree of precision with which the description language, the specification language, and the modification operators are defined. In order for the description language to be formal, there must be a well-defined set of primitive functional elements, and a specification of how the primitives can be assembled together. The key requirement is that there be a way of determining whether or not a description is well-formed. Design specifications are expressed in some language. This language should also be formal— there must be a well-defined way of specifying the desired properties of the schematic description. Design systems often modify particular schematic descriptions. These modifications should be drawn from a set of modification operators. The requirement on modification operators is that well-formed descriptions remain well-formed after a modification is executed.

2.2.2 Problem match

Formality in the description language, specification language and modification operators is not enough to ensure a good schematic synthesis problem. The formally-defined problem must also correspond in an appropriate way to the real design domain of interest. This correspondence consists of an appropriate choice of functional elements— elements that match the desired description granularity.

Granularity refers to an expressiveness-complexity trade-off. The functional elements can be very simple, requiring complex collections of elements to generate complex behavior; or the functional elements can be very complex, requiring only simple collections of elements to generate complex behavior. In the case of fine-grained functional elements, the range of possible descriptions that can be built from a small set of elements is large; in the case of complex functional elements, a more limited set of designs can be built, assuming that in both cases the overall complexity of the device behavior is equal. The benefit therefore of using a fine-grained description language is expressiveness; the cost is the computational complexity of the synthesis problem. A well-defined schematic synthesis problem involves a set of functional elements just fine-grained enough to cover the desired range of designs.

2.2.3 Derivability of Behavior

Appropriately chosen schematic languages allow for derivation of device behavior from a description of the behavior of each of its functional elements and their interconnections. The derived behavior should also be expressed in such a way that it can be compared to the specification language of the problem. The derivability-of-behavior hallmark ensures that there is a way of evaluating a design.

2.3 EXAMPLES OF SCHEMATIC LANGUAGES AND SYNTHESIS PROBLEMS

This section gives concrete examples of schematic synthesis problems. Although I present these problems informally, in each case, the textual and graphical descriptions shown here could be formalized.

Process plants

Figure 2.1 is a schematic description of a section of a process plant. The functional elements are idealized pumps, heat exchangers, mixers, condensers, etc. A synthesis problem in this domain might be: given a library of equipment elements, generate a schematic description of a plant that will refrigerate water.

Analog circuits

Figure 2.2 shows the schematic description of an analog circuit. The description consists of idealized elements— resistors, diodes, operational amplifiers— connected together in a graph. A synthesis problem in this domain might be: generate a circuit that will take a voltage signal input and produce a proportional current through a load that has a time-varying impedance.

Kinematics

Figure 2.3 shows a schematic description of a planar kinematic chain. In the kinematics domain, functional elements are links and joint types. A problem in this domain might be: given a library of joint-types and a range on link lengths, specify the number of links, their lengths, their topology and their joint types, in order to produce a straight-line motion (within some tolerance band) along 90 degrees of input link rotation.

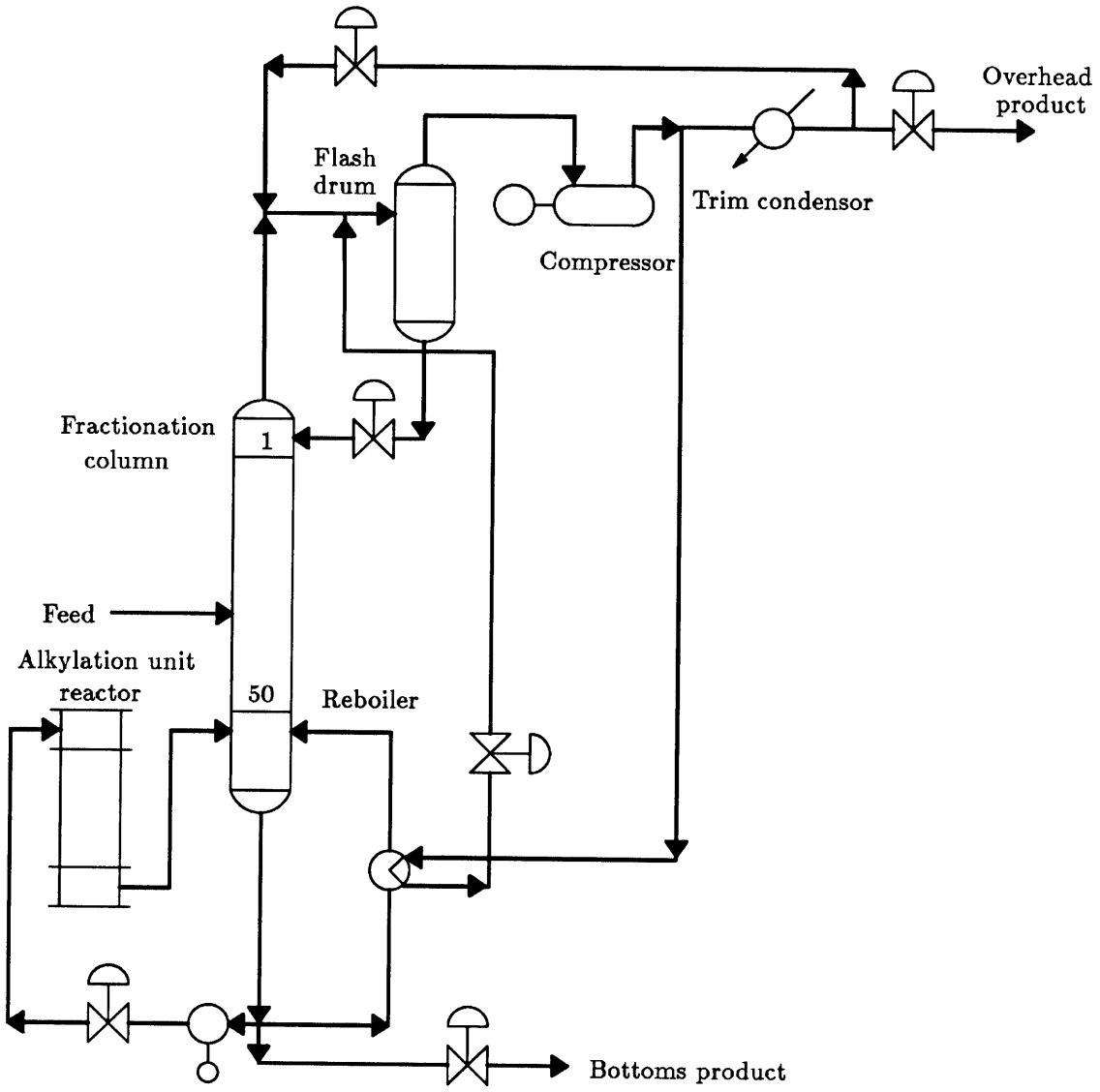

Figure 2.1: Schematic description of a process plant.

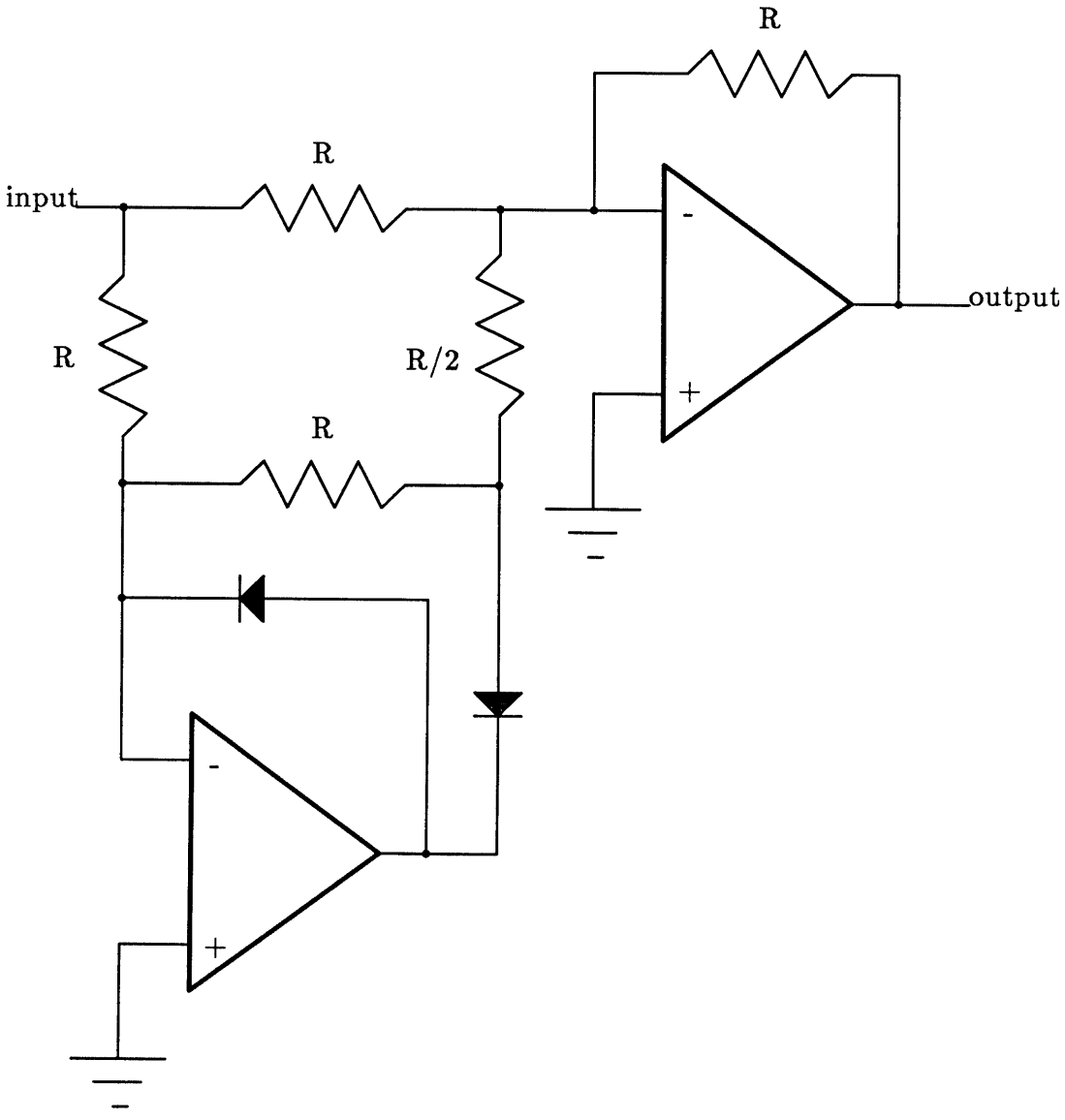

Figure 2.2: Schematic description of an analog circuit

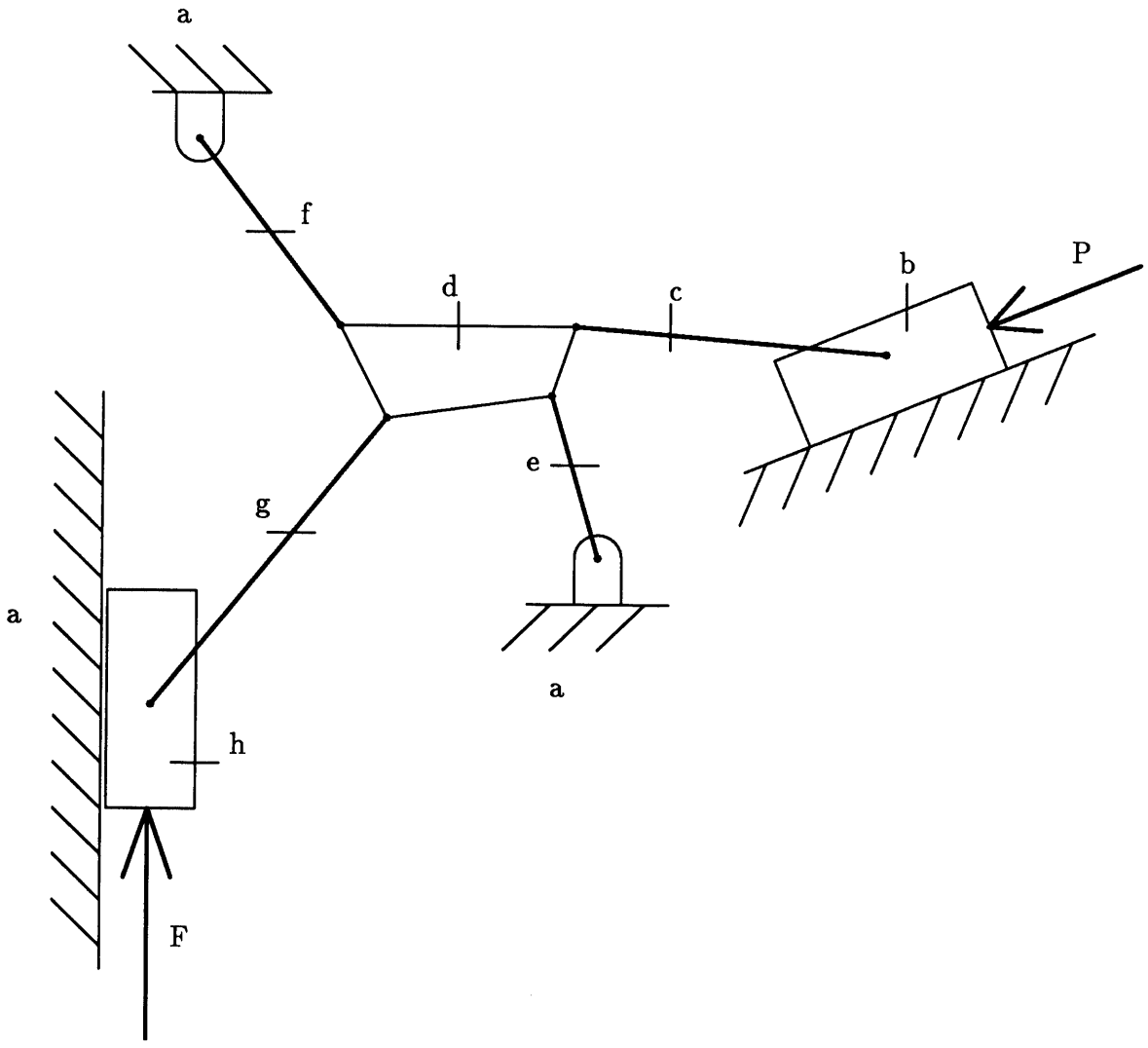

Figure 2.3: Schematic description of a kinematic linkage.

2.4 WHY SCHEMATIC SYNTHESIS IS A GOOD FIRST DESIGN STEP

The task of generating a physical description of a design from a behavioral specification can be broken into two steps: 1) generate a schematic description of the design in terms of functional elements; then 2) from the schematic description, generate a physical description of a device that approximates the schematic description. The generation of a schematic description as a first step in design is justified because the strategy reduces the complexity of the problem solving task and decouples functional and structural design issues.

2.4.1 Reduction in Problem Complexity

Abstraction is one strategy for reducing the complexity of a problem. A schematic description is an abstraction of a design, excluding all physical information except for topology from the design. Working on a design problem first in this space is a less complex problem than trying to design directly in the design space described by the physical representation language of the domain. In general there will be many fewer functional elements in a schematic description language than there will be structural elements in a physical description language. Therefore the space of possible schematic descriptions is usually considerably smaller than the space of possible physical descriptions. Given these factors, an approach to reducing the complexity of the synthesis process is to work first in the smaller space of schematic descriptions, and then, with the resulting schematic description as a starting point, work in the physical description space.

2.4.2 Decoupling Functional and Structural Design Issues

Focusing the initial problem solving effort on schematic descriptions decouples the functional and structural aspects of the design problem. Generating a schematic description, without concern for a possible physical implementation, forces the design system to get the ideal device behavior right first. This strategy makes sense since only devices with correct schematic descriptions will meet the design specifications. Once the schematic description is right, the synthesis of an efficient physical implementation can proceed.

In the next chapter, I present a technique for generating schematic descriptions of lumped-parameter dynamic systems. The particular technique I have developed is exact and in fact is a good strategy for human designers trying to solve the same kinds of problems. This is an instance of how focusing initially on schematic synthesis can lead to better designs.

A Schematic Synthesis Problem and Solution

Chapter 3

3.1 SUMMARY

This chapter describes a specific schematic synthesis problem and one of its solution techniques.

The domain consists of devices that can be described as networks of lumped-parameter, idealized elements in the translational-mechanical, rotational-mechanical, fluid-mechanical, and electrical media. Such devices include speedometers, accelerometers, pneumatic cylinders and pressure gages.

Schematic descriptions of these devices are represented with bondgraphs which are, in effect, transformations of generalized analog circuit diagrams that make Kirchoff current and voltage junctions explicit.

Problems are specified in this domain by specifying a dynamic model of the input and output environment of the device to be designed. These models are represented as chunks of bondgraphs. Further problem specification is made by identifying the desired relationship between a quantity in the input graph chunk and a quantity in

the output graph chunk. The problem is to synthesize a graph containing the input and output bondgraph chunk that will perform the desired transformation between quantities.

The solution technique is based on three steps: 1) generate a candidate design; 2) classify the behavior of the candidate; 3) based on the derived behavior and domain knowledge, modify the candidate (if possible) to bring it in line with the specification.

The procedure for generating a candidate design is based on the observation that any device with a non-trivial relationship between two quantities must contain a physical connection between the two quantities. The generation procedure produces a set of candidate designs from which all valid designs must be derivable by *addition* of bondgraph elements only.

A candidate design is classified by deriving its equations of motion and extracting from them their global behavior.

Finally, candidate designs are modified by first transforming them to a compact representation in which a single description captures the behavior of infinite sets of particular designs. After this compression, the required design modifications become clear. The modifications are carried out on the compact description of the design and the compacting transformation is reversed to yield a modified version of the original candidate design.

The key idea behind this technique is that an abstract characterization of the essential properties of the design descriptions expedites the analysis and modification of candidate designs.

3.2 DOMAIN DESCRIPTION

This section describes my choice of the lumped-parameter dynamic systems domain, explains the schematic description representation, defines problem specifications, and shows how device behavior can be derived from schematic descriptions.

3.2.1 Single-Input Single-Output Dynamic Systems

The domain in which I have applied the schematic synthesis concepts consists of devices that can be described as networks of lumped-parameter idealized elements, and whose behavior can be specified by a relationship between a single input quantity and a single output quantity. I call this domain *SISO dynamic systems* or just *dynamic systems.* Examples of devices in this domain include pressure gages, signal filters, speedometers, pneumatic cylinders, and accelerometers.

The lumped-parameter elements used in the dynamic systems domain are idealized generalized resistances, capacitances, inertances, as well as transformers and gyrators. These elements have instances in the fluid-mechanical, translational-mechanical, rotational-mechanical and electrical media. For example, in the electrical medium the elements are idealized resistors, capacitors and inductors. In the translational-mechanical medium the elements are idealized dampers, springs and masses. In the rotational-mechanical medium the elements are idealized rotary dampers, torsional springs, and rotary inertias. And in the fluid-mechanical medium the elements are idealized fluid resistances, fluid capacitances, and fluid inertances. In addition to these elements there are elements that can transform quantities in one medium to quantities in another medium– like idealized motors, piston-cylinders, or racks-and-pinions. This domain can be thought of as a kind of generalized analog circuit domain in that the elements impose constraints on generalized effort and flow variables (corresponding to voltage and current) and their interconnections obey generalized Kirchoff's laws.

I chose this domain for three reasons:

1. There already exists a well-defined schematic language for describing devices in this domain.

2. The domain is of engineering interest and importance.

3. The domain fits well with the second part of the global design problem addressed by this document: generating efficient physical descriptions from schematic descriptions (Chapter 4).

3.2.2 Example Problems and Solutions

Figure 3.1 shows several schematic synthesis problems in the dynamic systems domain along with a sketch version of example solutions. The figure is intended to give a rough sense of what this chapter is about and of the kinds of problems my technique addresses. The first problem is the design of a pressure gage. The solution is to connect the pressure source and output spring with a piston-cylinder. The second problem is to design a device to produce a voltage on a resistor that is proportional to an input velocity. The solution is to use a rack and pinion connected to a generator. The third problem is to produce a pressure in a fluid capacitance that is proportional to an input angular velocity. The solution is to connect the input through a rotary damper to a rack and pinion. The rack is then attached to a piston-cylinder connected to the capacitance. The final problem is to convert a voltage to an angular deflection. The solution is to connect the voltage source with a series resistor to a motor that is attached to the output. The output is also connected to ground with a torsion spring.

3.2.3 Representing Schematic Descriptions

I use *bondgraphs* as a schematic description language. The rest of this chapter relies heavily on the syntax and notation associated with this representation.

Bondgraphs

For the dynamic systems domain, bondgraphs provide a useful schematic language for representing designs. Bondgraphs are a modeling and analysis tool invented by

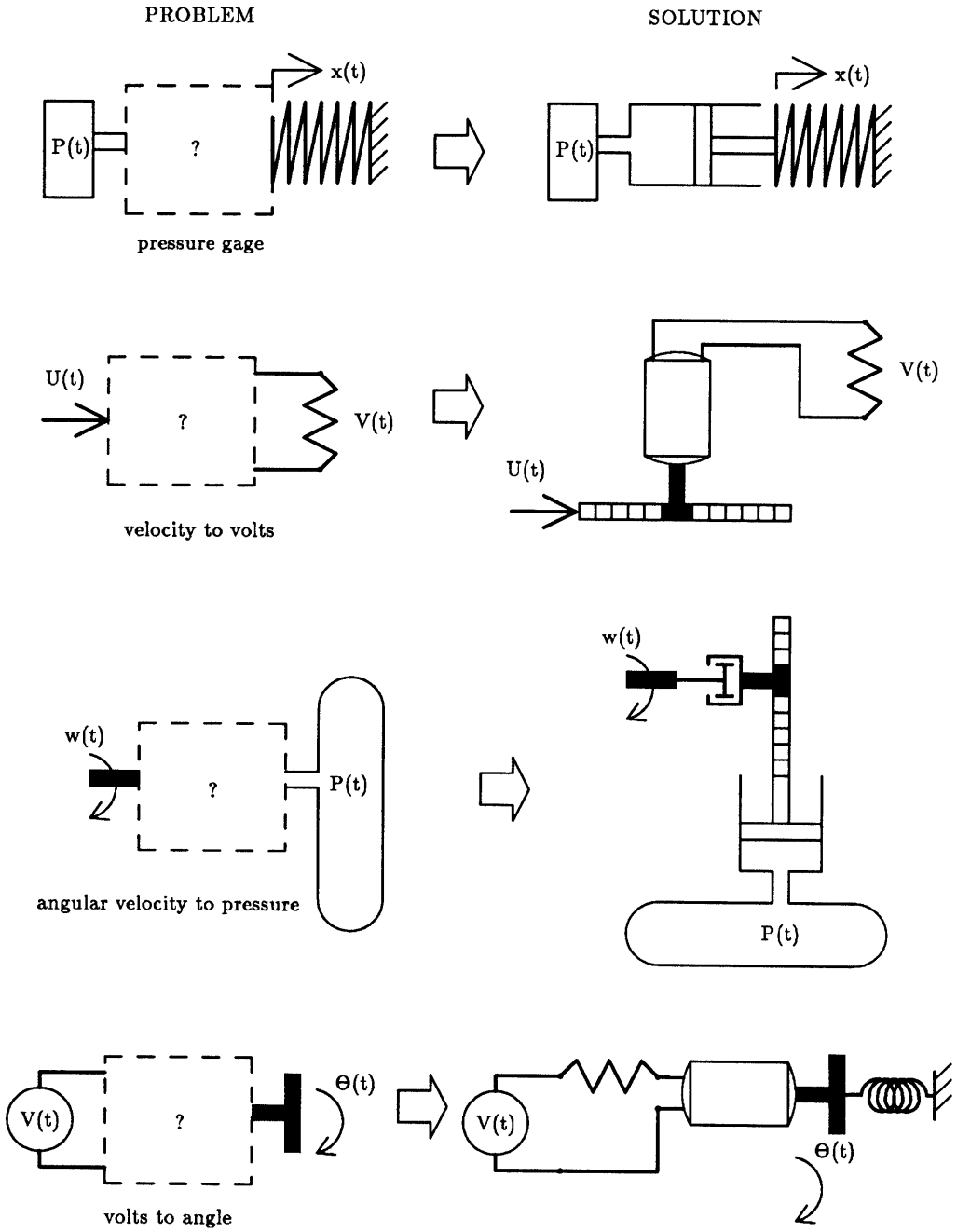

Figure 3.1: Example SISO dynamic systems problems and solutions.

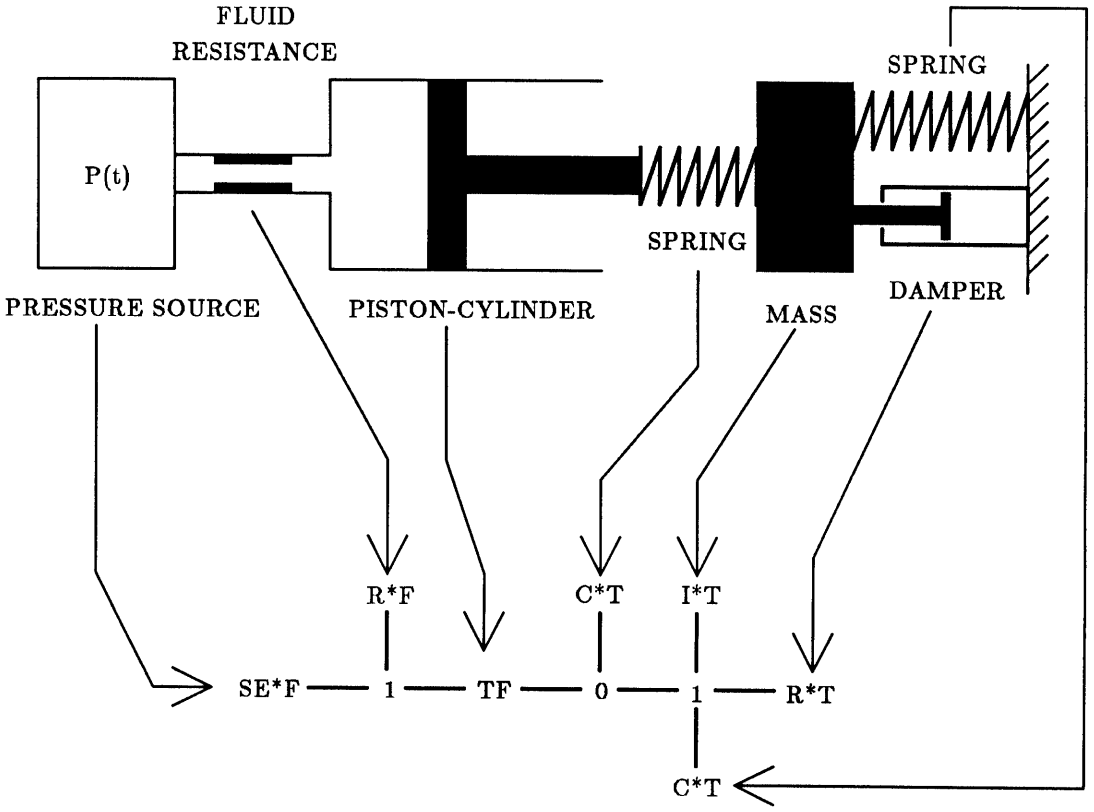

Figure 3.2: Example Bondgraphs

Paynter [Paynter61]. The following section is a cursory description of the bondgraph language; a much more detailed treatment of bondgraphs is in [Rosenberg83]. Bondgraphs are a formal language for describing the exchange of energy in systems composed of lumped-parameter elements. A bondgraph conveys the information contained in a block diagram, a signal flow graph or a linear graph with a single uniform syntax for systems in the electrical, translational-mechanical, rotational-mechanical, and fluid-mechanical media. Figure 3.2 shows three systems described with sketches and with bondgraphs. Corresponding elements share the same labels. The bondgraph notation is shown in figure 3.3.

Bondgraph definitions

Bondgraphs are constructed from ports and bonds, which are the vertices and edges of a graph structure. There are three kinds of ports: 1-ports, 2-ports and n-ports. There is one kind of bond. Our bondgraph notation is specified in figure 3.3.

A bond is a representation of a path for power flow. In my use of bondgraphs, associated with each bond is an effort variable and a flow variable. In the electrical medium, the effort is voltage and the flow is current. In the translational-mechanical medium the effort is force and the flow is velocity. In the rotational-mechanical medium the effort is torque and the flow is angular velocity. In the fluid-mechanical medium the effort is pressure and the flow is volumetric flowrate. The product of these two variables is the power associated with the bond. A bond between two ports indicates an exchange of power between the two ports.

A 1-port represents the lumped-parameter elements. There are four basic types of 1-ports: sources, generalized inertances, generalized capacitances, and generalized resistances. Each of these types of one-ports exists in each of the media. Sources are used to describe inputs that specify an effort or a flow. For example, voltage sources are described as sources of effort in the electrical medium (SE*E), and fluid flow sources are described as sources of flow in the fluid medium (SF*F). The other 1-ports constrain the relationship between the effort and flow on the bond associated with the 1-port. For example, an electrical resistance constrains the voltage and current on a bond to be related by the resistance parameter (Ohm's law). Similarly, a mass constrains the derivative of the velocity on the mass to be related to the force on the mass by the mass parameter (Newton's law). This constraint between the effort and flow on a 1-port is called the *1-port constitutive law.*

2-ports represent elements that transform quantities in one medium to quantities in another medium. There are two types of 2-ports: gyrators and transformers. Transformers convert effort in one medium to effort in another medium, and flow in

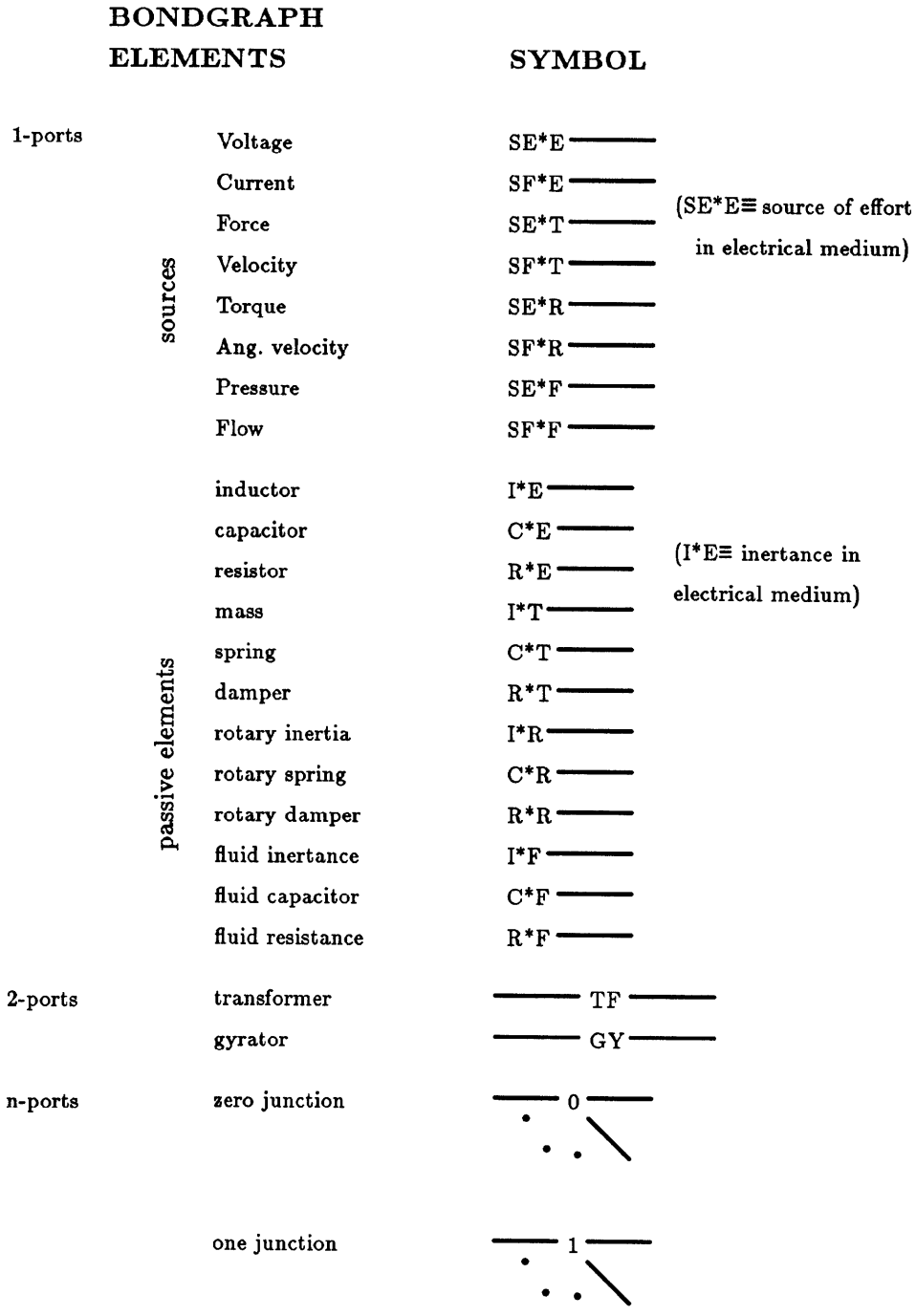

Figure 3.3: Bondgraph notation

one medium to flow in another medium. Gyrators convert effort in one medium to flow in another medium, and flow in one medium to effort in another medium. A motor is an example of a system that approximates a gyrator (torque is related to current; angular velocity to voltage), a rack and pinion is an example of a system that approximates a transformer (torque is related to force; angular velocity to velocity).

N-ports represent the junctions of bonds. There are two types of n-ports: *one*-junctions and *zero*-junctions. One-junctions represent junctions of bonds that all share a common flow, and whose efforts sum to zero. Zero-junctions represent junctions of bonds that all share a common effort, and whose flows sum to zero. (The choice of names for these junctions is an unfortunate historical convention). These two types of n-ports represent the generalized version of the constraints imposed by Kirchoff's current and voltage laws for electrical circuits.

Bondgraphs can be composed by connecting smaller chunks of bondgraph with bonds between n-ports. So a chunk of bondgraph consisting of a voltage source bonded to a zero-junction can be connected to a resistor bonded to a one-junction by establishing a bond between the zero-junction and the one-junction. The inductive definition of a bondgraph is as follows. A 1-port connected to an n-port is a bondgraph. A 2-port connected to an n-port at each of its ports is a bondgraph. Two bondgraphs connected by bonds between n-ports form a bondgraph.

Now that the bondgraph language is fully defined, the notation in figure 3.2 should be clear. The pressure source, the fluid resistance, and the piston-cylinder share the same flow (fluid flowrate) and so are attached to the same 1-junction. The piston and the spring share the same effort (the force across the spring is the same as the force on the piston) and so are attached to the same 0-junction. The mass, damper, and the other spring all share the same flow (velocity) and so are attached to the same 1-junction.

3.2.4 Deriving Behavior From Schematic Descriptions

In order to evaluate a schematic description with respect to a design specification, a design system must be able to derive behavior from schematic descriptions. In this domain, behavior consists of a nominal characterization of the equations of motion for the system. Deriving this behavior from a bondgraph involves two steps: 1) derive the equations of motion, and 2) characterize the equations in terms of their global properties.

Deriving equations of motion

The derivation of equations from bondgraphs is straightforward, and consists of three steps.

1. Impose constraints from generalized Kirchoff's laws. Assign a unique effort variable to each zero-junction and assign a unique flow variable to each of the bonds on the junction. Assign a unique flow variable to each one-junction and assign a unique effort variable to each of the bonds on the junction. Establish equations that state that the flow variables on a zero-junction sum to zero, and that the effort variables on a one-junction sum to zero. (Writing these equations requires the assignment of an arbitrary direction to the variables on each bond.)

2. Impose constraints from 1-port constitutive laws. For each 1-port (capacitance, resistance, inertance, or source) establish the appropriate constraint between effort and flow variables on the 1-port bond.

3. Solve the equations. Given the equations generated in steps one and two, eliminate all of the effort and flow variables except for the variables corresponding to the desired input variable and the desired output variable.

The result of this procedure is a differential equation relating the input variable and its derivatives to the output variable and its derivatives.

Characterizing the equations

The next step in deriving the behavior of the bondgraph is to characterize its equations of motion. Based on an idea from control theory [Ogata70], I have developed a characterization technique aimed at determining the nominal relationship between the input quantity and output quantity. The procedure for characterizing the equations is as follows:

Given a differential equation consisting of sums of derivatives of the input and output quantities (with constant factors):

1. Rearrange the equation so that the terms consisting of derivatives of the input are on the righthand side (RHS), and the terms consisting of derivatives of the output are on the lefthand side (LHS).

2. Perform a LaPlace transformation on the expression leaving an equation between polynomials in the LaPlace operator, s, and the input and output quantity. (In the case of equations of this form, the LaPlace transform is performed by simply replacing s^n for d^n/dt^n). Multiply both sides of the equation by s raised to some power to eliminate any s terms in the denominators of any fractions. Factor out as many s's as possible from both sides of the equation. If an s term can be factored out of both sides of the equation, simplify the equation by dividing both sides by the lower-order of the two factored-out s terms.

3. The order of the remaining factored out s term is called the *type number* of the system. If the s term is on the RHS, then the type number is positive. If the s term is on the LHS, then the type number is negative.

As an example consider the following equation between an input pressure and an output velocity (corresponding to piston-cylinder connected to a mass, a spring and a damper):

$$m\frac{d^2}{dt^2}V + b\frac{d}{dt}V + kV = A\frac{dP}{dt}$$

Performing the LaPlace transform leaves:

$$mVs^2 + bVs + kV = APs$$

Since there is one s that can be factored out of the RHS, the system has a type number of 1, meaning that the input pressure changes with the integral of the output velocity (position). The intuition behind this characterization is that the type number of the equations specifies which integrals or derivatives of the input and output quantity are nominally related. For example a system with an effort input and a flow output that has a type number of 1 would produce a flow proportional to the derivative of the effort, or the integral of flow (position, angle, charge or volume) proportional to the effort. The complete equations are important in refining the particular details of a design, and when choosing design parameters, but only this *type number* specification is important in getting the gross relation between input and output quantities right. One important property of this derivation procedure of behavior from bondgraphs is that it is algorithmic and can be automated.

3.2.5 Specifying a Problem

A schematic synthesis problem in the dynamic systems domain is specified by a relationship between an input quantity and an output quantity. This specification

includes three parts: 1) specification of the variables of interest, 2) specification of a relationship between variables, and 3) specification of the input and output dynamic model.

1. The specification of the variables of interest simply involves identifying either an effort or flow variable in the desired medium. For example, if a voltmeter is desired, the input is a voltage (or effort in the electrical medium) and the output is perhaps a velocity (flow in the translational-mechanical medium). Note that the input and output variables must be efforts or flows. Derivatives and integrals of these efforts or flows are specified when the relationship between variables is specified.

2. Specifying a relationship between variables requires the identification of the appropriate derivative or integral desired of the input and output variables. For example if I want a displacement in response to a voltage, I specify that the nominal relationship between the voltage and velocity will be a first derivative (ie. velocity will correspond to change in voltage, so displacement will correspond to voltage). If I want a displacement proportional to voltage rate-of-change, then I specify that the output velocity will be nominally related to the second derivative of the input voltage. This specification is represented as a *type number*. For example, the first derivative case is type 1, and the second derivative case is type 2. The problem specification can only include nominal, steady-state behavior of the device. That is, the dynamic response of the device is not part of the specification. The assumption is that in pre-parametric design, engineers first want to get the nominal behavior right, and will then optimize parameters to get a particular frequency response.

3. The third part of the specification is the specification of the dynamic model of the input and output. If for example, I want to design a voltmeter that

measures the voltage of a source that is unregulated or is unable to provide very much power, I will specify the input as consisting of an idealized voltage source (one that can provide an arbitrary amount of power) in series with a resistance. This is a good model of a battery. And if my voltmeter must actuate a spring-loaded indicator, I would specify the output flow as acting on a translation-mechanical compliance (a spring).

In summary, a specification consists of an effort or flow variable in a selected medium, an identification of the derivative or integral relation of interest (indicated with a type number), and a specification of a lumped-parameter model of the input and output. These three components of the specification are translated in the bondgraph language as the identification of either a one-junction or a zero-junction as the input and output junction; and bondgraph chunks associated with the input and output junction that model the input and output behavior.

Consider the following complete specification. In designing an aircraft instrument that must actuate a spring-loaded valve with a displacement proportional to rate-of-change in air pressure, I specify the following (shown in figure 3.4):

1. The input port is a zero-junction in the fluid-mechanical medium. The output port is a one-junction in the translational-mechanical medium (corresponding respectively to the air pressure and the velocity of the valve).

2. The relationship between output velocity and input pressure should be one of second derivative– yielding the desired relationship between *displacement* and *rate-of-pressure*, and so would be specified as a type 2 system.

3. The input bondgraph chunk is a pressure-source attached to the input zero-junction, followed by a fluid resistance attached to a one-junction. The output bondgraph chunk is a translational-mechanical compliance directly attached to the output one-junction.

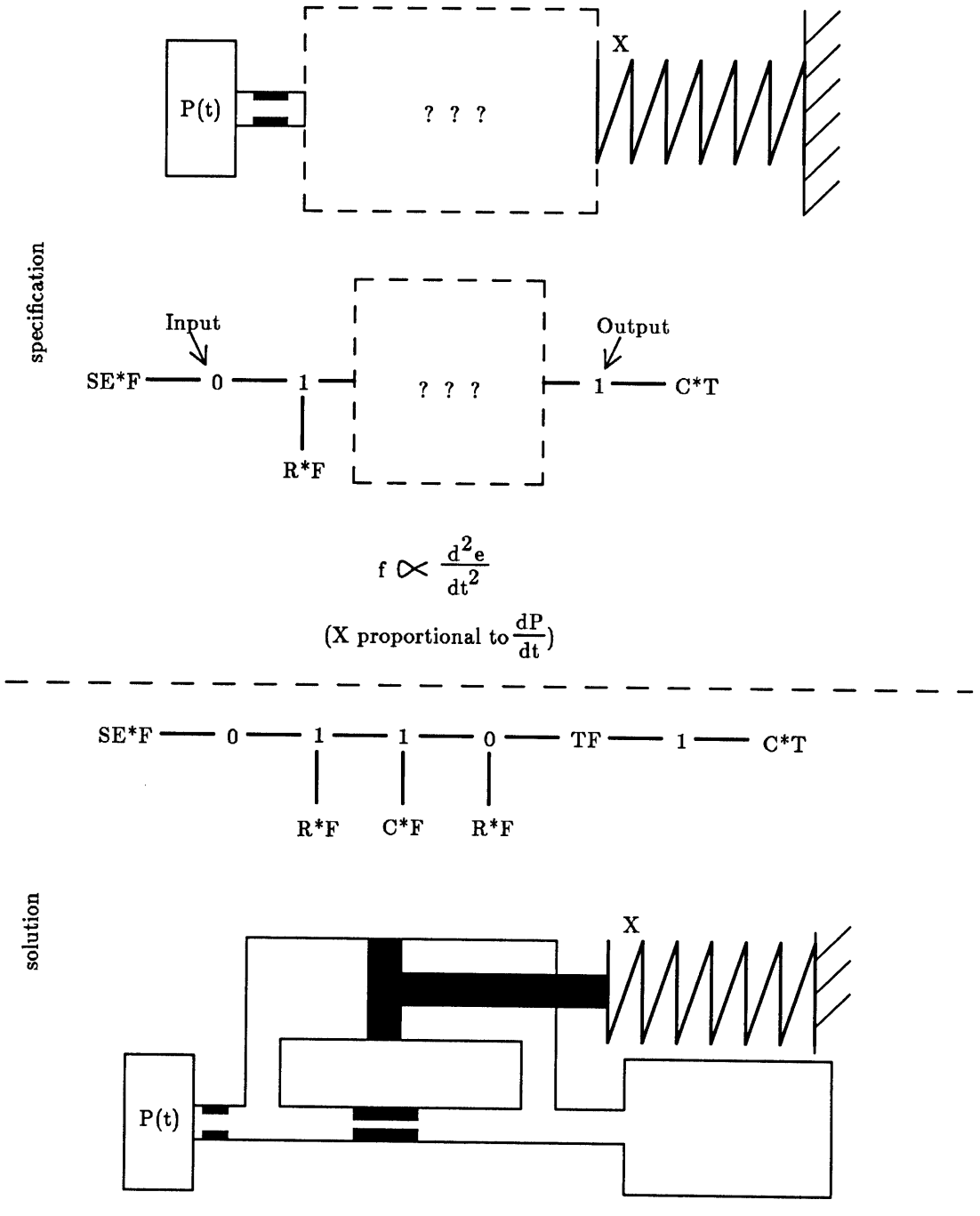

Figure 3.4: Example schematic synthesis problem specification and one solution.

The synthesis problem is to now find a bondgraph that will behave as specified. A satisfactory solution would be the bondgraph shown at the bottom of figure 3.4.

3.3 SOLUTION TECHNIQUE

The solution to the schematic synthesis problem is based on the following strategy:

1. Generate a set of candidate schematic descriptions from which all possible solutions can be obtained through augmentations to the description.

2. Derive and classify the behavior of the candidate descriptions.

3. Based on the behavior of the candidate descriptions, the desired behavior, and domain knowledge, modify the candidates to meet the specifications.

The following three subsections correspond to these three problem solving steps. There are several possible procedures that could be used that are based on different orderings of the ideas in these three steps. I chose the division of the procedure into these steps and this ordering of the steps for clarity of presentation.

3.3.1 Generating Candidate Designs

Part of the input to the schematic synthesis problem is two bondgraph chunks corresponding to the input and output environment of the device. The procedure for generating a set of candidate designs is based on the observation that any schematic description that performs a transformation from a quantity in the input graph chunk to a quantity in the output graph chunk must contain either a bond between the two graph chunks or a sequence of idealized transforming elements (2-ports) that connects the two chunks. I call this sequence a *power spine.*

Concept of a power spine

A bondgraph with a non-trivial relationship between a quantity associated with one n-port and a quantity associated with another n-port must have some sort of power connection between the two n-ports. In order to create a power connection between two n-ports there must be a minimum of either a bond between the n-ports (if the n-ports are of the same medium) or a sequence of 2-ports between the n-ports (if the n-ports are of different media). Any device that meets the design specifications must be derivable from augmentations to this connection between the n-ports.

Connecting input to output

The synthesis strategy for generating a candidate design is to generate a set of schematic descriptions consisting of the input and output graph chunks connected by a power spine. These candidate descriptions are generated exhaustively subject to the constraint that the number of intermediate inter-medium transformations be small (< 3). The size limitation is a constraint imposed by engineering practice—because of cost and reliability, it rarely is a wise design strategy to use more than three media in a SISO dynamic system.

The most important property of the candidate designs generated by constructing the power spine is that it is minimal. Minimality in this context means that any design, containing fewer than three inter-medium transformations, that meets the design specification must be derivable through the *addition* of bondgraph elements to one of the candidate designs.

3.3.2 Classifying Behavior

Given a minimal schematic description that will provide a relationship between input and output, the next step is to classify this initial description. It is possible that the candidate designs meet the specifications, it is also possible that there is a mismatch

between the specified behavior and the actual behavior of the candidate devices, requiring some modification to the initial designs. The degree of agreement between the behavior of the candidate descriptions and the specifications is determined by first deriving the equations of motion for the system and then categorizing these equations according to type number. If the type number of the candidate matches the type number of the specification, then coincidentally the candidates meet the design specifications, and the synthesis problem is solved. More likely, a modification to the design is required.

At this point the synthesis problem can also be assessed as being impossible. Since increasing the type number of a system requires the addition of elements and decreasing the type number of a system requires the removal of elements, it is not possible to decrease the type number of a candidate design. This is because there are no elements to remove from a candidate design. The input and output graph chunks can not be modified since they are part of the problem specification; and no elements can be removed from the power spine since the spine is a minimal connection between the input and output graph chunks (assuming a given set of inter-medium transformations). Therefore, if the specification of the design calls for a type number that is less than that of the candidate design, the synthesis problem has no solution.

3.3.3 Modifying Candidate Schematic Descriptions

The final step is to modify a candidate design that does not exhibit the specified behavior. This process involves four steps: 1) transform the candidate design to a compact description; 2) based on explicit domain knowledge, derive a set of modifications; 3) perform the modifications on the compact design; 4) reverse the compacting transformation. The rest of this subsection is divided into parts corresponding to these steps. I describe the steps at an intuitive level and then give a procedure for carrying them out.

Transform the candidate design to a compact description

The key concept that allows for efficient modification of faulty designs is the use of a compact representation that highlights only the features of the schematic description that contribute directly to its nominal behavior. The transformation of a bondgraph to its compact representation consists of first converting the graph to a generalized equivalent graph containing only an effort source, and then applying some simplifying rewrite rules. The rewrite rules eliminate bondgraph elements that do not influence the type number of the graph. The procedure for performing this transformation is as follows:

- If the bondgraph contains a flow source, replace the source with an effort source and a gyrator. This step does not alter the behavior of the graph except that the equations of motion will be in terms of an effort input instead of a flow input. This transformation allows problems involving effort or flow sources to be compressed into problems involving only effort sources, minimizing the amount of information about the domain necessary to perform the synthesis.

- The next step converts a graph in terms of particular media (electrical, fluid, etc.) into a graph containing only generalized resistances, capacitances, and inertances. Starting from the input port of the graph, traverse the graph converting 1-ports in a particular medium to generalized 1-ports. When a transformer is encountered, remove it. When a gyrator is encountered, remove it, converting all inertances in the un-traversed graph to capacitances and all capacitances in the un-traversed graph to inertances. Additionally in the gyrator case, for n-ports in the un-traversed graph, exchange one-junctions for zero-junctions and zero-junctions for one-junctions. These steps yield a graph with the same characteristics as the starting graph but remove the complexity of dealing with variables and parameters in different media. Transformers do not

influence the nominal form of the equations of motion of a system and gyrators basically influence the equations by converting effort variables to flow variables and vice versa. By swapping zero-junctions and one-junctions; and inertances and capacitances, the effects of a gyrator are maintained even though it has been removed. Removing transformers and gyrators compresses the space of possible schematic descriptions about which the design system must reason.

- The final step is to use the rewrite rules to eliminate bondgraph elements that do not influence the type number of the graph. Use any of the rewrite rules in figure 3.5 that apply to the graph; where the A's and B's stand for any 1-port, and the X's and Y's stand for any n-port such that ports with the same variable are of the same type. Eliminating these non-essential elements completes the transformation of the schematic description into a compact representation.

  An example of a situation in which a rewrite rule would be applied is an inertance and a capacitance were both attached to the same one-junction. In this case, rule 1 would be used (in reverse) to expand the one-junction into two one-junctions with a resistance on one and an inertance on the other. Then rule 6 would be applied to eliminate the inertance. These operations are justified because the inertance adds only transient effects to the behavior of the system when it shares the same flow with a capacitance. The nominal system behavior will be the same without the inertance.

  Rewrite rules are applied based on a match between the condition pattern in the rule and a region of the compact graph. The rewrite rules must only be performed on the regions of the compact graph that correspond to the input and output bondgraph chunks; but not on pieces of graph that span elements from both the input and the output chunk. This is because in the next design step modifications to the graph will be made at the point in the graph where the input and output graph chunks meet. Once a new element is added to

the graph as part of a design operation, the pattern on which a rewrite rule was based would no longer be valid. Since design modifications will never be made to the input and output graph chunks (they are part of the design environment), the pattern on which a rewrite rule is based in these cases will not change.

- Retain a record of the original graph, the transformation operations that were performed, and the rewrite rules that were applied. This records the correspondence between the compact description and the original candidate design, providing information necessary for the reversal of the compacting transformation.

The graph now consists of a single effort source and a network of idealized inertances, capacitances and resistances. This graph is a minimal characterization of the candidate schematic description. A graph in this minimal characterization represents an infinite set of graphs that have the same type number.

Based on domain knowledge, generate modifications

The power of the compacting transformation in this domain is that now all of the domain knowledge can be expressed concisely.

1. In the case of a bondgraph of type -1, adding a 1—R (a new one-junction with an attached R) along the power spine will make it a type 0. The only possible case in which a graph can have a type number of -1 is when a 1—I group is attached directly to an effort source. One instance of this case is a mass with a constant force applied to it. The mass accelerates at a constant rate and therefore the derivative of the velocity is proportional to the force (type -1). Adding a resistance (a translational damper in this case) creates a situation in which the velocity is proportional to the force.

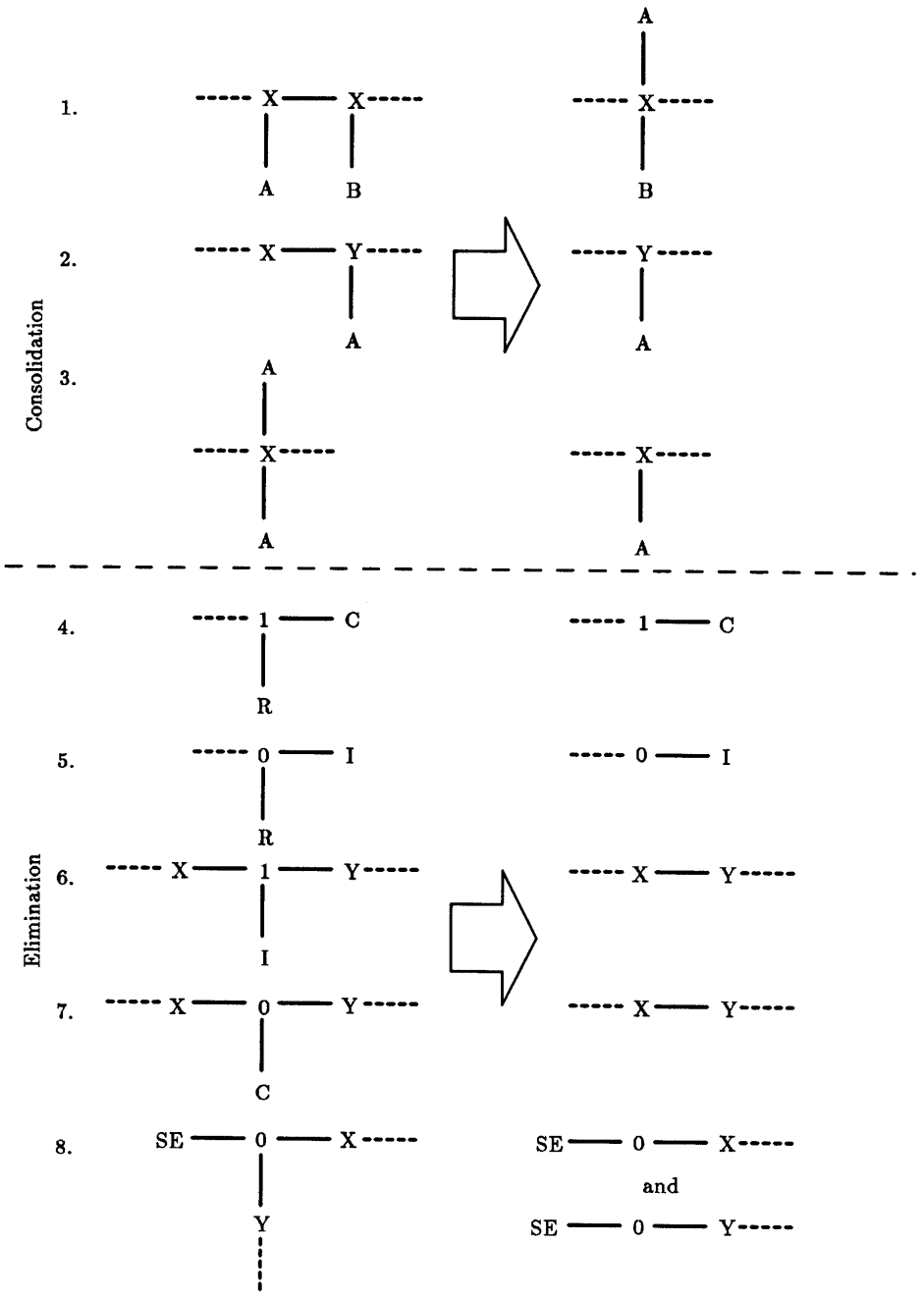

Figure 3.5: Graph-simplification rewrite rules

2. In all other cases, the type of a system is exactly the number of *isolated* 1—C or 0—I groups along the power spine of the graph. An explanation of isolation is given below.

The intuition behind an isolated 1—C or 0—I is as follows. Capacitances and inertances are the only bondgraph elements that have constitutive relations containing derivatives. That is, the relationship between the effort and the flow on a capacitance or an inertance is an equation between one quantity and the derivative or integral of the other quantity. This is in contrast to resistances and sources whose relations contain no derivatives or integrals. The only way to change the type number of a system is to add elements that add derivatives to the equations of motion.

There are certain conditions under which a capacitance or an inertance can influence the behavior of the graph. Imagine a mass connected to ground by a spring. If a force is applied to the mass, after some dynamic effects, it comes to rest at an equilibrium position that is determined by the stiffness of the spring and the magnitude of the applied force. In this case, the type number of the system is not influenced by the mass. The deflection of a different system containing only a spring with the same applied force would be the same as for the system with the spring and the mass. In this example the mass is not isolated, but the capacitance is isolated. I call a capacitance or an inertance that does influence the type number of the system *isolated.* The power spine of a graph connects the input n-port to the output n-port. The spine can be thought of as a string of bondgraph groups of the form (x—A, y—B, z—C) where x, y and z are n-ports and A, B, and C are 1-ports. The bondgraph groups 1—C and 0—I cause differentiation of the input if they are *isolated.* This differentiation is what determines the type number of a system. An isolated 1—C or 0—I group is one whose neighbors (on either side of it in the power spine string) are *isolating groups.*

- An isolating group for a 1—C is a 0—R, a 0—I, or a 0—SE; a 1—I or 1—R occurring as the last 1-port in the bondgraph sequence; or combinations of 1—R's or 1—I's and any other isolating groups. This last case— a combination of 1—R's or 1—I's and any isolating group— is also an isolating group because a 1—R or a 1—I next to 1—C does not influence the type number of a graph.

- The isolating groups for a 0—I are either 1—R's or 1—C's; 0—C's or 0—R's occurring as the last 1-port in a bondgraph sequence; or combinations of 0—C's or 0—R's and any other isolating group.

- Each of these isolation cases is shown in figure 3.6. The arrows in the graph chunks shown in the figure indicate the location of the isolating group with respect to the isolated 1—C or 0—I.

Given this statement of the domain knowledge, the derivation of modification operators is trivial. Except in the case of systems of type -1, the difference between the type number of the candidate design and the type number of the specification is equal to the number of new isolated 1—C's or 0—I's that must be created in the graph. If one isolated group must be created then an isolated 0—I or a 1—C could be added, or a non-isolated 0—I or 1—C already in the graph could be isolated by adding an isolating group.

Carry out modifications on compact description

The only constraint that must be enforced when creating new isolated groups is that new bondgraph elements be added to the power spine only between the compact descriptions of the input and output graph chunks. This is because the input and output graph chunks are part of the environment of the design and can not be modified.

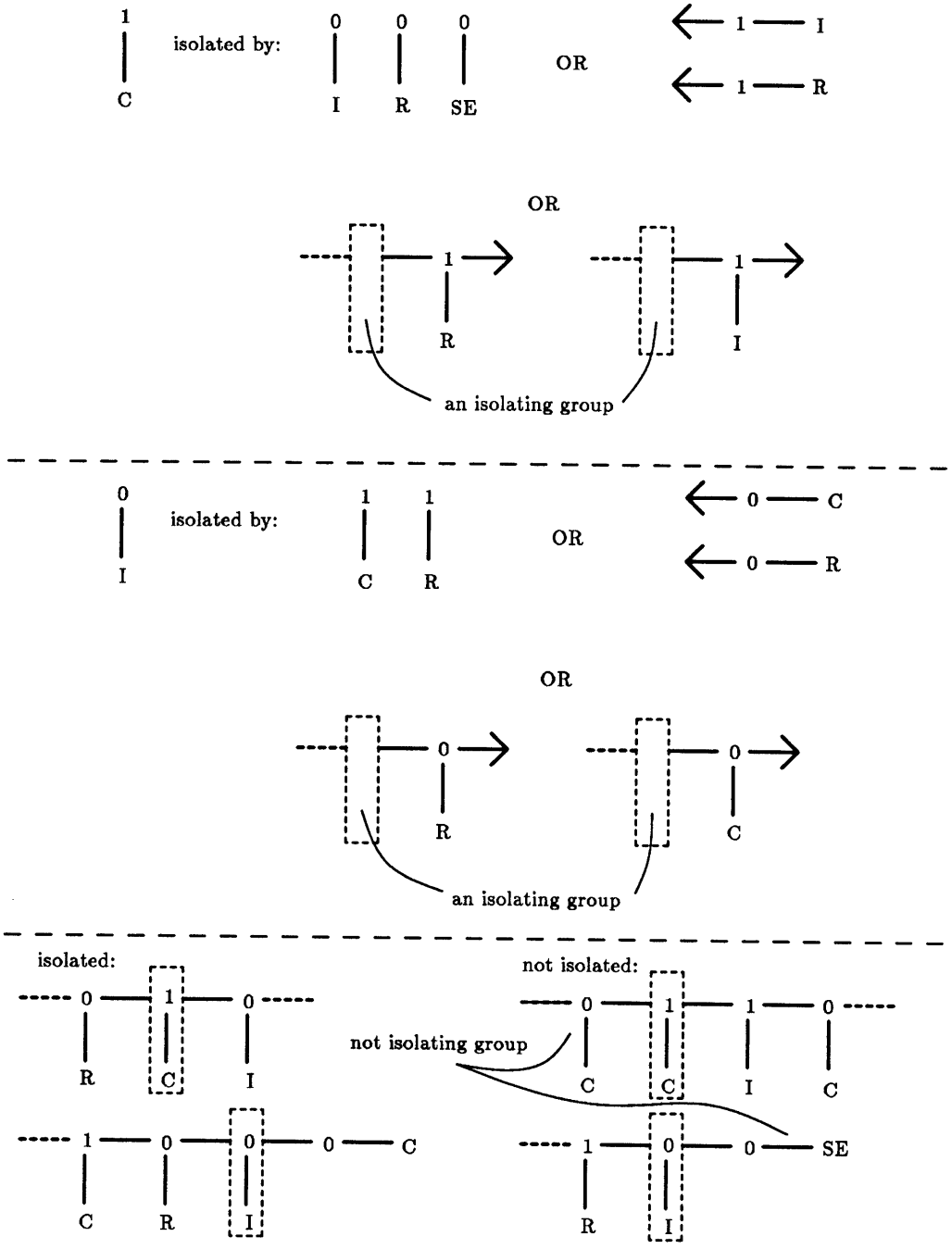

Figure 3.6: Individual bondgraph groups that isolate 1—C's and 0—I's

Reverse compacting transformation

The final synthesis step is to reverse the compacting transformation on the modified design in order to arrive at instantiations in specific media. The reverse transformation is straightforward:

1. Add the elements removed by rewrite operations.

2. Change the generalized graph back to specific media by re-introducing the transformers and gyrators.

3. Change any effort source and gyrator pairs back to flow sources.

The only decision that must be made in this process is where, with respect to the newly added 1-ports, the gyrators and transformers should be reintroduced. The approach I have taken to this decision is to create a version of the design for each possible location of the 1-ports with respect to the gyrators and transformers. There is a range of possible positions in the various media associated with the graph for locations of the newly-added 1-ports. Specifically, the 1-ports can be located anywhere along the power spine of the graph except for within the regions associated with the input and output of the device, as long as the order in which the 1-ports occur is maintained. 1-ports can be moved across transforming 2-ports by simply changing the medium associated with the 1-port. 1-ports can be moved across gyrating 2-ports by changing the medium associated with the 1-port as well as swapping 0's and 1's and C's and I's.

3.4 EXAMPLES

This section is aimed at clarifying the synthesis technique by presenting some concrete examples. I present four examples— one in detail, and three with highlights

only. An important point is that these examples are concerned only with idealized behavior; many of these schematic descriptions would be very difficult to realize physically. In my problem decomposition, the physical implementation issues are faced at the next design step.

3.4.1 Current Meter

This example is illustrated by figures 3.7 through 3.9. Imagine that an engineer wanted to design a device that would take as an input a current from an electrical circuit and actuate a mass with a displacement proportional to the current. Perhaps the device is to serve as a kind of governor on an electric motor circuit— an increase in motor current will move a mass, causing an increase in the rotary inertia of the system. The problem can be stated in the dynamic systems representation as follows: synthesize a graph that will connect graph chunk A with graph chunk B (shown in the figures) such that the integral of the flow on the output n-port is proportion to the flow on the input n-port (ie. a system with a type number of 1). The solution technique proceeds according to the steps developed in the preceding section.

1. Construct the power spine. In this case the power spine is a gyrator and a transformer. The bondgraph elements that form the power spine are between the two vertical dotted lines. These could be thought of as a motor and a rack-and-pinion that connect the electrical graph chunk to the translational mechanical graph chunk. Just one candidate design is shown although there will in general be several possible power spines that connect the input graph chunk to the output graph chunk and that consist of fewer than three gyrating or transforming elements.

2. Derive the behavior of the candidate design. Deriving the equations of motion that relate the output flow to the input flow reveals that the candidate design is

a type 0 system— the output flow (a velocity) is proportional to the input flow (a current). Since the candidate design does not meet the specification, the transforming and modification steps must be carried out. Fortunately since the type number of the system must be increased (from 0 to 1), the modification is possible.

3. Transform the candidate design to its compact representation. Because the input to the candidate design is a flow source, the first step is to replace this source with an effort source and a gyrator. This is because all designs are transformed to contain only an effort source. The next step is to remove all of the gyrating and transforming elements, replacing particular instances of inertances, capacitances, resistances and sources with generalized inertances, capacitances, resistances, and sources. Note that removing a gyrator requires swapping one-junctions and zero-junctions on all the downstream n-ports and swapping inertances and capacitances on downstream 1-ports. The final compacting step is to execute any rewrite rules that may apply. Rule 1 can be used to separate the 1—I and 1—R that share the same n-port. Then rule 6 can be used to eliminate the 1—I.

4. Derive appropriate modifications. The design is specified to be a type 1 system— a displacement (integral of output flow) proportional to current (the input flow). Since the candidate design is of type 0, a 1—C group or a 0—I group will have to be added to the power spine. Observing the compacted design reveals that a 1—C group can be added along the power spine and will be isolated (will have a SE—0 and a 1—R on the left side, and a 1—I on the right side). Adding a 0—I will however require adding another 1—R to its right, since it would otherwise not be isolated on its right side.

5. Carry out the modifications. Adding the two derived graph chunks creates two new compact descriptions of designs that are both type 1.

6. Reverse the compacting transformation. The next step is to reverse the steps executed during the compacting transformation (only design A is shown). First the inertance is added back to the design, reversing rewrite rules 6 and 1. Next the gyrators and transformers are re-introduced. Finally the effort source and gyrator are replaced by a flow (current) source.

7. Consider other locations for newly added 1-ports along the power spine. In the previous step, I chose to leave the added 1-port to the left of the gyrator and transformer in the power spine. The final step is to instantiate the other possible locations for this 1-port along the power spine. It can go between the gyrator and transformer or on the right side of the transformer. Note that moving a 0—I across a gyrator changes it to a 1—C. I show the resulting configurations using mnemonic icons for the bondgraph elements.

3.4.2 Speedometer

The next example is the synthesis of a speedometer-like device. The input is a flow source in the rotational mechanical medium. The output is a flow on a rotational inertia. The desired relationship is between the position of the inertia and the velocity at the input (type 1 system).

1. The first step is to generate a power spine and to derive its behavior. In this example, the power spine is two gyrators (these might be thought of as back-to-back motor-like and generator-like elements). The behavior of the candidate design is that of a type 0 system.

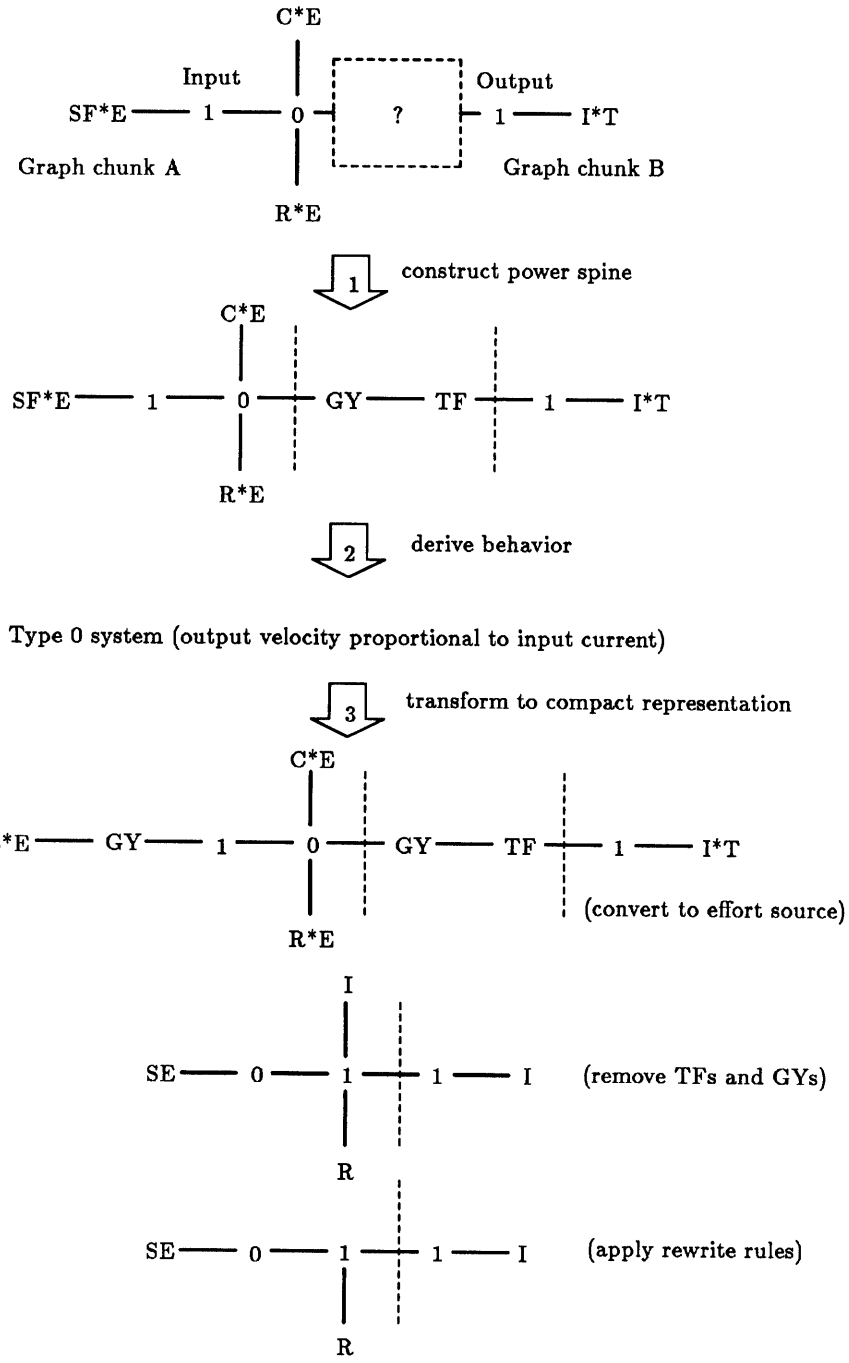

Figure 3.7: Current meter synthesis

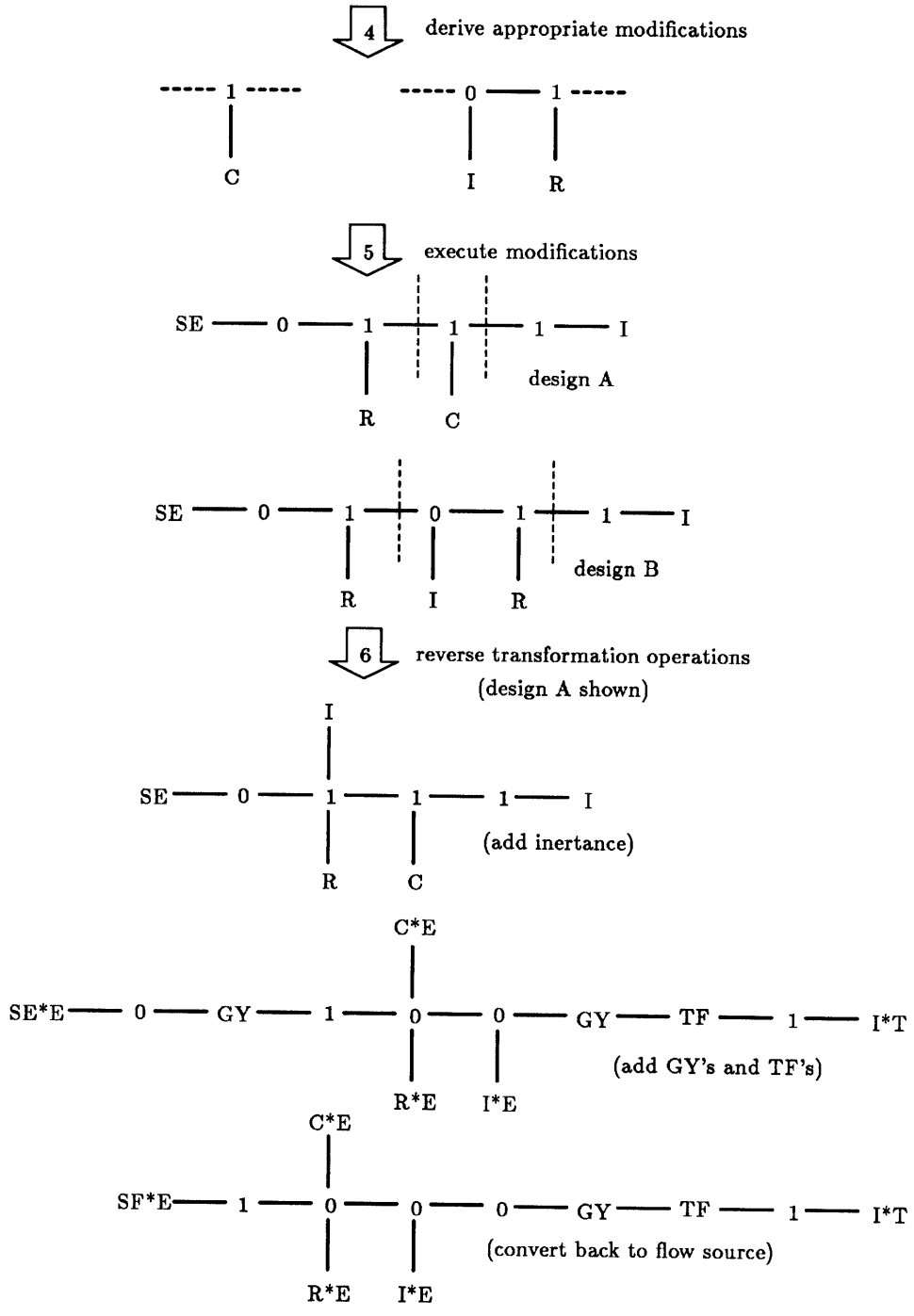

Figure 3.8: Current meter synthesis (continued)

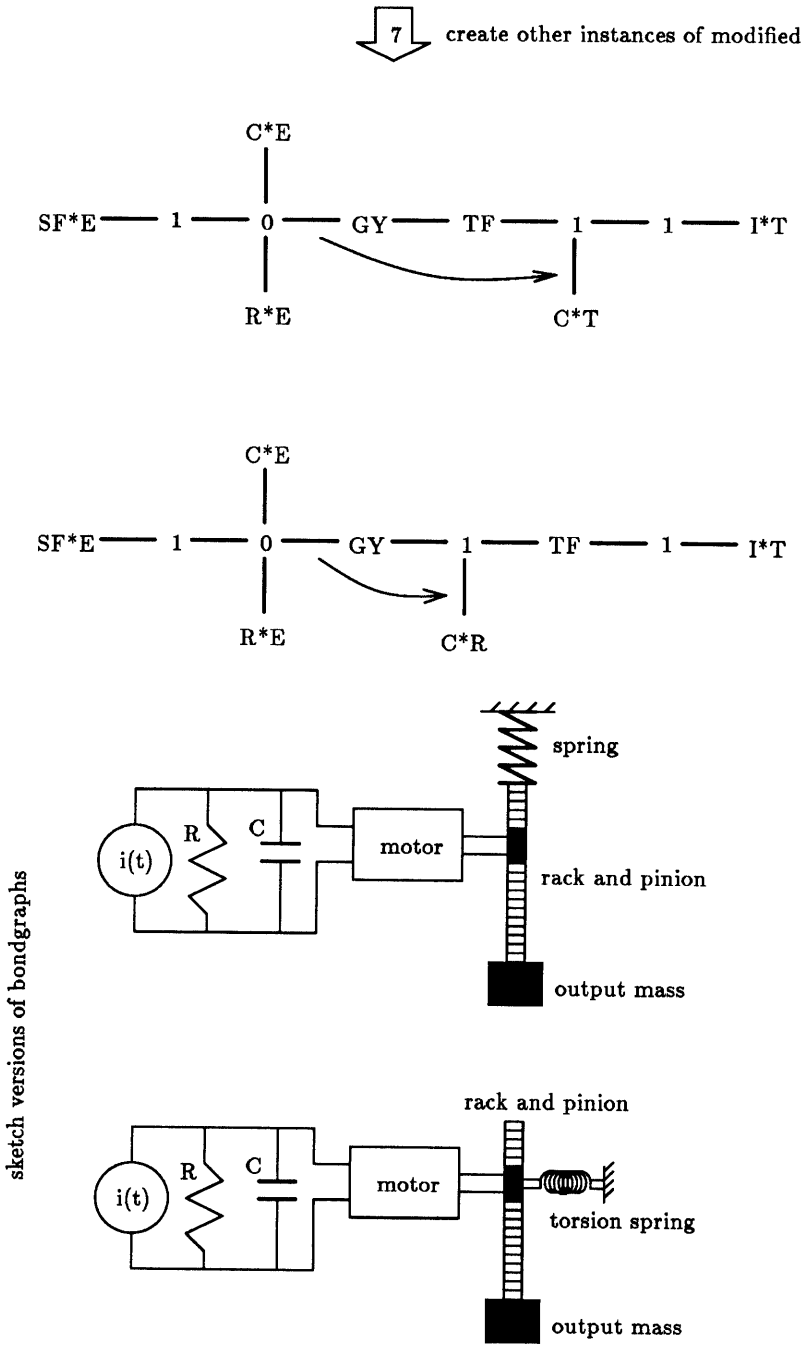

Figure 3.9: Current meter synthesis (continued)

2. The next step is to convert the design to its compact representation. This involves replacing the flow source with an effort source and gyrator and then removing the gyrating elements.

3. Then, appropriate modifications are derived. A 1—C or a 0—I must be added to the power spine to change the behavior from type 0 to type 1. In order to isolate the 1—C it must be paired with a 0—R. The 0—I must be paired with a 1—R. These operations are then executed, yielding two modified designs.

4. Reversing the transformations is the next step. Finally, the added 1-ports are moved around along the power spine to create all of the possible instantiations of the modified designs. Only one such instantiation is shown in the figure.

3.4.3 Rate-of-climb indicator

I used the rate-of-climb indicator example in the last section to illustrate how problems are specified. Recalling the scenario, the input is an effort (pressure) source in series with a fluid resistance, the output is a flow on a spring. The desired relationship is between the deflection on the spring (integral of the flow) and the rate of pressure change on the input (derivative of the effort).

1. First the power spine is generated. In this case, it is simply a transformer between the fluid chunk and the translational mechanical chunk (a piston-cylinder like element). The behavior of this candidate design is that of a type 1 system (ie. the deflection of the spring is proportional to pressure not rate-of-pressure).

2. Next the candidate design is simplified by removing the transformer. Note that after removing the transformer, it appears as if rewrite rule 4 could be used to eliminate the resistance. The rewrite rules can however only be applied

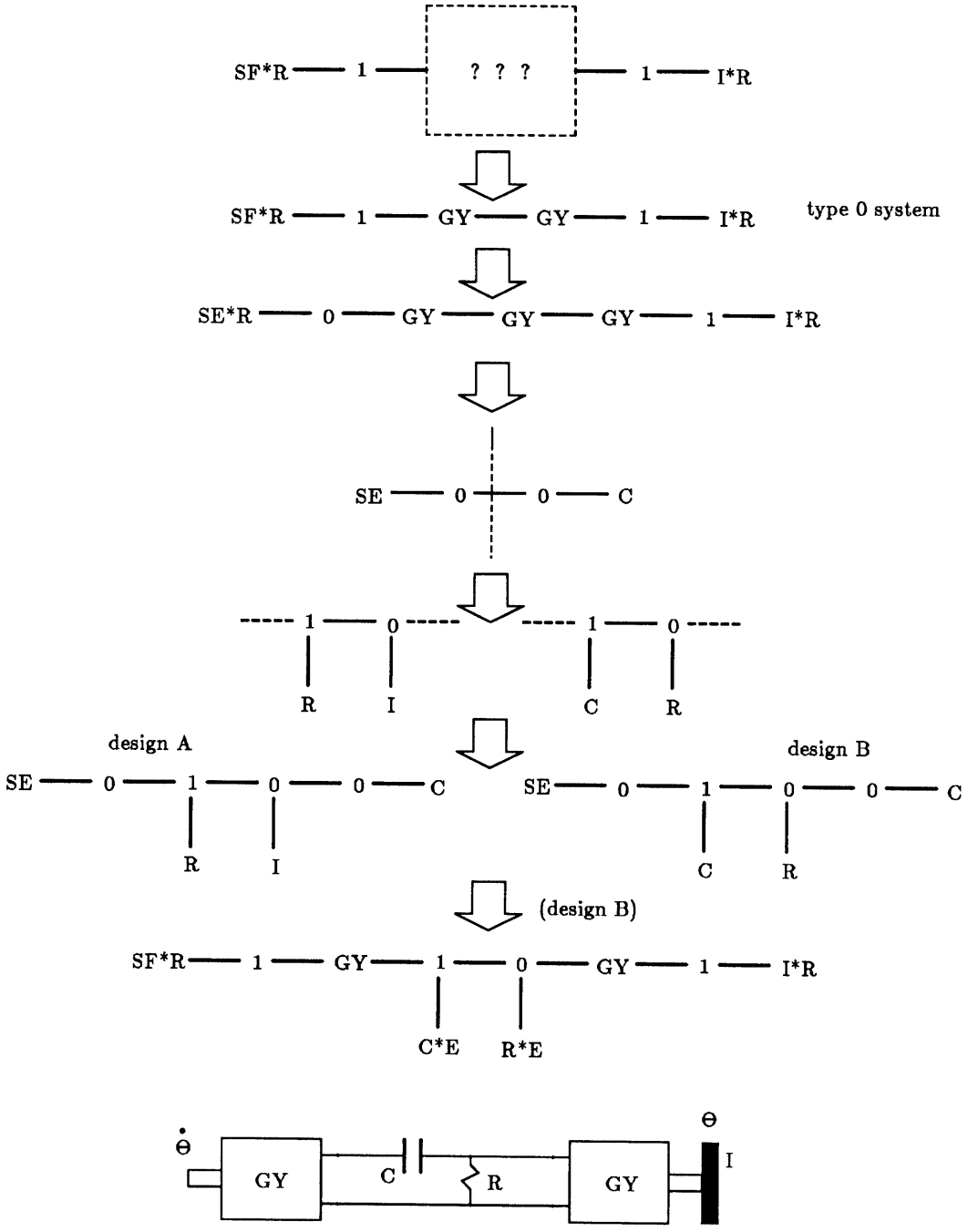

Figure 3.10: Speedometer synthesis

to the initial graph chunks and not across the power spine. This is because the addition of elements to the power spine would invalidate the results of the rewrite rules.

3. The appropriate modification is the addition of a 1—C and a 0—R, or the addition of a 0—I. These modifications are executed.

4. The reverse transformation operations are made and the added 1-ports moved around within the power spine to generate the final designs. Only one design is shown.

3.4.4 Accelerometer

I used the accelerometer example as the motivating scenario in the introduction to this document. This subsection shows how a schematic description of an accelerometer can be derived. An accelerometer is a device that produces a displacement proportional to an acceleration of the device reference frame. In this case, the input is a flow (velocity) source and the output is a flow (velocity) on a spring. The desired relationship is between the derivative of the input (acceleration) and the integral of the output (position). This constitutes a type 2 system.

1. The first step is to generate the power spine. In this case it is simply a bond between the input and output chunks. The resulting device is of type 0. (Since an ideal translational mechanical flow source can specify an arbitrary velocity independent of force, the spring has no effect on the input).

2. Transforming the candidate design is trivial in this case, since the graph contains no 2-port elements. The only required operation is replacing the flow source with a generalized effort source.

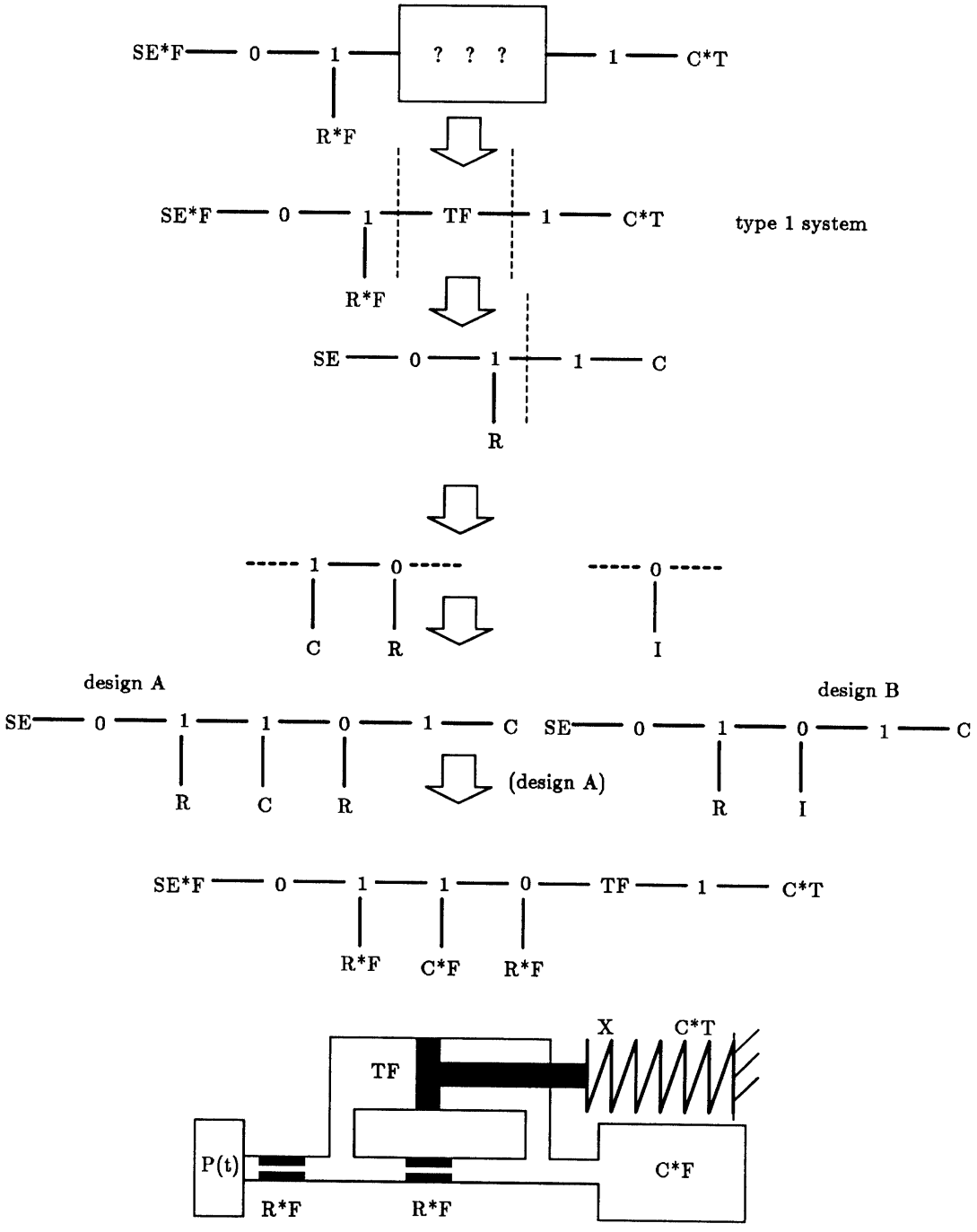

Figure 3.11: Rate-of-climb meter example

3. In order to create two isolated 1—C or 0—I groups, the only modification that is required is either to add a 0—I to the power spine or to add a 1—C flanked on both sides by 1—R's.

4. Reversing the transformation simply requires converting the effort source back to a flow source.

3.5 DISCUSSION AND ANALYSIS

This chapter has presented a formal solution to a synthesis problem that people have traditionally solved in an ad hoc, heuristic way. Synthesizing devices in this domain is simple given the right representation and abstractions. This section analyzes the success of the technique, explores its extension within the dynamic systems domain and to other domains, outlines some lessons to be learned from this instance of a schematic synthesis problem and solution, and discusses the computer program implementation of these ideas.

3.5.1 Utility of the Technique

When I began thinking about the SISO dynamic systems synthesis problem, I gave several engineers some sample problems to see how they performed the task; most of them found ways to avoid doing the exercise. One or two engineers actually found a solution. No one found more than one solution. The technique I have developed provides all possible solutions (containing fewer than three inter-medium transformations) to a specified solution. This results in an improvement in the state of the art in this area of synthesis. Not only are these techniques useful for computer programs but also are useful for pencil and paper approaches. The results of this single experience suggest that design in general will benefit from formalizing some

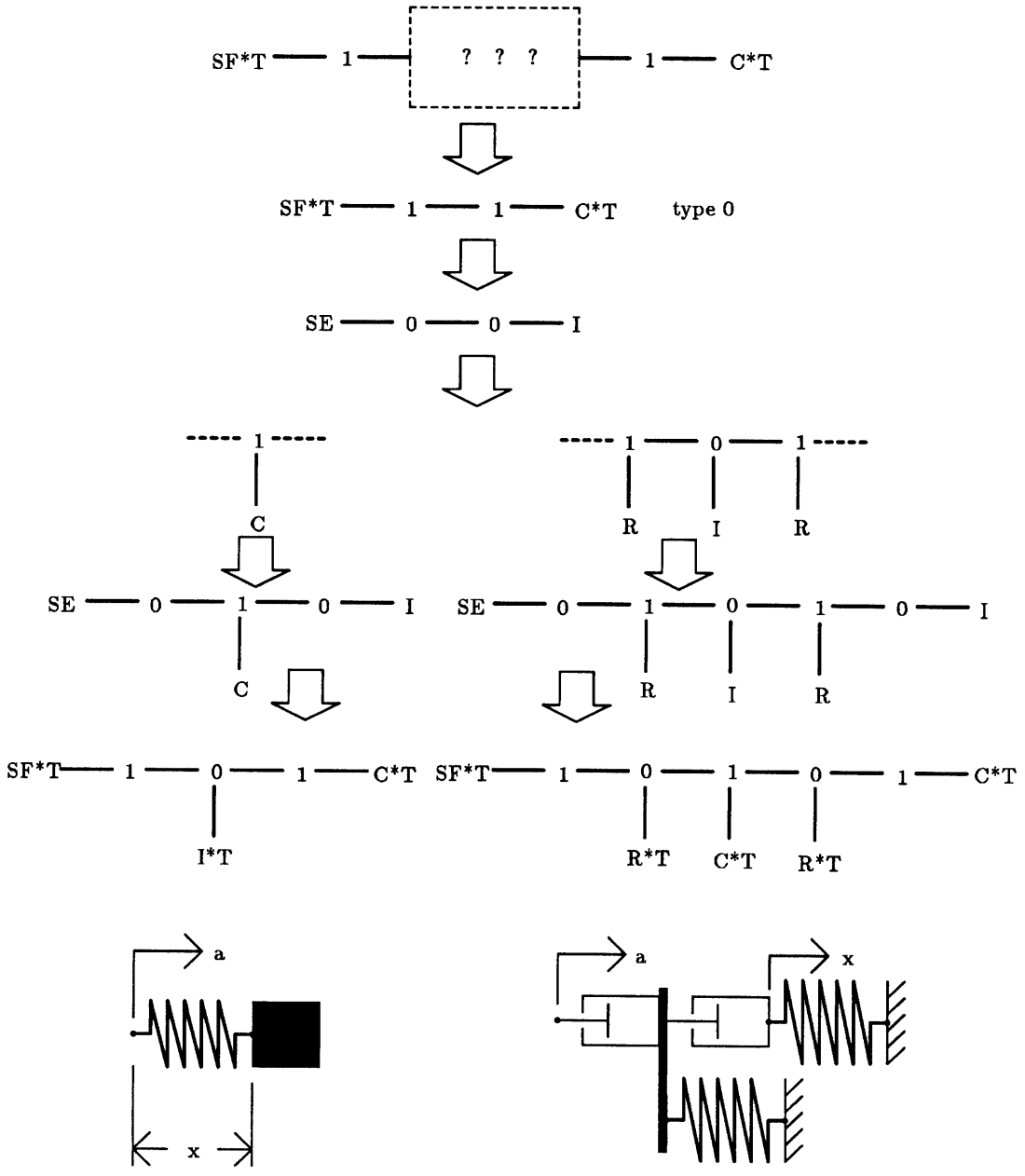

Figure 3.12: Accelerometer synthesis

domains and from finding synthesis techniques in those domains. In many ways human engineers have been successful because they have been able to find *some* solution to a problem. Synthesis techniques for even fairly simple engineering tasks can help engineers find a much greater number of feasible solutions.

3.5.2 Completeness of the Technique

My schematic synthesis technique is in a trivial sense incomplete– it will not generate all possible designs that meet the problem specifications. The incompleteness results from two factors. First, there are an infinity of designs that meet the specifications due to the fact that a pair of inter-medium transforming or gyrating elements can always be added to a design to create a new design. An example of this case is a back-to-back motor-generator pair added to an electrical device. Second, elements that do not change the nominal behavior of a design can always be added to a design to create a new design. An example of the second case is the addition of a redundant capacitance to a 1-port that already has a capacitive element. Because of this situation, completeness should be defined with respect to the compact representation of the design, after all of the extraneous elements have been removed, and the design has been converted to a generalized form without transformers and gyrators. Under these conditions the technique is complete. The completeness rests on the fact that the type number of a system is determined by the number of isolated 1—C's and 0—I's. This can be proven by the following argument. A positive type number indicates the number of times the input quantity has been differentiated. There are only two possible ways a quantity can be differentiated in this domain. Either a capacitance differentiates an effort to produce a flow, or an inertance differentiates a flow to produce an effort. Resistances serve to convert efforts to flows and vice versa. In my compact representation, the only sources are effort sources. The only way to differentiate an effort source is to connect it to a capacitance (an instance of

this case is a force source attached to a spring) or to connect it to a series resistance (to convert the effort to a flow) and then to a parallel inertance (to differentiate the flow into an effort). These two cases correspond to single isolated capacitances and inertances. Since the power spine of a device is a sequence of elements, this argument applies to an arbitrary number of differentiations. For each additional differentiation (increase in the type number) an isolated capacitance or inertance must be added. Because the synthesis technique produces all possible ways of modifying a design in order to increase the number of isolated capacitances and inertances in the graph, it produces a complete set of compact designs.

3.5.3 Why the Technique Works

My solution to the problem of schematic synthesis is possible because of some properties of the domain and also because of choices that I made in approaching the problem.

The right schematic representation

Bondgraphs are a good representation for the dynamic systems synthesis problem. Bondgraphs are formal and they make the right features of the design explicit.

Because bondgraphs are a formal language there is a well-defined way of composing descriptions and a way to decide if a description is well-formed. These properties are a necessary but not sufficient requirement for the development of design operators. The ideas of isolated capacitances and inertances, and the compacting transformation operators are built upon this formal foundation. In the case of the SISO dynamic systems synthesis problem, I did not have to develop this representation from scratch, but instead was able to exploit an existing representation language.

Bondgraphs also make the right design features explicit and suppress irrelevant details. At the level of semantic content, bondgraphs express only the constituent

lumped-parameter elements and their topology. The bondgraph does not express any physical information. Since the problem is to generate a schematic description, a representation that excludes all physical information allows the synthesis technique to focus on relevant information. There are several possible purely schematic representations in the SISO dynamic systems domain. Other choices are circuit diagrams or block diagrams. The key advantage of the bondgraph is that it converts graphs with loops in them to simple sequences of elements. This is possible because of the zero-junction, corresponding to an explicit identification of the generalized Kirchoff voltage junction; and because of the one-junction, corresponding to an explicit identification of the generalized Kirchoff current junction.

Power spine

One source of power in the schematic synthesis procedure is the ability to generate the set of candidate designs from which all possible solutions (containing fewer than three inter-medium transformations) must be derivable. Because the power spine is the minimal connection possible between the input and output graph chunks (for a particular set of media) and because any graph that produces a relationship between two n-ports must have a connection between those n-ports, all possible graphs that have the right behavior must be derivable through augmentations to the power spine. This domain property allows the system to generate designs that are *close* to a final solution— all that remains is to find appropriate modifications.

Ability to abstractly characterize necessary properties

Because of the ability to perform a compacting transformation that preserves the properties of a graph that contribute to its nominal behavior, an infinite set of graph instances can be compressed into a single minimal description. This compacting allows the design system to reason about any particular design by reasoning about

a minimal description of that design. Without the ability to abstractly characterize a particular design, the system would have to reason about millions of special cases; with the ability to abstract, the entire domain knowledge can be expressed by a few brief statements.

3.5.4 Extensibility

The ideas in this chapter have been directed at a very specific synthesis problem. This subsection deals with the problem of extending the ideas within the dynamic systems domain and to other domains.

Extension within dynamic systems domain

The devices that can be synthesized with the technique described in this chapter are limited in several ways: they are single-input, single-output; there is no feedback or amplification; and they can not be specified to have specific dynamic response characteristics. Extension of the solution technique to erase these limitations requires either an extension of the representation language, an extension of the synthesis procedures or both.

The concepts of feedback and amplification are necessary to describe many devices like controllers, power supplies or servo systems. In order to describe these systems the representation language must be extended to include *active bonds* [Rosenberg83]. In this extended bondgraph language, signal flows (having negligible power flow) can control a gain on an active bond. This extension allows for the description of amplification and feedback. The language remains formal, and behavior can still be derived from the graphs. The complexity of the problem is however increased because of additional primitive elements.

Using this extended bondgraph representation would also require extensions to the synthesis procedures. Specifically, the connection property of the domain that

leads to the concept of a power spine is still valid, although the kinds of connections that are allowed are increased to include signal-only connections. Under these conditions the synthesis procedure may require the consideration as special cases each of several classes of devices— those with feedback, those with hidden sources, those without any signal flows.

If the problem were extended to address multiple-input multiple-output systems, the synthesis procedure would become more complex. The power spine concept no longer is as strong as in the SISO case. There are now many possible ways for the multiple inputs and outputs to be connected together other than a single sequence of idealized transforming and gyrating elements. Prabhu [Prabhu88] has explored some of the issues in the multiple input and output problem.

The problem addressed in this chapter deals only with the nominal behavior of a dynamic system— I have not considered dynamic issues like bandwidth. To some extent these issues can be addressed by optimizing parameter values numerically. In other cases, filtering elements or damping must be added to the designs to achieve the specific numerical specifications desired. The synthesis procedure in this chapter is however a necessary first step towards meeting more detailed numerical design specifications. Any design that meets a detailed numerical specification must also meet the specification of the nominal behavior of the device. So, in this case the work in this chapter is directly applicable to a more specific synthesis problem.

Extension to other domains

The three major segments of the solution to the dynamic systems synthesis problem are: development/selection of a schematic language, a procedure for generating candidate designs, and a procedure for categorizing and altering designs falling into different classes of behavior.

The particular schematic language I use in the dynamic systems domain can not

be extended to any other domain, although the properties of bondgraphs that make them a good choice for the dynamic systems problem can be used as guidelines for selecting schematic languages in other domains.

The power spine idea in the dynamic systems domain has incarnations in other domains as well. In the domain of material handling systems (ie. design of conveyers) there must be a connection between input and output. In the domain of engineering structures (beams, brackets, supports, struts) there must be a connection between input and output. In fact in the structures case, perhaps the candidate designs could be maximal connections— a structure that occupies all of the possible space between the input and output points of the structure. Then, any valid design might be derivable from material removal operations on this candidate design.

The classification and modification procedures in the dynamic systems domain are problem specific; however, the general idea is applicable in many other domains. Imagine for example, a synthesis problem in the shaft and bearing system domain. A minimal characterization of the system in terms of adjacent surfaces could be derived in order to compute the degrees of freedom of the device, or to determine whether or not the device could be assembled.

3.5.5 Lessons from the Dynamic Systems Example

I picked the dynamic systems problem as a tractable first step towards developing principles for computational tools for conceptual design. I hold this problem and solution up as an instance of a solution to a schematic synthesis problem that has desirable properties and that can lead to better solutions to problems in other domains. The following items explain the lessons that I think can be derived from this example.

- *The dynamic systems domain is carefully bounded.* The devices are single-input single-output, there are no hidden sources or signal flows, and specifications are

made in terms of nominal behavior not numerical properties of the frequency response. Careful bounding of the scope of a problem allows for meaningful analysis of the solution technique and provides hope of tractability. Bounding the problem is an important first step in any domain.

- *The bondgraph representation is a formalism.* This means that it is approachable with known analytical tools and its properties can be well-understood. Such representations are limited in their expressiveness but also have well-defined properties. Finding formalisms for problems is a subgoal of developing powerful solution techniques.

- *A key issue in any design problem is controlling the complexity of the solution space.* In the dynamic systems domain, the primary tool for controlling complexity is to compress many instances of schematic descriptions into a single abstract characterization. This approach is valid in many other domains. Compression and abstraction should be one of the primary goals in the development of any design problem solving technique.

3.5.6 Implementation

I have written a computer program that implements a solution to the schematic synthesis problem described in this chapter. The program was written to clarify the thinking involved in devising a solution technique, and to that extent was quite useful. This subsection explains two issues associated with the program: its deviation from the theory presented in this chapter, and an explanation of one of the program modules— a Macsyma-based bondgraph equation derivation program.

Deviation from theory

Because the program was written primarily as a way to solidify my conceptual thinking about this schematic synthesis problem, it evolved over several iterations. The final version of the program does not exactly match the solution technique presented in this chapter. The actual program deviates from the theory presented in this chapter in one major way: it relies on a set of debugging rules to modify the candidate designs rather than performing the compacting transformation and looking for the appropriate number of isolated 1—C's or 0—I's. An example debugging operator is: *If the output is a flow variable, the input is a flow variable, and differentiation is desired, add a series resistance and capacitance to ground* (a 0—R followed by a 1—C). These rules were derived by analyzing my own design reasoning while debugging deficient designs, and in fact correspond to the same operations as are derived by the technique presented in this chapter. The debugging rules are however much more complicated than the procedure using the concept of a compact representation and isolated inertances and capacitances. It was not until I began exploring the completeness of the debugging procedure that I realized that there was a more compact way of characterizing the designs and that the domain knowledge could be expressed more concisely.

Equations from bondgraphs

One very useful byproduct of the implementation is a program for deriving equations from bondgraphs. This is a tedious process when performed manually. Previously, two programs had been written (Enport and Dart) [Rosenberg83, Hood87] for deriving equations from bondgraphs, but in each case the numerical values for the bondgraph parameters had to be specified and the equation solving was performed numerically. I was able to exploit the symbolic mathematics capabilities of MACSYSMA [Macsyma86] in order to write a program to derive the equations

symbolically. This process has not been previously automated.

Function Sharing to Generate Efficient Physical Descriptions

Chapter 4

Overview

The result of the synthesis techniques in Chapter 3 is a schematic description of a design. The next step is to generate an efficient physical implementation. A modular physical description corresponding to the schematic description can be generated by plugging in a structural element from a library of possibilities for each idealized functional element. The resulting physical description is overly complex and contains too many elements. The *function sharing* procedure described in this chapter is one way to transform this modular and complex physical description into an efficient physical description.

Function sharing in mechanical design is the simultaneous implementation in an artifact of several functions by a single structural element. Consider for example the difference between the devices shown in figure 4.1. Although the two are functionally similar, the upper device is much more efficient because each structural element implements several functions. This chapter describes how function sharing can be used in a computational design procedure that produces efficient designs from modular

designs.

4.1 INTRODUCTION TO FUNCTION SHARING

4.1.1 The Concept Of Function Sharing

In synthesizing a design, engineers represent an artifact in terms of its constituent functional elements, its *schematic description.* Designers must also represent an artifact in terms of its constituent structural elements, its *physical description.* In the case of computational design systems, these representations will often be explicit and distinct, while in human design systems the representations are often informal and interleaved. Given a physical and schematic description of a device, there will be a correspondence between elements in each. Function sharing is viewed as a mapping from more than one element in a schematic description to a single element in a physical description. In the case of the devices shown in figure 4.1, some of the functional elements in a possible schematic language might be *cutter*, *actuator*, *finger interface*, *bearing*, and *key ring interface.* The structural elements in a physical language might be the geometrical descriptions and material properties of each separate part.

4.1.2 Function Sharing is Important

Function sharing is important for three reasons. First, designs that exhibit function sharing are in most respects better designs than the non-function-sharing counterpart. Second, a design simplification procedure that results in function sharing allows the designer to think in a modular, decomposed fashion, with the option of subsequently processing a design to make it more efficient. Third, function sharing is one of the sources of novelty or interestingness in mechanical design. So, an understanding

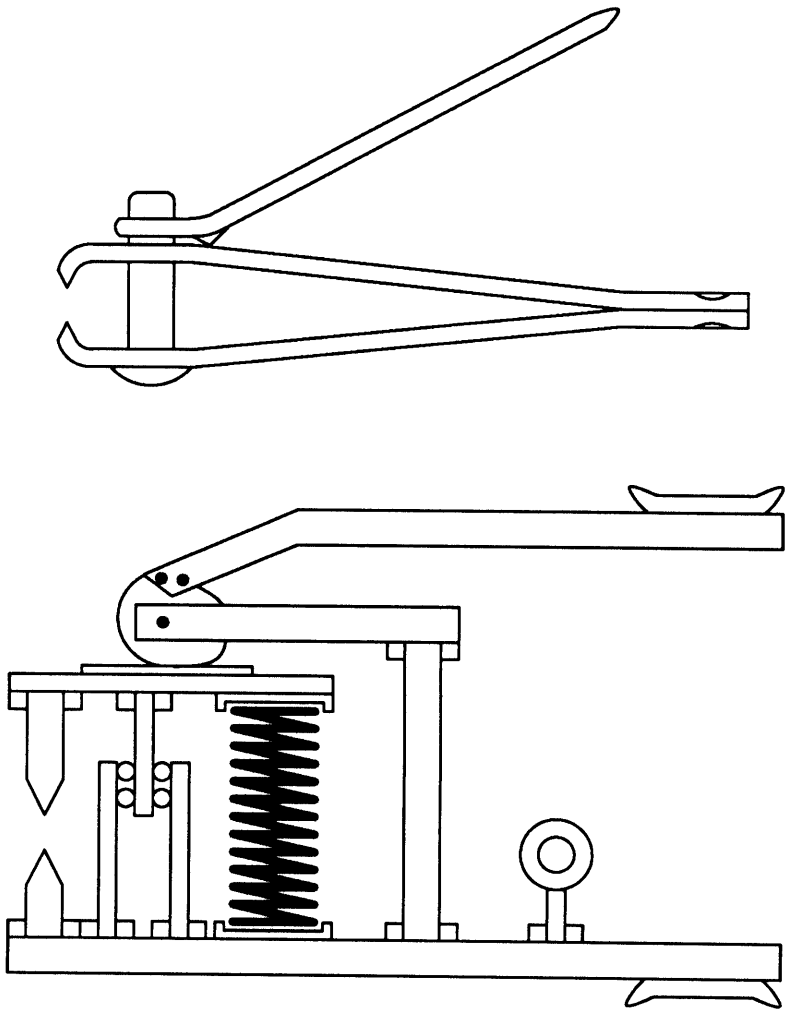

Figure 4.1: Example of function sharing

of function sharing is in part an understanding of novelty.

Function sharing as a desirable design attribute

If automobiles were designed without function sharing they would be relatively large, heavy, expensive and unreliable. But because elements like the sheet-metal body perform many functions (electrical ground, structural support, aerodynamic faring, weather protection, and aesthetics among others) automobiles can be manufactured relatively inexpensively and can perform relatively well. In fact, most mass-produced devices exhibit function sharing. The leads on integrated circuit packages are electrical conductors, thermal conductors and attachment points for the circuit board. The structure of a styrofoam coffee cup holds liquid, insulates, provides a graspable surface, and rests stably on a table.

As a general rule, mass-produced designs that exhibit function sharing are better than those that do not exhibit function sharing. Function sharing contributes to design quality in two major ways. First, designs exhibiting function sharing will generally be less expensive to produce than designs that do not exhibit function sharing, as a result of fewer parts, easier assembly and less required adjustment and maintenance. Second, designs that share function generally perform better than those that do not, resulting from decreased size and weight. On the other hand, function sharing is generally a poor design strategy for research devices and prototypes where debugging, adjustment and diagnosis are very important, because function sharing often causes device performance parameters to be coupled in complicated ways to device physical parameters. This coupling makes adjustments and diagnosis difficult. While function sharing may be an important design strategy for a ballpoint pen or an electric drill, it is a potentially disadvantageous strategy for a steel rolling mill or a laboratory instrument. For the purposes of this document, I assume that a system that can generate design descriptions that exhibit function sharing is a valuable tool.

From modular designs to efficient designs

Designing devices at the schematic description level is easier than designing at the physical level. This is because schematic representation languages are generally more restrictive than physical representation languages. This restriction focuses the designer on a small set of functional primitives, without clouding the preliminary design problem with how the functional elements may eventually be implemented. Since the vocabulary of functional elements in most domains will be much smaller than that of the structural elements, the schematic design space is also much smaller than the physical design space.

One approach to generating a physical description from a schematic description is through a one-to-one mapping from each functional element to an element in the physical description. This mapping can be accomplished by selecting a structural element from a library of possibilities that has the nominal function of its corresponding functional element in the schematic description. This procedure can be thought of as generating a starting point for the function sharing procedure. Once this modular physical description is generated, a function sharing procedure can convolve the description into an efficient and highly integrated design. The function sharing procedure described in this chapter is one approach to generating highly integrated designs. In tandem with a procedure for generating schematic descriptions of designs (chapter 3), and from them modular physical descriptions, the procedure allows a design system to transform a behavioral specification into an efficient physical description.

Novelty in mechanical design

I am interested in innovative design. Function sharing is part of the perception of novelty, simplicity, or interestingness in a device. To the extent that function sharing can be understood, one of the components of novel mechanical design can

be understood. (Appendix D describes some of my observations of design attributes that contribute to interestingness).

4.1.3 Why Function Sharing is Possible

Of the infinite physical properties of a structural element, only a small set are relevant to the behavior the designer intends for that element. In addition to the primary properties of a structural element that provide that element's intended function, there are many secondary properties that are incidental to those that implement the intended function. The key idea that allows function sharing to be achieved by a design procedure is that these secondary properties of structural elements can be exploited to eliminate redundancy. By recognizing and exploiting secondary properties of one element, neighboring elements can be eliminated from the design.

For example, a modular design of an automobile would include a ground wire running from the tail light to the battery. By recognizing that there is already an element (the automobile body) that travels from the tail light to the battery, and that this element has the secondary property that it conducts electricity, the ground wire can be eliminated. In this example, this simplification is possible because steel sheet metal has many properties other than high stiffness, high strength, ductility, and low cost (ostensibly the reasons for using steel for car bodies). Performing this reasoning requires a physical representation of the design, and an ability to recognize secondary properties of regions of the physical description.

4.1.4 Summary of Function Sharing Procedure

The procedure for achieving function sharing consists of three steps:

1. A structural element is deleted from the physical description.

2. Alternative features in the physical description that can potentially implement the function of the deleted element are identified.

3. The identified features are modified to accentuate their desirable secondary properties.

This procedure is illustrated in figure 4.2. The initial design is a lamp fixture hung by a piece of link chain, through which the electrical cord is woven. In order to simplify the lamp design, an element (the chain) is deleted from the design. Next, the tensile properties of the electrical cord are recognized. Finally, these properties are exploited by making the cord thicker. Through this three step process, the design is simplified.

4.2 DOMAIN DESCRIPTION AND REPRESENTATION

4.2.1 Dynamic Systems

As a domain for exploring function sharing, I have chosen mechanical devices whose primary behavior can be described by a differential equation relating an input and output quantity and whose schematic description can be expressed as a network of lumped-parameter idealized elements. The computer program (discussed later in this chapter) that implements the function sharing procedure is further limited to devices that can be described with fluid-mechanical elements and mechanical-translational elements. Such devices include pressure gages, accelerometers, force transducers, and pneumatic cylinders. Examples of diverse designs that are not in this class are conveyer systems, peach pitters, ball bearings and computer displays (although these systems can be described with differential equations and networks, such descriptions do not characterize their primary behavior). I call this domain *dynamic systems.*

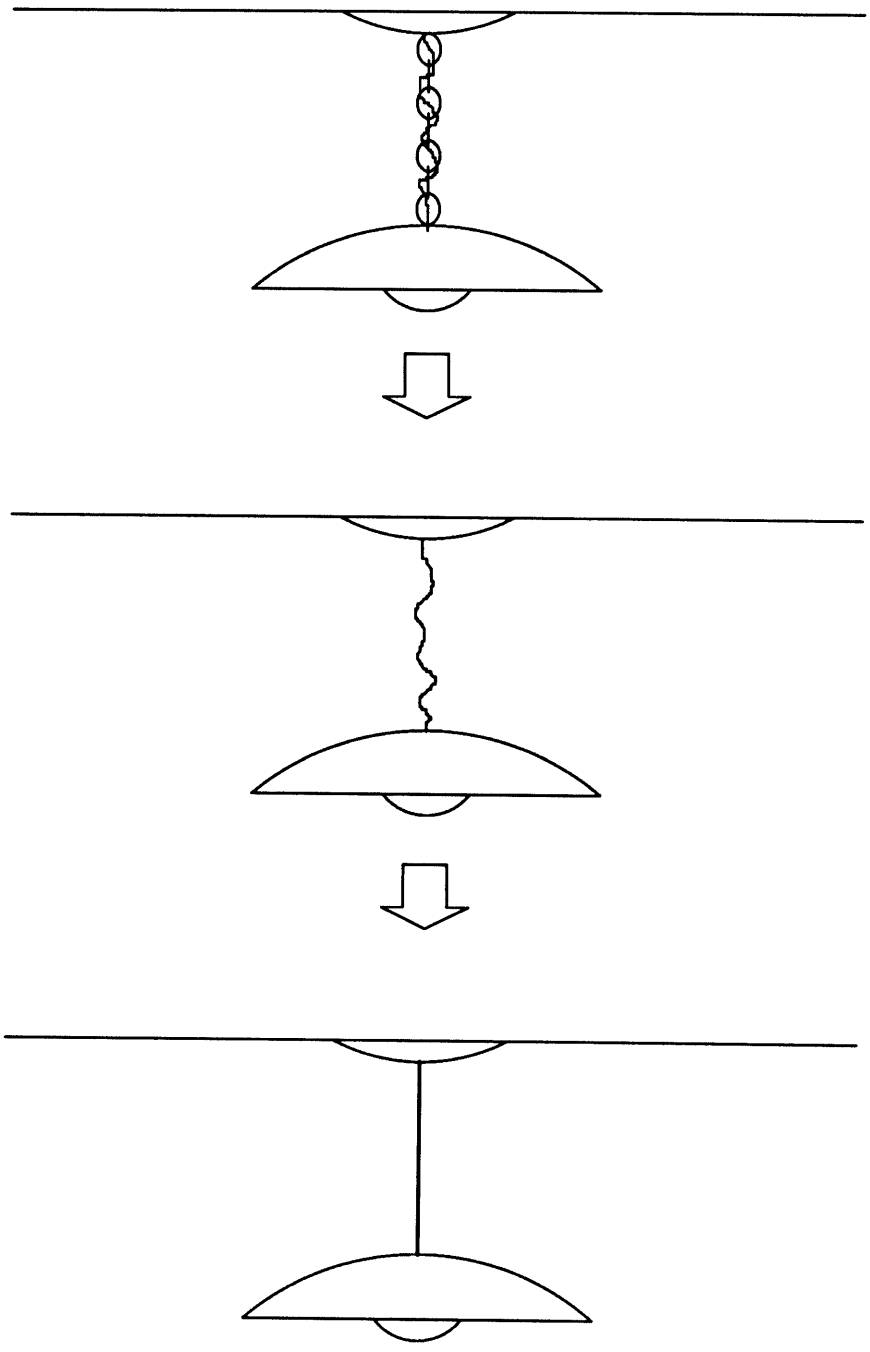

Figure 4.2: Illustration of function sharing procedure.

4.2.2 Physical Representation

I have chosen a two and one-half dimensional geometry to describe devices physically. In this representation, physical descriptions of designs consist of a collection of structural elements (these can be thought of as design components like piston-cylinders or springs), which are in turn built from orthogonally connected rectangular-prismatic sections of material (these rectangular-prismatic sections are the primitive physical building blocks of the system). Figure 4.3 shows the physical description of a piston-cylinder structural element. Figure 4.4 shows the top view of the physical description of a rate-of-pressure indicator containing the piston-cylinder structural element.

The two and one-half dimensional representation is required rather than a two-dimensional representation to allow fluid to flow past obstacles in the design. For example, if the piston-cylinder in figure 4.3 were only two-dimensional, the two cavities above and below the piston rod would not be in fluid communication and the piston would not function as expected. By adding an additional half dimension (thickness) to the design description, the upper and lower cavities can be connected as long as the piston rod is only half as thick as the cylinder.

In this two and one-half dimensional geometry, Newtonian physics applies as if the device were fully three dimensional if one imagines the device to be sandwiched between two infinitely stiff and strong, frictionless plates. This particular physical representation was chosen to simplify the computational geometry problems while still maintaining the applicability of Newtonian physics.

In addition to specifying the x and y locations and dimensions of each rectangular-prismatic section of the design, a limited set of material properties are also specified. The material of each section is specified to be either *stiff* or *flexible*, and of *high* or *low* density. Each section is also specified as either attached to or free from the imaginary plates that sandwich the design. Designs consisting of fluid-mechanical elements also include a working fluid specification (either *gas* or *liquid*). All design

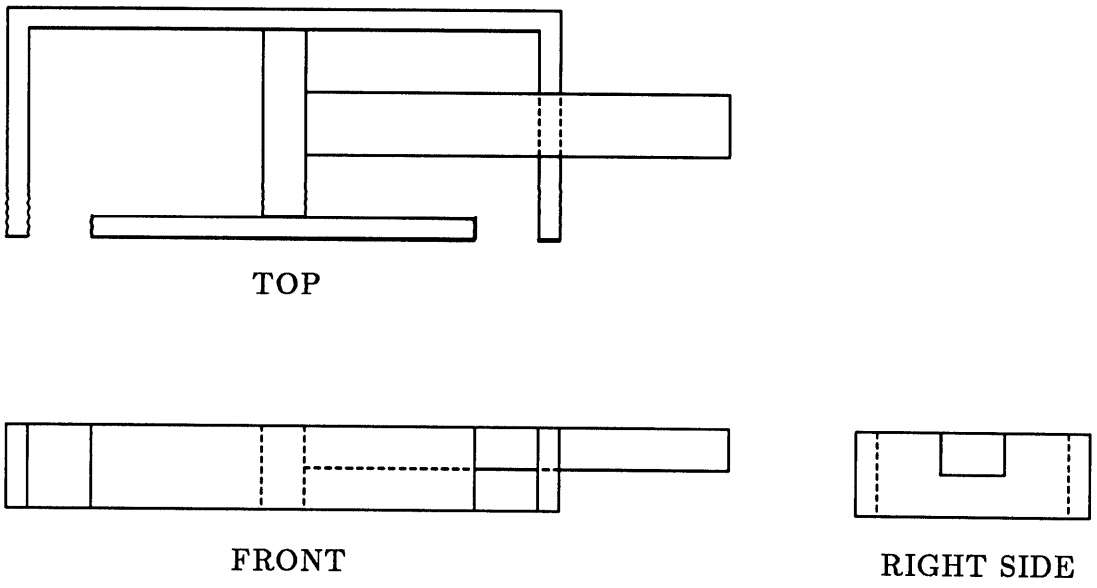

Figure 4.3: Example of structural element represented as a collection of orthogonally configured rectangular-prismatic sections.

descriptions contain a specification of an external environment for the design (also either *gas* or *liquid*).

4.2.3 Nominal Functional Specification

In the dynamic systems domain, the schematic description of a device can be expressed as a network of idealized lumped-parameter functional elements. The schematic description can be thought of as a generalized circuit diagram, so the behavior of the entire device can be derived from the behavior of the individual idealized elements and the network laws. In addition to specifying the strictly physical properties of a design, the function sharing procedure requires a specification of the intended function of each structural element. This specification is an explicit linking of functional elements from the schematic description to regions of the physical design description. A function specification includes the name of the function and one or two reference

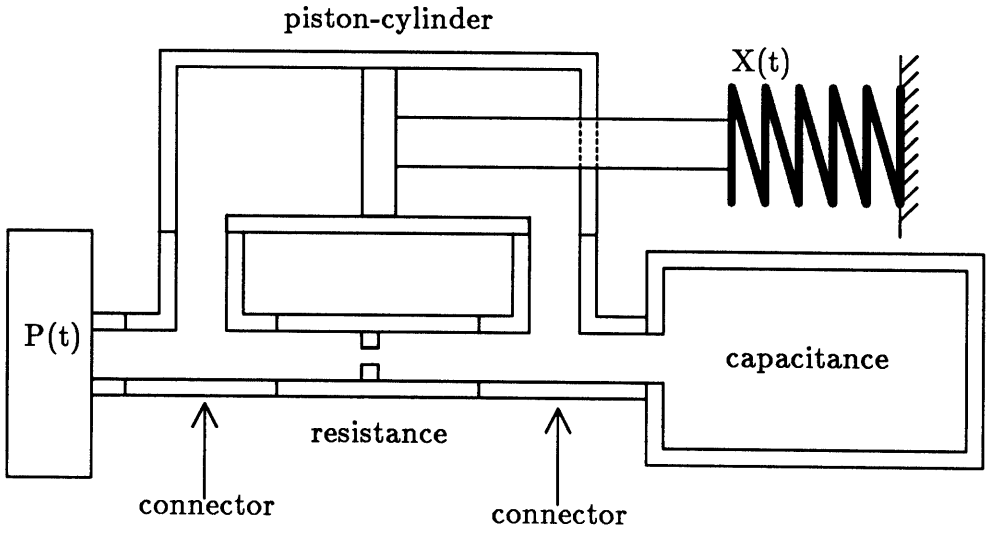

Figure 4.4: Example of physical description of a design.

points in the design. For example, the specification of a fluid resistance would indicate first, that the function is *fluid resistance*; second, that the resistance is from x=5 y=0 to x=10 y=0. The nominal functions of all structural elements in a physical description are specified similarly.

In addition to the lumped-parameter functions, two special functional elements are also necessary: *connection* and *ground.* These are used to specify how regions of the physical description connect the other functional elements together.

4.2.4 Example Physical Design Description

This subsection gives an example of a complete physical description for a simple pressure gage. Figure 4.5 shows the graphical version of the description, followed by the LISP structure that the computer implementation of function sharing operates on. Notice that the description is hierarchical: the design contains components and components contain sections. Notice also that the function of each structural element

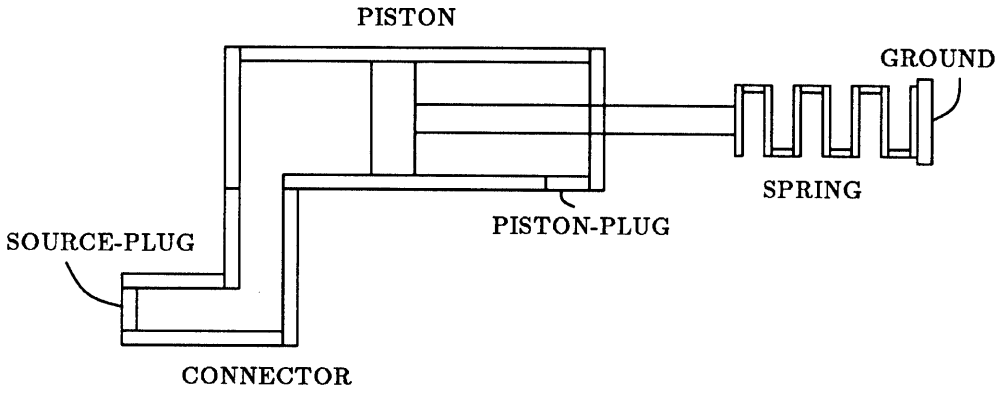

Figure 4.5: Complete physical description of pressure gage

(called a *component* in the LISP structure) is specified.

```
(DESIGN
  :NAME PRESSURE-GAGE
  :FLUID GAS
  :X-DIMENSION 173
  :Y-DIMENSION 50
  :COMPONENTS
    ((COMPONENT
       :NAME PISTON
       :X-DIMENSION 70
       :Y-DIMENSION 20
       :X-LOCATION 44
       :Y-LOCATION 30
       :WORKING-FLUID LIQUID
       :SECTIONS
         (SECTION
           :NAME SECTION-A
           :X-DIMENSION 2
           :Y-DIMENSION 20
           :X-LOCATION 0
           :Y-LOCATION 0
           :FLEXIBILITY STIFF
           :DENSITY HIGH
           :THICKNESS FULL
           :ATTACHED? T)
```

```
                .
                .
                .
      (SECTION
        :NAME SECTION-H
        :X-DIMENSION 44
        :Y-DIMENSION 4
        :X-LOCATION 26
        :Y-LOCATION 8
        :FLEXIBILITY STIFF
        :DENSITY HIGH
        :THICKNESS TOP
        :ATTACHED? NIL))
    :CONNECTIONS
      ((SECTION-C SECTION-D) (SECTION-D SECTION-E)
       (SECTION-E SECTION-F) (SECTION-F SECTION-A)
       (SECTION-G SECTION-H))
    :FUNCTIONS
      ((FN
         :TYPE FLUID-PORT
         :POINT-A-X 5
         :POINT-A-Y 20
         :POINT-B-X 47
         :POINT-B-Y 20)
       (FN
         :TYPE TRANS-PORT
         :POINT-A-X 70
         :POINT-A-Y 10
         :POINT-B-X 0
         :POINT-B-Y 10)))
(COMPONENT
  :NAME SPRING
  :X-DIMENSION 25
  :Y-DIMENSION 10
  :X-LOCATION 114
  :Y-LOCATION 35
  :WORKING-FLUID NIL
  :SECTIONS
    ((SECTION
       :NAME SECTION-A
         .
         .
         .
```

```
            ))
      :CONNECTIONS
        ((SECTION-A SECTION-B) (SECTION-B SECTION-C)
         (SECTION-C SECTION-D) (SECTION-D SECTION-E)
         (SECTION-E SECTION-F) (SECTION-F SECTION-G)
         (SECTION-G SECTION-H) (SECTION-H SECTION-I)
         (SECTION-I SECTION-J) (SECTION-J SECTION-K)
         (SECTION-K SECTION-L) (SECTION-L SECTION-M))
       :FUNCTIONS
         ((FN
            :TYPE TRANS-CAPACITANCE
            :POINT-A-X 0
            :POINT-A-Y 5
            :POINT-B-X 25
            :POINT-B-Y 5)))
    (COMPONENT
      :NAME CONNECTOR
            .
            .
            .
             )
    (COMPONENT
      :NAME SOURCE-PLUG
            .
            .
            .
             )
    (COMPONENT
      :NAME PISTON-PLUG
            .
            .
            .
             )
    (COMPONENT
      :NAME GROUND
            .
            .
            .
             )
  :CONNECTIONS
    ((SOURCE-PLUG CONNECTOR) (CONNECTOR PISTON) (PISTON PISTON-PLUG)
     (PISTON SPRING) (SPRING GROUND)))
```

4.2.5 Function Sharing Problem Definition in the Dynamic Systems Domain

The input to the function sharing procedure is a physical description of a device augmented with a specification of the intended function of each element; the output is also a physical description. The objective of the procedure is a simplification of the initial physical description that reduces the number of structural elements.

4.3 FUNCTION-SHARING PROCEDURE

The function sharing procedure consists of three steps: 1) a structural element is deleted from the design. 2) geometrical features that can potentially implement the function of the deleted element are found. 3) modifications are made to the design to exploit the secondary properties of the features found in step 2. After illustrating these steps with an example, I explain the details of each step for the dynamic systems domain.

4.3.1 Example of Function Sharing Procedure

Figures 4.6 and 4.7 display an example of the function sharing procedure in the dynamic systems domain. Each description in the figure corresponds to a different state during the simplification. First, the fluid resistance is deleted from the design. Next, a potentially resistive feature is found with respect to points A and B. In this case the feature is a path between A and B passing between adjacent edges. This feature is modified to create a clearance between the adjacent edges, such that resistance is established between A and B.

Second, the fluid capacitance is deleted from the design and a feature that can

provide fluid capacitance is found with respect to point A. In this case, it is simply a cavity adjacent to point A. Finally this feature is modified by extending the cylinder length, thereby providing the capacitance of the deleted element.

4.3.2 Deleting a Structural Element

The physical description of the design is represented as a collection of structural elements. The first step in the function sharing procedure is to remove one of these structural elements. Removal of a structural element from a design may cause some side effects. For example, removing a fluid element from a design may cause leakage. Removing a mechanical translational element may cause parts of the design to become disconnected. Because of these side effects, the design must also be repaired after the deletion step. In addition to removing the element and repairing the design, the deletion step must establish reference points in the design with respect to which alternative features should be found.

Device repair

Deleting fluid elements includes several possible cases: a *dangling element* (only connected to the design at one point), a *sandwiched element* (one that constitutes the only connection between two regions of the design), and a *bridging element* (one that constitutes one of several connections between two regions of the design).

An essential part of repairing a design after a fluid element is deleted is the modification of connectors. Connectors are special elements whose only purpose is to provide a fluid connection between other elements. When one of the elements attached to a T-type connector is removed, then the connector must be modified to be a straight or L-type connector. When an element is removed from a straight or L-type connector, then a plug is substituted for the connector.

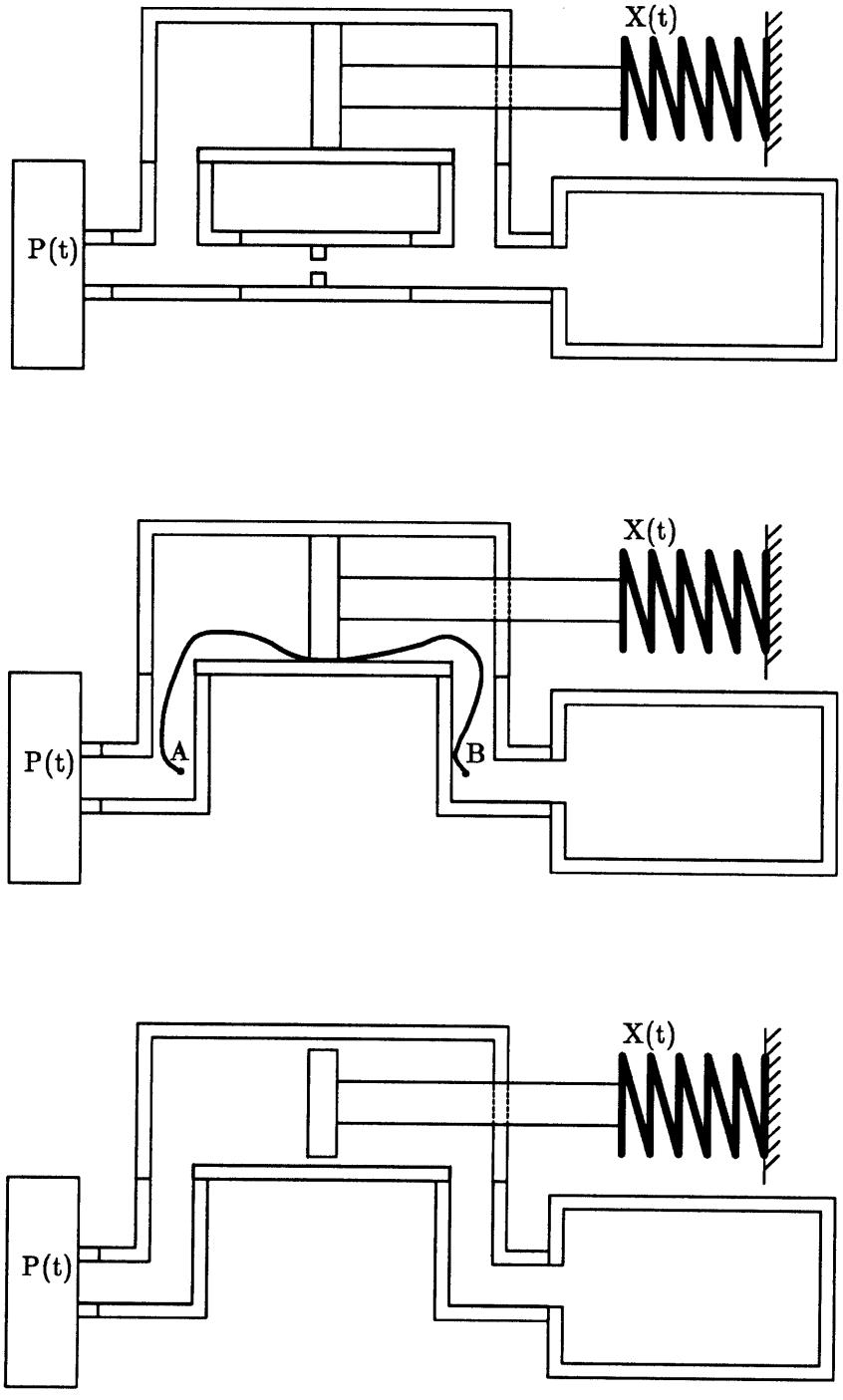

Figure 4.6: Eliminating the fluid resistance.

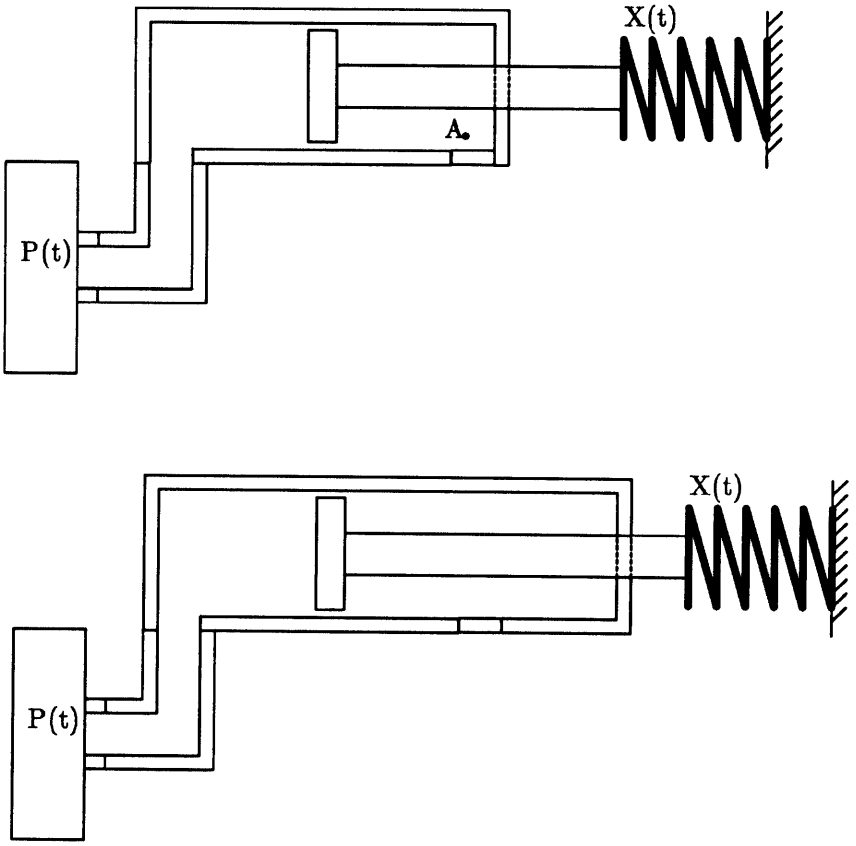

Figure 4.7: Eliminating the fluid capacitance.

Dangling elements are deleted by removing the element from the design and replacing the element with a plug. Sandwiched elements are deleted by removing the element and connecting the two disconnected regions of the design to each other. Bridging elements are deleted by removing the element and modifying the connectors to which the element was attached, or establishing plugs in the cases where the element was attached directly to other elements.

Deleting mechanical elements also involves several special cases similar to the fluid cases except that leaks are not a problem. There are dangling elements, sandwiched elements, and bridging elements.

Dangling elements are deleted by removing the element. Sandwiched elements are deleted by removing the element and connecting the disconnected regions of the design at the points at which they were attached to the deleted element. Bridging elements are deleted by removing the element. In the special case where sandwiched elements are connected to a ground element, then in addition to connecting the orphaned ground element to the rest of the design, a design is also created in which the both the sandwiched element and the ground element are removed.

In the case of both fluid-mechanical and translational-mechanical elements, connecting disconnected regions of the design may require a rotation and translation of one of the regions. It is also possible that reconnection is not possible without causing geometrical interference between the regions. In this case, component deletion fails and the system can not simplify the design.

New reference points

The deleted structural element corresponds to a functional element in the schematic description. This functional element can be thought of as being approximated in the physical description with respect to some reference point(s). For example, a fluid

resistance is defined with respect to two points in the fluid flow. A translational mass is defined with respect to a single point. So, for every deleted element there will be one or two corresponding reference points in the physical design description. These reference points are those specified in the descriptions of the deleted structural element.

When an element is deleted its associated reference points may have to be moved. For example, the reference points associated with a bridging fluid resistance will be moved to the center of the connectors to which the resistance was attached. When regions of the design are translated and rotated with respect to each other, the reference points must also be translated and rotated.

4.3.3 Recognizing Alternative Features

After deleting the structural element to be eliminated, the next step is to find alternative features in the physical description that can potentially implement the function of the deleted structural element.

Organization of function sharing knowledge

I have approached the feature finding task as a physical property recognition problem—identifying one of a set of known configurations of structural primitives that can be modified such that they approximate the relevant function. For example, fluid resistance can be derived from a narrow passage between two edges, a long narrow channel through a solid region, or an orifice in a plate. For each of these ways of implementing resistance there is a physical feature that could be potentially modified to achieve the resistive function. For example a path between two reference points that passes between two adjacent but detached edges could be resistive if a clearance were established between the edges. A path between two reference points

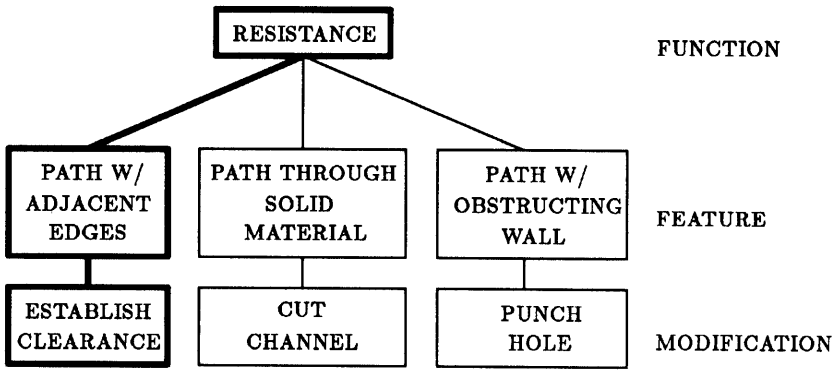

Figure 4.8: Organization of function sharing recognition and modification procedures.

that is obstructed by a solid wall could be resistive if a hole were punched in the wall. These relations constitute the function sharing knowledge base. The relations for fluid resistance are shown in figure 4.8. The relations for the other functions in the domain are shown in figures 4.9 and 4.10.

Recognition procedures

The procedures for recognizing any particular physical feature are ad hoc and specific to my two and one-half dimensional geometrical representation. For example finding a path between two points obstructed by a wall involves first finding all internal walls in the design, and then performing a search along section faces from each reference point to each side of the internal wall. For the most part these procedures are straightforward, if somewhat tedious, computational geometry problems.

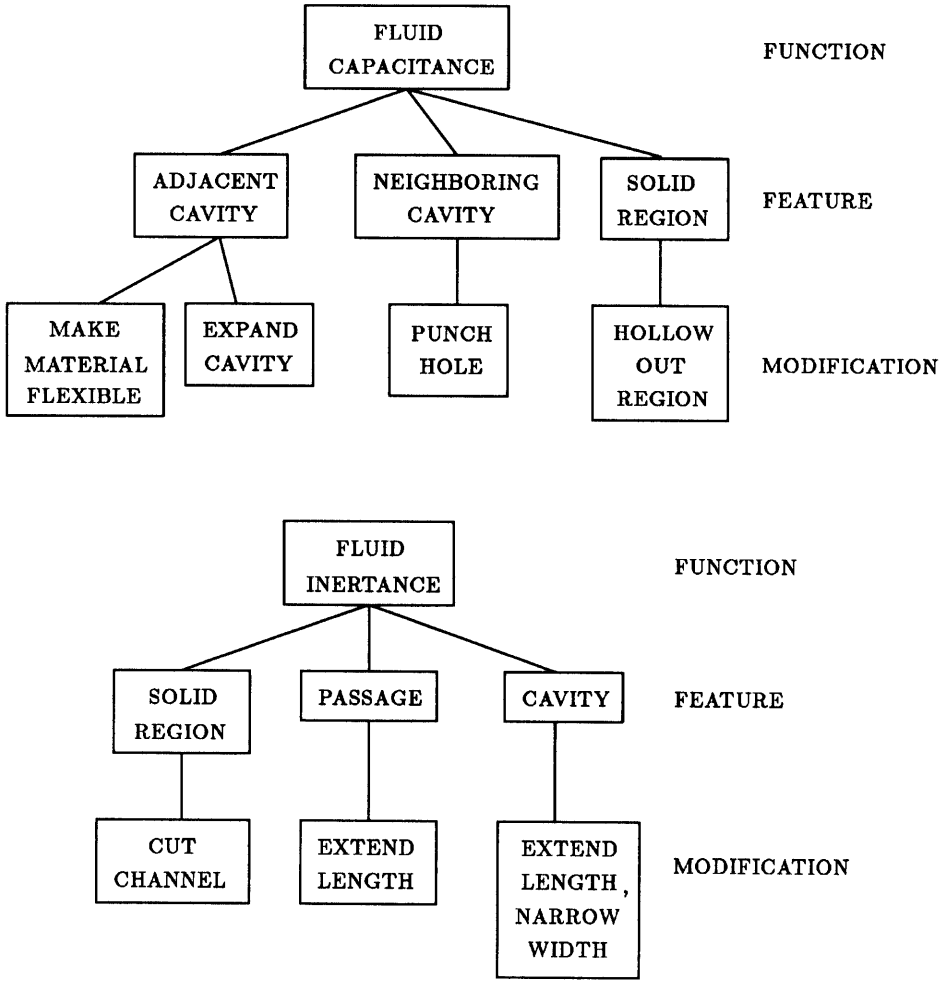

Figure 4.9: Knowledge for other functions in dynamic systems domain.

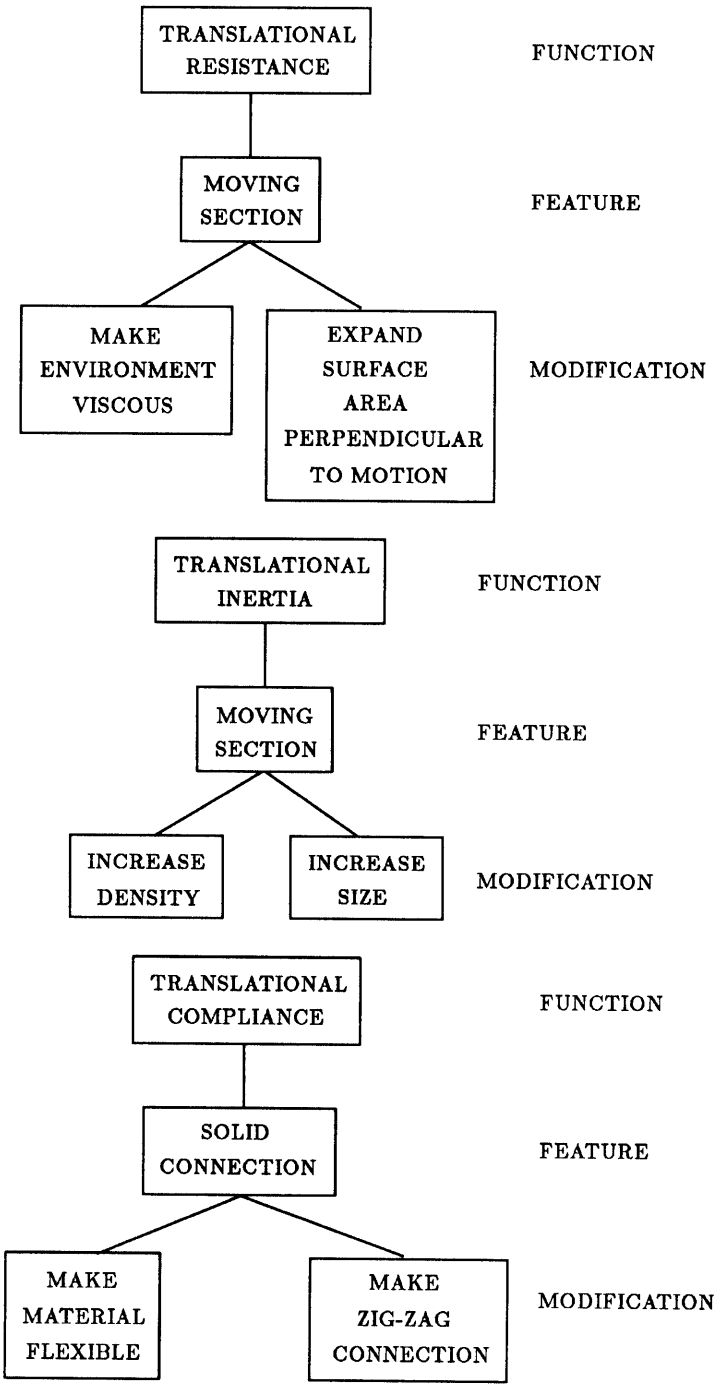

Figure 4.10: Knowledge for other functions in dynamic systems domain (continued).

4.3.4 Modifying Alternative Features

Once the potentially useful features are found, the final step is to execute the modification operators associated with each feature. In the case of the fluid resistance for example, if the feature found by the recognition procedure were a path obstructed by a wall, the modification would be to punch a hole in the wall. Modifications are executed only if the modification would not cause geometrical interference between different regions of the design. Geometrical interference is defined as the situation in which two sections of material share some of the same two and one-half dimensional space.

The modification procedures, like the recognition procedures, are ad hoc. As an example, punching a hole in a section involves removing the existing section, replacing it with two new shorter sections with a gap between their ends, and updating the component connection list. Punching a hole in my geometrical representation is actually equivalent to creating a gap.

4.3.5 Multi-Function Elements

The discussion of the function sharing procedure in the previous three subsections assumed for clarity that the structural element to be replaced had only one nominal function. In cases where a structural element corresponds to more than one functional element, some modifications to the recognition procedures must be made. These differences can be understood by considering two special cases: two parallel functions and two series functions.

In the parallel function case a structural element implements two functional elements with respect to the same reference point or points. Under these circumstances, the recognition procedure must find a set of alternative features for *each* of the two

functions. This recognition step involves doing the recognition process twice, once for each function. For example, if a structural element implements resistance between point A and point B, and implements capacitance between point A and point B, then eliminating this element requires that *both* an alternative resistance and capacitance be found with respect to points A and B.

In the series function case a structural element implements two functional elements with respect to sequential reference points in the design. An example of this situation would be a resistance with respect to point A and point B, and a capacitance with respect to point B and point C. Under these circumstances, the recognition procedure must also find a set of features for each of the two functions. However, the recognition must be with respect to an intermediate reference point (B-prime) within the remaining design. Each possible B-prime (selected from all of the reference points that are associated with the functions provided by the remaining design) is tried as an intermediate reference point.

4.3.6 Control

My implementation of the function sharing procedure leaves the control to the user. Specifically, the user selects an element to eliminate and chooses from among several modified designs.

The control could also be automatic. In this case, the program would identify each structural element in the design that corresponds to a single functional element. A new copy of the design would be made for each such element found. The structural element associated with each copy would then be deleted. Next, an attempt would be made to find alternative features that could potentially replace the deleted element in each design. Another copy of each design would be made for each alternative feature that can implement the function of the deleted element. The modification

associated with each feature would be executed in the corresponding design. For each modified design, the procedure would be repeated. Because there will in general be several alternative features for each deleted element, the worst-case number of designs generated by this procedure is exponential in the number of structural elements.

4.4 IMPLEMENTATION AND RESULTS

This section discusses the computer program that implements some of the function sharing concepts.

4.4.1 Scope and Goals of the Implementation

There were three factors that motivated a computer program implementation of the function sharing ideas I have presented.

1. *Unearthing conceptual bugs.* Writing programs is a good way to eliminate fuzzy conceptual thinking. While an idea that has not been made precise is difficult to implement as a program, writing the program helps to make the idea precise. In fact the major motivation for writing a computer program in this project was to solidify my conceptual thinking.

2. *A tool mock-up.* The particular computer program that I wrote can also be thought of as a mock-up of a future conceptual design tool. The representation I use in the implementation is too impoverished to allow this particular program to be of much practical use, but hopefully the program does suggest ways in which a future tool might be employed.

3. *Exploring novel design.* One of the hypotheses for this project is that the unbiased application of physical-feature-based design operators in function sharing

leads to novel design. This hypothesis is not tremendously strong since novelty is very much a function of a human observer. However in one case, the program did surprise me in an interesting way, supporting the spirit of the novelty hypothesis.

4.4.2 Three Example Runs

This subsection shows the screen output of the program running three examples. Each figure shows the progression of the design simplification as the user instructs the program to eliminate structural elements. In each of the examples only one or two of the outcomes of the program are shown, although there in general will be many designs generated. The particular examples shown in this section illustrate the range of simplification operations that the program can perform. There is a brief textual explanation for each figure.

Accelerometer 1

Figure 4.11 shows the program trace as it simplifies an accelerometer. An accelerometer is a device that produces a displacement (or other output quantity) proportional to the acceleration of the device reference frame. In most cases accelerometers will consist of some sort of mass and spring. The initial design in figure 4.11 consists of a mass attached to a loop-like spring which is in turn attached to ground. The spring is the orthogonally-connected rectangular prismatic version of an oval spring. The program is directed to eliminate the mass element. First the mass is deleted from the design. Next the system searches for alternative potentially massive features with respect to the top center portion of the spring. The system finds that the uppermost section of the spring has massive properties, and accentuates these properties by increasing the section thickness.

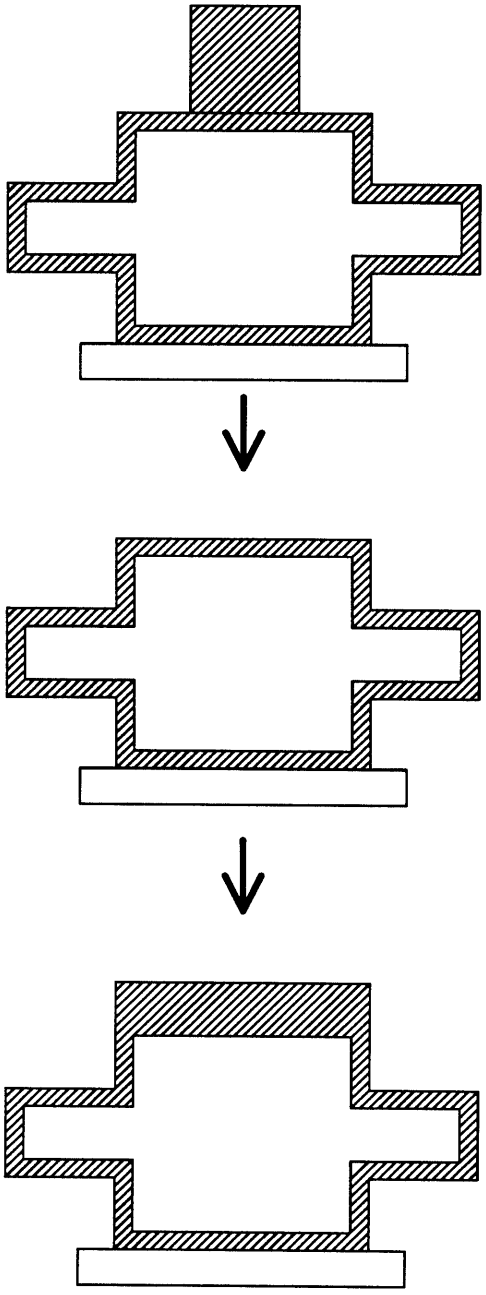

Figure 4.11: Accelerometer example 1.

Accelerometer 2

Figure 4.12 shows the program operating on another accelerometer design. In this case, the initial design consists of a cantilever beam attached to a mass and damper. The beam and damper are attached to ground. The program is instructed to eliminate the mass. It first deletes the mass, and then looks for massive features with respect to the end of the cantilever beam. It finds that the beam itself is massive, and accentuates this property by increasing the beam thickness. Next, the program is instructed to eliminate the damper from the design. First it deletes the damper and then looks for alternative damping features with respect to the end of the cantilever beam. It finds that the beam has some damping because it moves in a gas environment causing drag. The program accentuates this property by changing the design environment to a viscous fluid (indicated by the dots). In another branch of the simplification, the program extends the piston thickness in order to replace the mass.

Rate-of-climb

Figure 4.13 shows the program's simplification of a rate-of-climb indicator. This instrument consists of a diaphram with housing, a parallel capillary tube, and a capacitance at the housing side of the diaphram. The program is instructed to eliminate the capillary tube. In response, it deletes the tube and looks for alternative resistive features. The program finds that there is a path with an obstructing wall that can implement resistance if a hole is punched in the wall. The program makes a hole in the wall (one side of the diaphram). Next, the program is instructed to eliminate the capacitive chamber. It deletes the chamber and searches for alternative capacitive features. It finds that there is a cavity associated with the diaphram

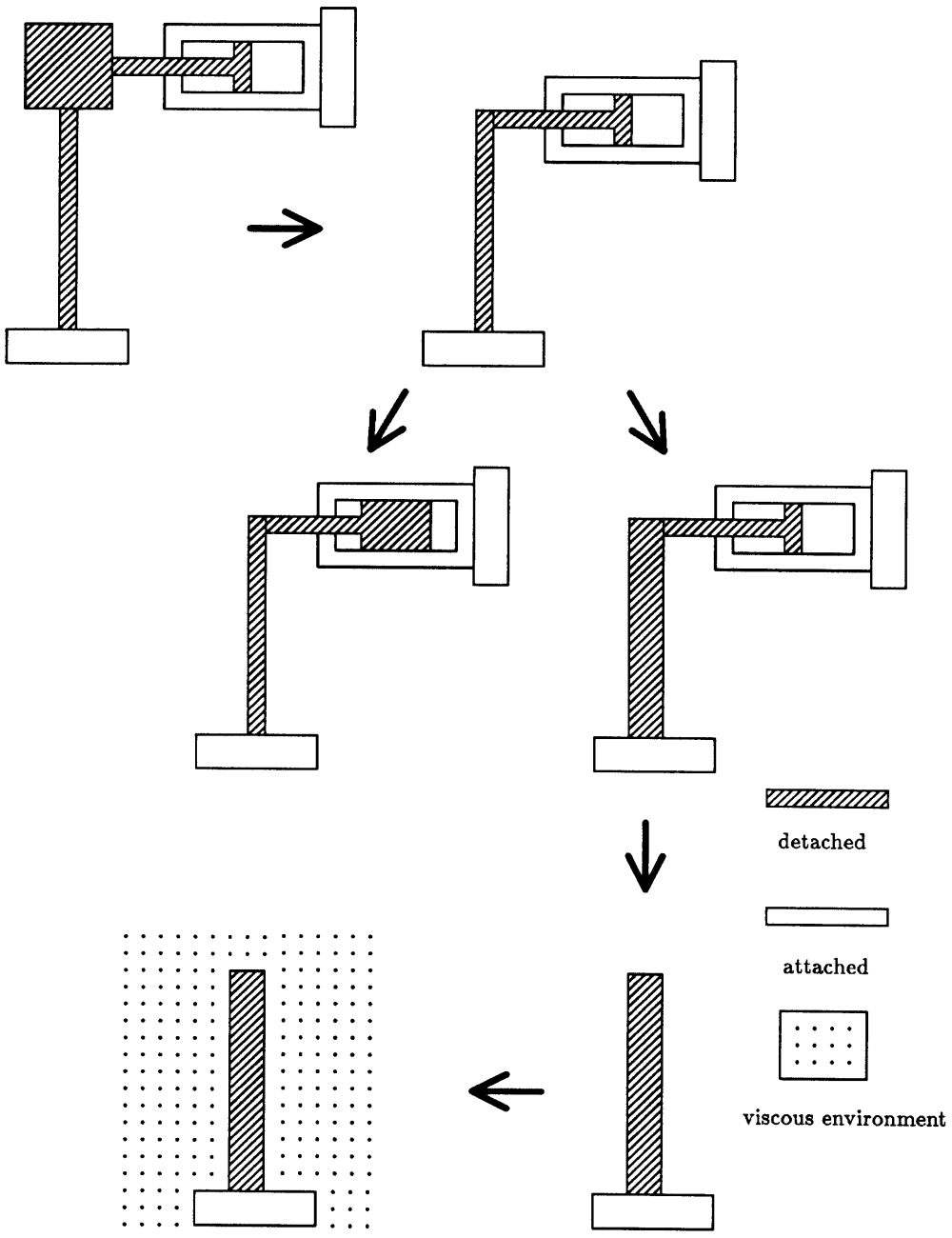

Figure 4.12: Accelerometer example 2.

housing and accentuates its capacitive properties by 1) detaching the housing walls (from the invisible plates that sandwich the design) and 2) making them of a flexible material (indicated by shading of sections).

4.4.3 Some Program Details

Programming bottlenecks

The total size of the function sharing program is 5500 lines of LISP code. The primary programming bottleneck in this implementation was the development of the component deletion procedures. Deleting a structural element from a design requires identifying which special case the deletion belongs to, removing the element, finding new reference points for the function of the element, and repairing and/or reconnecting the design. These processes are highly geometrical in nature, and require tedious programming. I expect that this same bottleneck will occur when these ideas are extended to fully three-dimensional geometry.

The feature recognition problem associated with each function in the dynamic systems domain was not difficult in the two and one half dimensional geometrical representation I used. I expect that feature recognition will be elevated to the primary programming bottleneck in the three dimensional case.

Scope of Implementation

The implementation includes the feature finding and recognition procedures for fluid resistance, fluid capacitance, translational mechanical inertia, and translational mechanical resistance. I did not implement the procedures for translational mechanical compliance or for fluid inertance.

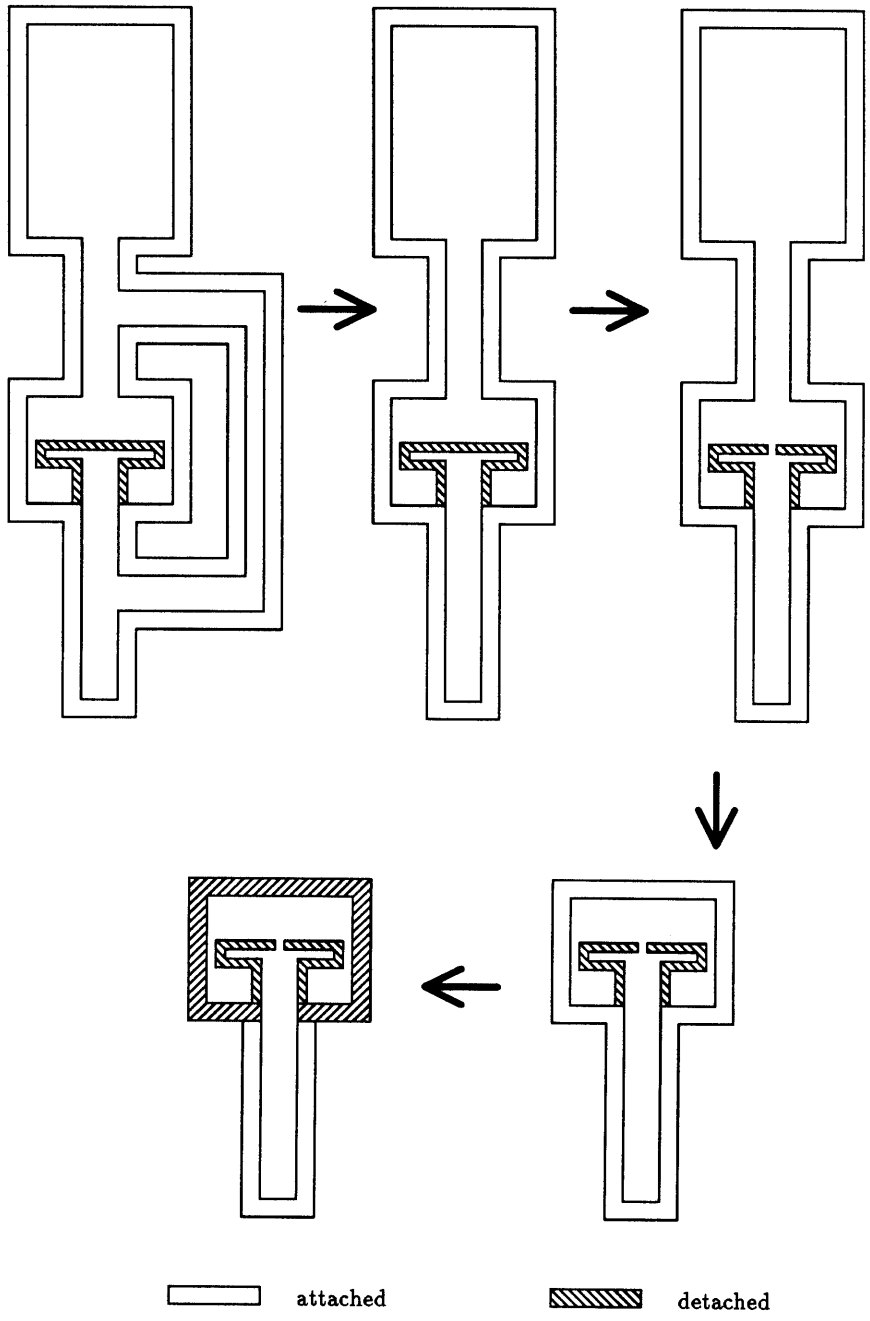

Figure 4.13: Rate-of-climb example 1.

4.5 ANALYSIS OF FUNCTION SHARING PROCEDURE

The preceding sections have not dealt with many of the subtleties involved in function sharing in the dynamic systems domain and in general. This section analyzes the procedure from several perspectives and discusses some of the issues associated with adapting the procedure to other domains.

4.5.1 Meaning of Physical Descriptions

The physical descriptions produced by the function sharing procedure can be thought of as parameterizations of a design description. In the case of a piston-cylinder, there are many possible parameters in the physical description that *may* be relevant to the design. The function sharing procedure is a way of identifying those parameters that should be considered in a particular design. For example, a piston-cylinder is normally parameterized by the piston area and the stroke length. There are, however, many other parameters that relate to the element– among them the thermal conductivity of the cylinder wall, the size of the input and output ports, or the mass of the piston. After the function sharing procedure operated on the rate-of-climb indicator example, two of these extra parameters were identified as important– the clearance between the piston and cylinder and the volume of one end of the cylinder. This identification resulted from the function sharing feature recognition and modification procedures. The procedure has identified a different and important set of parameters and the designer is alerted that the original design can be simplified if these parameters are considered when performing the detailed design and selecting dimensions.

4.5.2 Clobbering Existing Functions

Executing a modification to exploit a secondary property of some structural element may invalidate the primary function of that element. For example, consider the result of eliminating the fluid capacitance in figure 4.14. The expansion of the diameter of the resistance would impair its resistive function.

In my implementation of function sharing, all modifications that do not lead to geometrical interference are carried out. I leave the judgement as to whether or not existing functions have been impaired to the designer. To do otherwise would require explicit knowledge of why a particular element functions the way it does, or very sophisticated analysis tools that can determine that a component no longer performs as specified. If the representation of each structural element also contained a set of constraints on allowable modifications to its structure, then some clobbering modifications could also be avoided.

4.5.3 Design Knowledge Level

There are many alternative approaches to expressing function sharing design knowledge. I chose the physical feature level. This represents a trade off between efficiency and expressiveness. For example, figure 4.15 illustrates the variety of ways one might represent the concept of fluid resistance in a design system.

The simplest approach is to identify particular structural elements with particular functional elements. I call this the *component level.* This approach constrains the design system to using a particular component every time resistance is needed. Alternatively the system could represent resistance as being implemented by a variety of generic physical features like holes, channels, or porous materials. I call this the *feature level.* This approach gives the system more freedom to construct

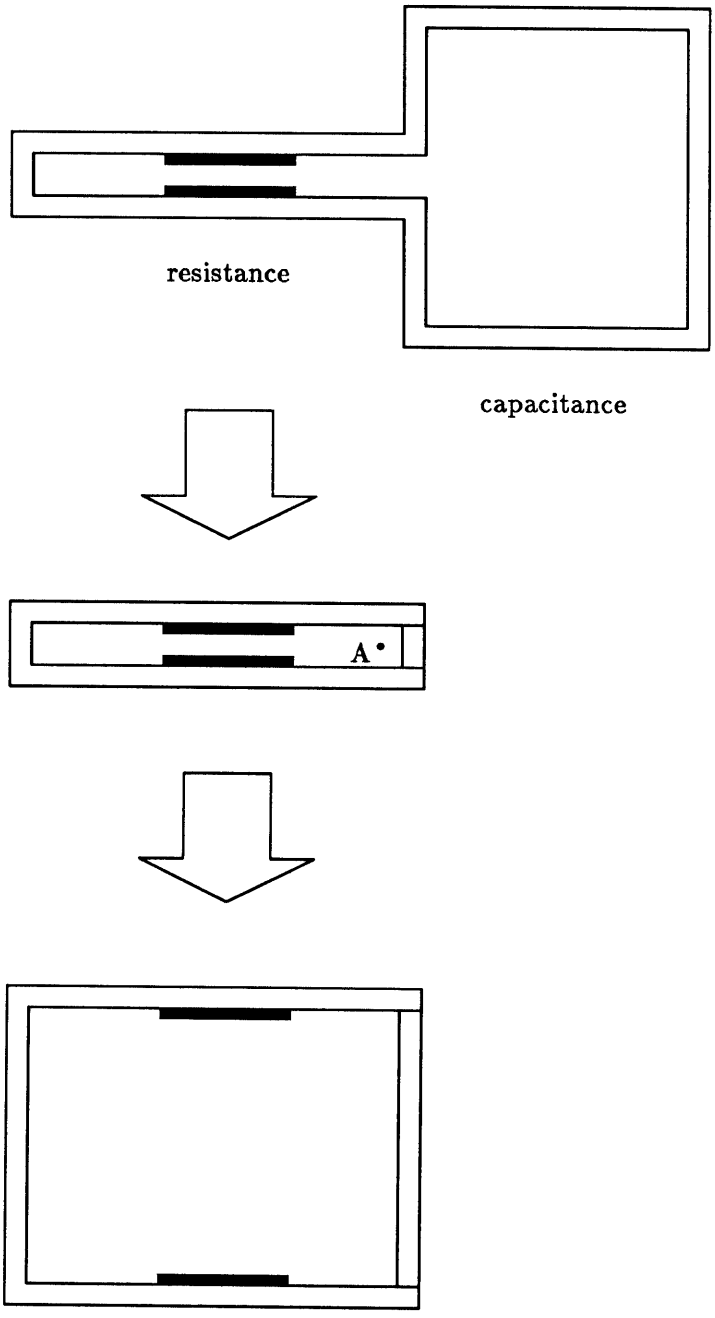

Figure 4.14: Modification invalidates primary function of element.

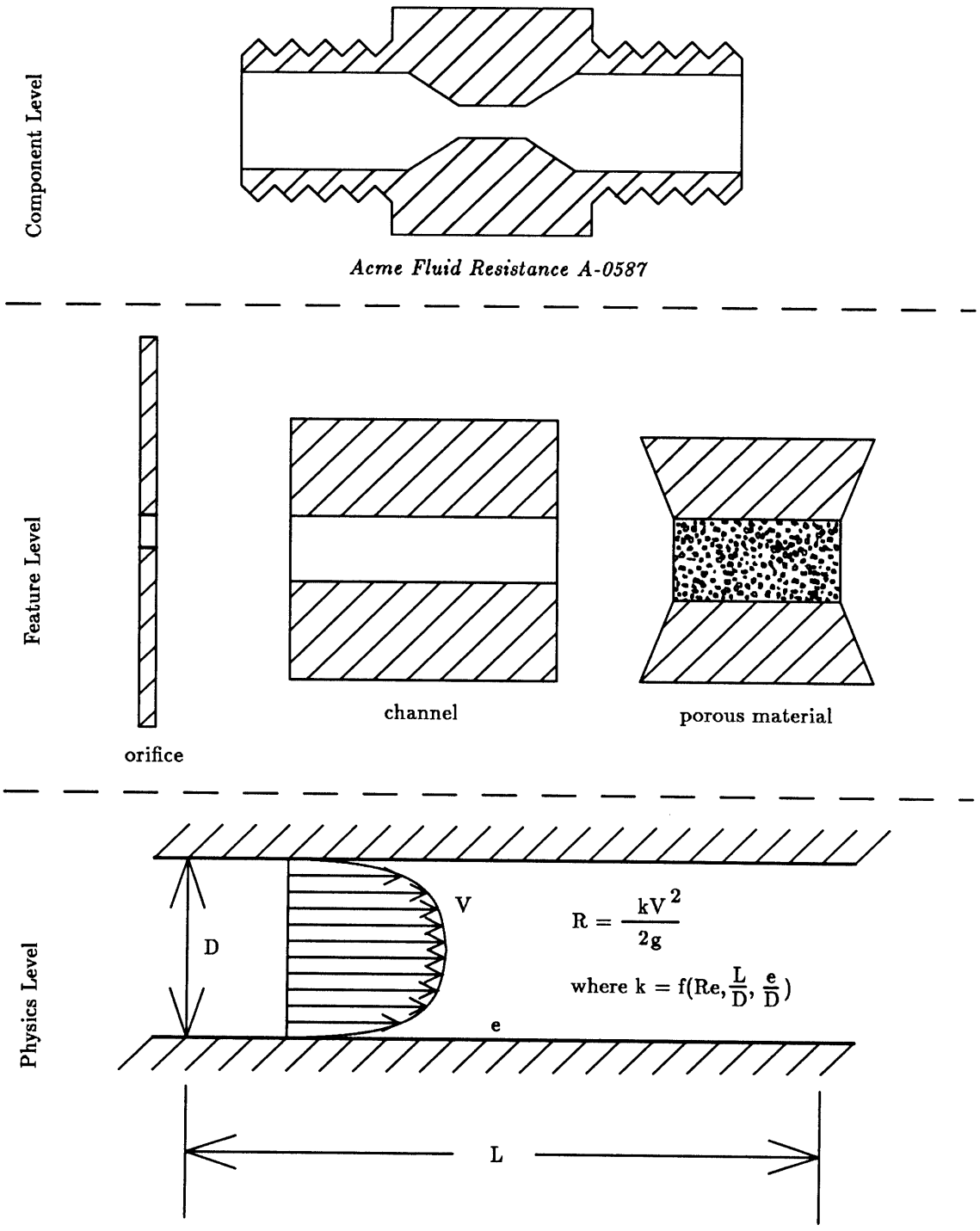

Figure 4.15: Knowledge level for fluid resistance.

a resistive configuration. Alternatively, the system could represent resistance as a mathematical concept with a generalized, parameterized model of geometry. I call this the *physics level.* Given this scheme, a system has a great deal of flexibility in synthesizing resistive configurations.

However, with the flexibility and expressiveness come computational complexity. The reasoning required to derive a resistive configuration from equations at the physics level is severe; but finding resistance at the component level is trivial. I chose a middle ground. Representing resistance as a set of physical features gives flexibility and expressiveness, without swamping the system with complexity.

4.5.4 Novelty in Mechanical Design

A novel mechanical design can be defined as one that no one (or at least not the design community in question) has generated before. One hypothesis of this work is that the unbiased application of the physical feature based design operations in function sharing would yield some novel designs. One design the computer implementation generated truly surprised me. Figure 4.16 shows one branch of the function sharing procedure on the rate-of-climb example. In this case eliminating the fluid capacitance led to a novel use of the cavity associated with the space between the resistance and the piston. The recognition procedure identified the cavity between the resistance and the cylinder wall as potentially resistive and the modification procedure exploited the situation by punching an access hole into the cavity.

As long as the feature recognition procedures are sound, designs will be generated that are viable candidates. Some of these designs will be unanticipated because the recognition procedures are more thorough and faster than human designers, and because they do not encode any designer intention or functional information about the physical description.

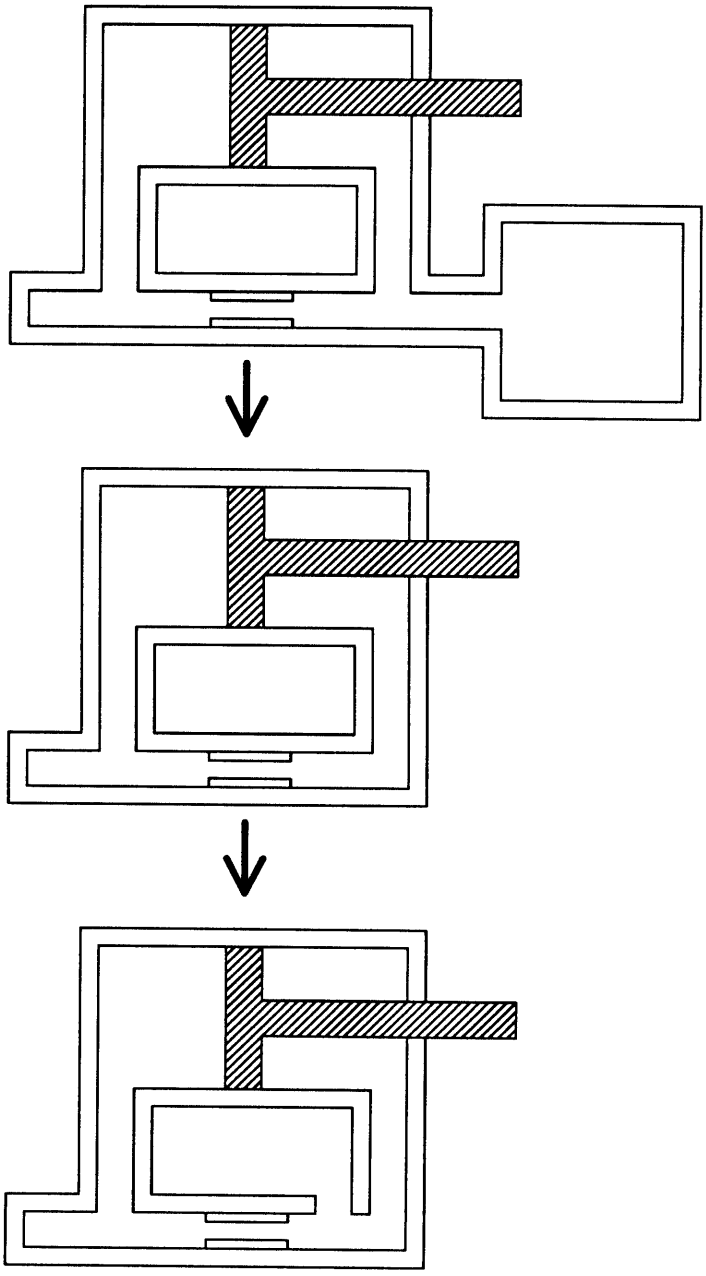

Figure 4.16: A surprising design.

4.5.5 Extensibility

The function sharing ideas presented in this document can be applied to other domains. There are several requirements on these domains, and on the way in which knowledge about the domains is organized. Certain properties of a domain also determine the success or failure of the function sharing procedure.

- There must be a definable language for describing the function of the constituent elements of a design. Designs in the dynamic systems domain can be described schematically with a concise functional language. Global device behavior can be specified by a differential equation relating an input quantity to an output quantity. More important, a network of lumped-parameter elements (fluid resistances, fluid capacitances, etc.) can accurately describe the functional architecture of the device. The existence of a formal and concise language for the schematic description of the device makes the function sharing procedure possible: the function of any particular region of the physical description can be precisely and completely specified, and there is a framework upon which the function sharing knowledge base can be built.

- There must be a definable mapping from function to structure. In the dynamic systems domain, each functional element corresponds to a set of physical features with which it can be implemented. Fluid resistance for example is a functional element that can be implemented with a hole in a plate, a gap between adjacent edges, or a channel cut through a solid. This mapping is accurate and concise. The corresponding mapping for *aesthetically pleasing* or even *aerodynamic* would be much more difficult to define. This is as much a requirement on the domain, as a requirement on the granularity of the functional language. In general the more fine-grained the functional language, the simpler the task

of generating a function-feature mapping. An additional requirement in this vein is that the physical feature recognition problem must be computationally tractable.

- There must be locality of function in the physical design description. If the physical features associated with a particular functional element are highly distributed, then these features can not be easily deleted from the design. In the dynamic systems domain this property is present, as indicated by the existence of definable reference points for a particular functional element.

Some domains that appear to meet these criteria are architectural floor plans, mechanical support and constraint (ie. brackets, bearings, and struts), chemical processes, and assembly machines.

4.5.6 Miscellaneous Issues

This section is a catch-all for isolated comments and responses to commonly-asked questions about the function sharing procedure.

Why start with a modular design?

My presentation of the function sharing procedure assumes that the initial design is highly modular and that it is subsequently processed to become highly integrated. This approach is motivated by two factors:

1. Modular designs are easier to generate, understand, model, debug and analyze than integrated designs. For these reasons, I believe that future design systems will generate modular designs. An important problem will then be how to make these designs more efficient. Function sharing is one answer to this problem.

In fact, Chapter 3 of this document describes a procedure for generating a schematic description of a design, which is one step from being a modular physical description.

2. My observation of actual engineering design practice is that prototypes are modular. This is partially because modular designs have all of the properties stated above, but also because modular designs are easy to fabricate and modify with standard shop equipment. The design prototype is a natural starting point for the function sharing procedure.

Does order matter?

The design shown in figure 4.16 would not have been generated if the resistance had been eliminated before the capacitance. In general the order in which structural elements are eliminated from a design will determine different design states during the function sharing process.

Relation to general function-from-structure problem

The function sharing procedure has as a subproblem a special case of the function-from-structure problem. This problem, determining the function of a device from its physical description, is difficult and underconstrained. In the function sharing case, instead of determining what a certain region of the physical device description *does*, the program only has to determine what it *could* do.

Discussion

Chapter 5

5.1 CONTRIBUTIONS

This project has yielded at least five important results:

- Mechanical design concepts can be generated computationally.
- The separation of schematic and physical synthesis reduces problem solving complexity and clarifies design reasoning.
- The concept of type number and isolated capacitance and inertance can be used to synthesize a schematic description of a SISO dynamic system.
- Function sharing can be explained and automated.
- Novel design concepts can result from the unbiased application of design operators.

5.1.1 Mechanical Design Concepts Can Be Generated Computationally

Some work has been done on generating descriptions of structural shapes [Cagan88, Murthy87] and on generating electrical circuits [Roylance83, Ressler84, Williams88]. To my knowledge, no other research results have shown how it is possible for a computational system to generate a physical description of a multi-part mechanical design from a description of its behavior.

This project has resulted in techniques for generating an efficient physical description of a device from a specification of its input-output behavior. These techniques involve reasoning at the functional-schematic level as well as at the structural-physical level. The techniques represent at least an upper bound on the kinds of reasoning and representations necessary for this synthesis process.

The results of this project are both humbling and encouraging. They are humbling in that the domain is simple and the representations impoverished. Extending the results to the kinds of problems that engineers routinely solve remains a difficult step. The results are encouraging in that some of the mystery has been taken away from design. If appropriate representations for schematic and physical descriptions can be developed, synthesis procedures will follow; and the resulting systems will probably generate more complete sets of design alternatives than is current human practice.

5.1.2 Separation of Schematic and Physical Descriptions

One of the central theses of this project is that design problems should be explicitly separated into schematic synthesis problems and physical synthesis problems. The reasons for this separation are a reduction in problem solving complexity and an

enforcement of clarity of the synthesis issues.

My estimate of the size of the design space (in the SISO domain) at the physical description level is about 10^{17}. This number is derived from considering possible configurations of 5 or fewer structural elements from a library of 20 possible elements, and then considering the number of ways in which this collection could be modified by 3 of the roughly 20 available modification operators in the function sharing procedure. These parameters are chosen to reflect the typical values from the function sharing procedure operating on a typical modular physical description. My estimate of the size of the design space at the schematic level is about 10^4. This number is based on the number of possible bondgraph sequences containing fewer than 5 elements. Working in the bondgraph space is potentially a much less difficult problem than working in the unconstrained physical description space. Once a schematic description has been synthesized then the space of possible physical descriptions is constrained to about 10^8 possible designs (since the configuration and type of structural elements is specified by the bondgraph). These rough figures indicate the reduction in complexity that results from working first in the abstract schematic description space before venturing into the physical description space.

The other major argument for separating schematic and physical reasoning is that it focuses the problem solving activity. In the dynamic systems domain, any design whose bondgraph does not exhibit the specified behavior will not meet the design specifications. This suggests that getting the idealized behavior right at the bondgraph level is a good first design step. This choice of strategy in no way limits the range of designs at the physical level, since any design that meets the specs must have the correct schematic description. By explicitly separating the schematic and physical reasoning in solving the design problem, the procedures that solve the two steps can be narrowly focused. For example, the schematic synthesis procedures do

not have to use any geometrical reasoning.

5.1.3 Synthesizing SISO Dynamic Systems

Engineers have typically solved the dynamic systems synthesis problem in heuristic ways. Chapter 3 of the thesis presented a complete technique for synthesizing SISO dynamic systems in the translational, rotational, fluid, and mechanical media. The technique is based on: 1) using the power spine concept to generate a bondgraph from which all valid designs (smaller than a certain size) must be derivable through the addition of elements only; 2) using rewrite rules to reduce a bondgraph to its essential skeleton; 3) and defining the concept of an isolated capacitance and inertance in order to modify the type number of a system. This is not only a complete explication of the domain knowledge for this class of schematic synthesis problems, but is also an instance of a solution to a schematic synthesis problem that meets the desiderata presented in the thesis introduction.

5.1.4 Function Sharing Can Be Explained and Automated

Although the idea of function sharing as a property of designs is not new, to my knowledge this project is the first to propose the idea of function sharing as a design procedure. This research identifies and defines the phenomenon as a mapping from more than one element in the device schematic description to a single element in the device physical description. In addition to elucidating the phenomenon, I have described and implemented one procedure for achieving function sharing in the dynamic systems domain. The result is a program that simplifies designs. It is likely that other specific procedures and techniques for achieving function sharing will be developed, but they will probably be based on the idea of using relations between

functional attributes and structural features to exploit useful secondary properties of regions of the physical design description.

5.1.5 Novel Designs can be Generated by Unbiased Application of Design Operators

This project was initially motivated by an interest in novel conceptual design. One of the important results of this research is that novel design can be partially achieved by the unbiased application of design operators. If the knowledge encoded in the operators is valid then the results will be valid. If the operators act in every case in which they are appropriate, they will perform operations that people have not thought of. The novelty results in part because the design operators are faster than the human equivalent and therefore generate many more concepts than the human designer; and in part because the design operators are systematic— having no bias with respect to how a design feature *should* be used.

The design procedures in this project generated unanticipated or novel design concepts several times. At the schematic synthesis level, the novel designs were simply valid configurations that I had not been able to generate without applying the systematic design procedures. At the function sharing level, the novel designs resulted from the fact that the program can find and recognize structural features quickly and without bias of intended function.

5.2 FUTURE WORK

This section presents some of the lessons that can be derived from this project and applied to future work, and then presents several ideas for projects that follow directly or indirectly from this research.

5.2.1 Lessons for the Future

Although many of the ideas in this document apply only to the dynamic systems domain, the intent of the research was to develop some insights into the problem of computational approaches to pre-parametric design.

CAD tools will require functional reasoning

Commercial computer-aided design (CAD) tools are designed to store geometrical information about a part (as well as occasional fabrication instructions in the form of text strings). As CAD tools become more useful they will perform tasks like: identification of undesirable secondary behavior of a part; derivation of part models; recognition of commonality between several parts; and simplification of complicated parts. Geometrical information is not enough to perform these tasks. In fact, the ability to derive possible behaviors from geometry is not enough. These CAD tools must also represent and manipulate the function of a part that the designer intends. This was a requirement of the function sharing segment of this research— in order to be able to eliminate some structural element, the program had to have access to its intended function.

Geometrical reasoning is essential

I initially believed that computation could be used to generate novel mechanical designs without having to represent the geometry of the design explicitly. In fact, as I discuss in Appendix D, almost all of the interesting properties of mechanical devices are derived from the device geometry. The schematic description is often the easy part of mechanical design, and the potentially innovative and interesting aspects of a design are derived through geometrical reasoning. This observation suggests a

need for better techniques for manipulating, modifying and recognizing geometrical features in a design.

Formal schematic languages are needed

Progress in the dynamic systems domain was possible because of the existence of a formal language for representing device schematic descriptions. The bondgraph language captures most of the functional information important in designing a dynamic system. Because the language is formal, a program can decide if a description is well-formed, and the bondgraph modification operators are well-defined. An additional property of the bondgraphs is that the equations of motion of the system can be derived from the schematic description. These properties of the language make the schematic synthesis problem tractable. Languages with similar properties will be necessary in other domains in order for schematic synthesis to be performed.

5.2.2 Future Projects

Several projects are direct extensions of the work that I have described, and several projects have come up as ideas that are peripheral to the ideas in this document.

Other schematic languages

As I wrote in the lessons section, formal schematic languages are essential to performing pre-parametric design computationally. These languages exist is several domains: kinematics, analog circuits, digital circuits, chemical process plant design, hydraulic circuit design, and dynamic systems. An important project is to identify other domains in which these languages can be developed. In most mechanical design domains, the schematic language is implicit in the designers reasoning. This may

be because the schematic synthesis problems are simple compared to the physical description synthesis problems (as in the case of designing a bracket), or it may be because no one has invested the effort in developing such a language. This future research effort should be problem driven— that is the schematic synthesis task should be one that would either benefit from automation because it is tedious or because it is very difficult. Some possible domains are: bowl feeder design, shaft and bearing system design, and the design of zero-degree-of-freedom parts like brackets and supports.

Automatic device modeling

The function sharing procedure in some ways is aimed at deriving useful secondary properties of devices. This reasoning is accomplished through the use of an explicit representation of the relationship between certain physical features of a design and certain functions. These same relationships could be used to construct a primary functional model of a design from the physical model. Once a functional model is built it could be used to predict the performance of the device. The performance results could then be examined for unanticipated and possibly detrimental attributes. Engineers spend a great deal of time constructing functional models of devices in order to gain insights into the relationships between the design parameters and the device behavior. A useful tool would be one that could derive a specified type of model from a physical design description.

Manufacturing compiler

One of the results of the function sharing procedure is the simplification of the design through a reduction in the number of structural elements. The function sharing ideas along with several others could be used to develop a sort of manufacturing

compiler. The idea would be to describe a prototype design physically to the system and to generate a highly simplified version of the design that would be ready for inexpensive manufacturing. In addition to the function sharing ideas, this kind of procedure would require the ability to reason about the costs of various kinds of design alternatives.

Other classes of design reasoning

Appendix D discusses some other kinds of design reasoning that lead to innovative design concepts. Finding shape-optimal structures, novel combination of existing device features, and exploiting unusual physical laws and effects are all potentially useful classes of design reasoning that may be suited to computational approaches.

Literature Review

Appendix A

The research described in this thesis is one of the first efforts to perform pre-parametric mechanical design computationally. Because of the infant state of this field, there are few papers from which my research has followed directly. Many papers have however been instrumental in shaping my thinking about research in computation and design, and indeed about the activity of design itself. I divide these papers into two categories, and have accordingly divided this chapter into two sections. In the first section, I present one paragraph summaries of the papers that deal directly with issues that are addressed by this thesis. In the second section, I present one or two sentence summaries of papers that have been indirectly influential on my results. In each section, the papers are ordered alphabetically. The primary purpose of this review is to allow future researchers ready access to a large number of relevant research papers that are widely scattered in the literature.

The actual references for the cited literature are at the very end of this document mixed with other references that have been used in the main text.

A.1 DIRECTLY RELEVANT PAPERS

[Doyle88] describes work aimed at generating explanations of device behavior. In particular the input to the explanation problem is a time history of events or device states. The output is a network of causal connections between the events in the observation. Each connection corresponds to a physical phenomenon or mechanism. This causal network can be thought of as a sort of schematic description of the device. Doyle works with a very broad class of devices, including toasters, bicycle coaster brakes and tire gages. To deal with this wide variety of devices, he has developed a very rich representation for physical phenomena, including many kinds of transduction and transport. The key contribution of this work is the exploration of the power of various types of constraints in controlling the combinatorial complexity of the explanation problem.

[Freeman71] is important primarily because it is one of the first papers to discuss using connections between functional and structural descriptions as a knowledge base for performing design operations computationally.

[Hirschtick85] is a thesis dealing with the problem of providing design advice with respect to a cross-section of an aluminum extrusion. The basic approach is to use a hierarchical structure of features (for example lines are bottom level features and knife-edges are top level features) in order to recognize important design attributes in the two-dimensional cross-section. The recognized features serve as the input to a rule-based system that encodes design knowledge from an aluminum extrusion handbook.

[Ishida84] deals with the problem of deriving function from structure. The general function from structure problem is underconstrained, but this work focuses on a limited class of problematic functions (device leakage for example). The objective of

the research is to develop a statement of the problematic function or unanticipated behavior in terms of a predicate operating on the geometrical representation of the problem. So device leakage, for example, is described as a property of the geometry of the device, expressible as a condition on the connectivity of a graph of part faces.

[Jakiela87] integrates computational feature recognition with rules associated with design for assembly. In particular Jakiela has encoded the Boothroyd-Dewhurst design-for-assembly guidelines as procedures operating on a solid model description of a part. The system is interactive and suggests to the designer modifications that will make the designed part easier to feed. If the designer accepts the suggestions, the modifications are made automatically to the solid model of the part.

[Murthy87] is an extension of Penberthy's work on the graph of models [Penberthy87]. The program described in Murthy's paper is based on the idea of attempting to find a solution to a design problem in the parameter space associated with a given design configuration. If a solution can not be found using several different design models in the analysis, then modification operators change the design configuration. This procedure was applied to the problem of designing beams for torsion and bending. In one case, the system generated a description of a tube by modifying a solid bar.

[Prabhu88] derives some mathematical properties of multi-input, multi-output devices characterized by power flows between inputs and outputs. Prabhu uses a bondgraph representation for these devices, and concentrates on synthesizing the "junction-structure" or network of transducing elements that can link the inputs with the outputs while dividing the power flows according to specification. Because the bondgraph is a formal mathematical representation, certain results such as soundness and completeness are proven.

[Ressler84] is a thesis describing a computational procedure for designing op-

erational amplifiers. The procedure is based upon a hierarchical circuit grammar. Amplifiers are viewed as consisting of three stages. Stage one may be implemented as a differential pair, a current cancellation configuration or a super beta circuit. A differential pair may be viewed as a load and emitter coupler pair. A load can be resistive, a current mirror , a simple pair, or a Darlington pair. There are similar expansions for the other amplifier stages. The design procedure is to implement the amplifier with the simplest possible pieces, then analyze the resulting circuit at each stage in the instantiation process. If the circuit will not meet the specifications, then a more complex option is chosen.

[Rieger77] is one of the first papers attempting to describe the causal structure of a mechanical device. This paper presents a language for describing devices, and develops a model in this language for a thermostat. The functional elements of the model are terms like: continuous and one-shot enablement, state coupling, state equivalence, state antagonism, threshold, and rate confluence. This language was designed to allow descriptions of complex, highly non-linear device behavior, that can not be described with differential equations. Rieger uses these descriptions to simulate the behavior of the devices by assigning a computational behavior to each functional element in the device description.

[Rinderle87] discusses the relationship between form and function in design. He points out that in mechanical design there is often no clear hierarchical decomposition from function to form, as there is in VLSI design. He suggests that there are some fundamental relationships between form and function in mechanical design and that these relationships can be described. Rinderle postulates that the codification of some of these relationships could help novice designers make preliminary design decisions.

[Roylance83] is probably the first effort at synthesizing analog circuit configura-

tions computationally. In this master's thesis, Roylance describes a design-rule-based system that designs simple RLC circuits. The design rules allow the system to backward chain from an equation specifying the desired circuit behavior.

[Suh78] introduces the idea of design axioms. The paper presents the following hypothesis: "There exists a small set of global principles, or axioms, which can be applied to decisions made throughout the synthesis of a manufacturing system. These axioms constitute guidelines or decision rules which lead to 'correct' decisions, i.e. those which maximize the productivity of the total manufacturing system, in all cases." The work described in this paper attempts to generate such a set of axioms. Two example axioms are: 1. Minimize the number of functional requirements and constraints, 2. Satisfy the primary functional requirement first. Satisfy the others in order of importance.

[Sussman80] is an article containing several interesting ideas. First, he introduces the idea of constraint nets as a computational structure. Rather than express mathematical relations as a unilateral procedure for computing certain values in terms of others, constraints express only the relationship between quantities. The constraint net can be used to compute a selected quantity from the remaining quantities. The other major idea in this paper is that of almost-hierarchical descriptions. This term is used to denote the fact that engineered devices have roughly hierarchical relations between functional descriptions and their structural implementations, but that this relation is sometimes muddled by function sharing in the design. Sussman introduces a concept called slices to deal with these almost-hierarchical descriptions.

[Williams88] is one of the very few pieces of work that deals with the synthesis of a design configuration. Williams explores the domain of digital circuit design at the FET level. His program is based upon using qualitative analysis of the behavior of a preliminary circuit design to guide design modifications. Williams develops a

time-based dependency scheme for tracking the influence of one event in time on another event.

A.2 INDIRECTLY RELEVANT PAPERS

[Antonsson87] is a position paper describing some desiderata for research in engineering design research. Antonsson proposes that research in engineering design has suffered in most individual cases because a hypothesis was not articulated and tested. He gives several examples of research problems that follow the hypothesis generation and testing paradigm.

[Cagan87] is a paper describing a system to derive novel structural modifications to a design based on equations describing the design. The key insight is that mathematical integrals are indications that the structure of the design can be usefully decomposed into subregions. For example, since strength of a bar is expressed in terms of an integral along the bar radius, it may make sense to consider the bar as consisting of concentric annuli. Within this formulation, an optimization of the design may specify zero density for the inner annuli, thus generating a tube.

[Davis84] is a paper about diagnosing problems in digital circuits. The really interesting idea in this paper is that potential problems may only be obvious in a particular representation of the device. For example, finding a problem due to a solder glob is difficult using the schematic description of the circuit, but relatively easy using the wiring diagram of the circuit.

[de Kleer85] is relevant to my work primarily because it introduces the no-function-in-structure principle. This principle states that systems that are to derive behavior or function from a structural description of a device must represent the device with a description that is devoid of any functional information.

[Frieling77] is an early attempt at creating a representation language for describing the function of mechanical devices.

[Gero87] is an survey article describing some of the major prior work in knowledge-based systems for design. Gero proposes that future work be done in three areas: design analysis/formulation, design synthesis, and preliminary design evaluation.

[Glegg73] and [Glegg69] are books based on the introspection of Gordon Glegg, a British designer and design educator. He presents a set of heuristics gleaned from his own experience.

[Jansson87] discusses teaching engineering synthesis and analysis. Much of my work was inspired by the scenario documented in this paper of the design by Y.T. Li of a novel tiltmeter. The design was based on an abstraction of the problem and clever use of analogy.

[Maher87] proposes that certain design synthesis tasks can be thought of as a search problem within a design space derived from an or-tree representing a strict hierarchical expansion of high level goals. Maher implemented a system that imposes constraints on the combinations of design options that are allowed. These constraints help to prune the design space.

[Penberthy87] presents a concept called the graph of models. The idea is to construct a graph in which nodes are models of a particular domain, and links correspond to the invocation or relaxation of specified assumptions. This graph allows a design analysis system to move between different levels of model sophistication.

[Rinderle87a] discusses the automatic generation of critical design relations. Such relations might be ratios of design variables that compress the number of equations required to describe a design.

[Rodenberger83] is useful in justifying computer tools for design. The argument is that design engineers at General Dynamics, Inc. make design decisions costing

about $100,000,000 for each project they work on. So, even small improvements in designer productivity can have large payoffs.

[Serrano87] presents techniques for computing design values from specified parameters and a constraint network. These techniques are useful for a sort of spreadsheet-like design tool.

[Simon81] contains a chapter called The Science of Design, in which he proposes that design should become a science. This chapter introduces the concept of satisficing versus optimal solutions.

[Kannapan87] develops an instance of a schematic language for mechanical design. The language includes primitives shafts, bearings, handles, supports, etc. Pieces of devices described with this language are combined together to form new designs.

[Ward88] describes a mechanical design *compiler* for selecting specific machine elements from catalogs from a description of the topology of a power train. The technique for performing this compilation requires the development of a special interval mathematics for representing the properties of components and specifications.

[Winston83] describes a set of ideas for constructing definitions of objects in terms of relations between their functional and structural attributes. These definitions allow a system to recognize a new artifact based on its structural description and a definition extracted from a set of precedents. A novel object could be recognized as a cup by inferring that because it has a handle it is liftable, and because it has a flat bottom it rests stably, and because it has an upward-pointing concavity it can contain liquid. These ideas can be inverted in order to design new artifacts based on existing precedents.

Invention as Novel Combination of Existing Device Features

Appendix B

NOTE:

This appendix chapter is a version of:

Ulrich, K.T. and W.P. Seering, "Computation and Conceptual Design," *Robotics & Computer-Integrated Manufacturing*, Vol. 4, No. 3/4, pp. 309-315, 1988.

B.1 Summary

Design is the transformation between a functional and a structural description of a device. Conceptual design is the initial stage of this transformation. We hypothesize that most new designs are derived from knowledge of existing designs. We identify a special case of this process and call it *novel combination.* By describing a fully implemented program which designs novel mechanical fasteners, we explain how knowledge of existing devices can be represented and used. We highlight the issues arising from this implementation and propose four areas of future research. This work

is important for establishing a fundamental understanding of conceptual design.

B.2 INTRODUCTION

Computation has influenced design in drafting, databasing, and analysis, but there has been a dearth of computational attention to conceptual design. This is because of the perceived difficulty in understanding how new ideas can be generated. Conceptual design is a segment of the design process which receives perhaps the least attention in terms of resource allocation and yet involves decisions which have perhaps the greatest influence on eventual project success. The objective of this work is to use a computational approach to obtain a better understanding of engineering idea generation.

B.2.1 Function, Structure and Conceptual Design

The *function* of a device is its purpose or intended use. A functional description consists of a set of functional attributes which might be characterized by a truth table, a performance curve or a set of verbal phrases. The *structure* of a device is its physical form. A structural description consists of a set of structural attributes, which might be communicated through a drawing, a model, or a physical implementation of the device itself. While structure and function are related, they do not uniquely determine each other– a single structural attribute can contribute to more than one functional attribute, and a single functional attribute can be implemented in several ways by different combinations of structural attributes. Design is the transformation between a functional and a structural description of a device. The structure of a device can be specified at many different levels of detail ranging from a sketch on a napkin to a scale model. We are primarily interested in conceptual design, the

first stage of the design transformation, which ends with a structural description at a level of detail more typical of the napkin sketch than the scale model.

B.2.2 Scope, Objectives and Approach

Although most of the ideas we present transcend the boundaries of traditional engineering disciplines, our research focuses on the mechanical design domain. Our primary objective is to better understand how new ideas for mechanical devices can be generated. This goal is motivated by the long-term desire to establish a rigorous theoretical foundation for enhanced design teaching, to develop better design tools, and to eventually design autonomous inventive machines. Our research approach has been to develop well-defined procedures which operate on a well-defined representation of a particular domain of objects to solve specific idea generation tasks. We then use computation to test these procedures and representations. This approach is very useful for eliminating fuzzy conceptual thinking. Notice that while our intuition as designers is useful, our central aim is to develop effective design procedures and not to understand the particular procedures human designers use when performing conceptual design tasks.

B.3 CENTRAL CONCEPTS

The key hypothesis of our work is that most new designs come from knowledge of old designs. The novelty of a design concept is a reflection of the intellectual distance (or the complexity of the information processing) between what is already known and what is proposed as a new solution to a problem. As a starting point of our research we have analyzed a limited but important class of conceptual design we call *novel combination.* Novel combination is the process of extracting individual structural

attributes from each of several known devices and combining these attributes in novel ways to create structural descriptions of new devices. This form of invention is ubiquitous. It pervades human activities from cooking to computer design. As an example, the threading tap could have been invented by combining the gradually-increasing-cutter-size feature of a keyway broach with the helical-path feature of a screw. In order to understand how this sort of conceptual design can be performed, we explain how knowledge of existing devices can be represented and how this knowledge can be used procedurally to fulfill a functional specification of a design.

B.3.1 Knowledge Representation

Since we have defined design as the transformation between a functional and a structural description of a device, a system which performs design must have knowledge which includes functional and structural descriptions and the relationships between function and structure. Our representation for a device is a network of functional and structural attributes linked by causal relations. For example, figure B.3.1 shows the knowledge representation of a shaft coupling. Some of the functional attributes of this device are *transmits torque, compliant in tension, compliant in bending, strong in torsion* and *corrosion resistant.* Some of the structural attributes are *bellows construction, strong material, elastic material, polymeric material,* and *polycarbonate material.* These attributes are linked causally as shown in the figure. For example, the device is compliant in tension because it is a bellows construction and because it is made of an elastic material. The material is elastic because it is polycarbonate. These relations can be viewed as propositions of the form *structure 1* $\wedge$ *structure 2* $\rightarrow$ *function A.*

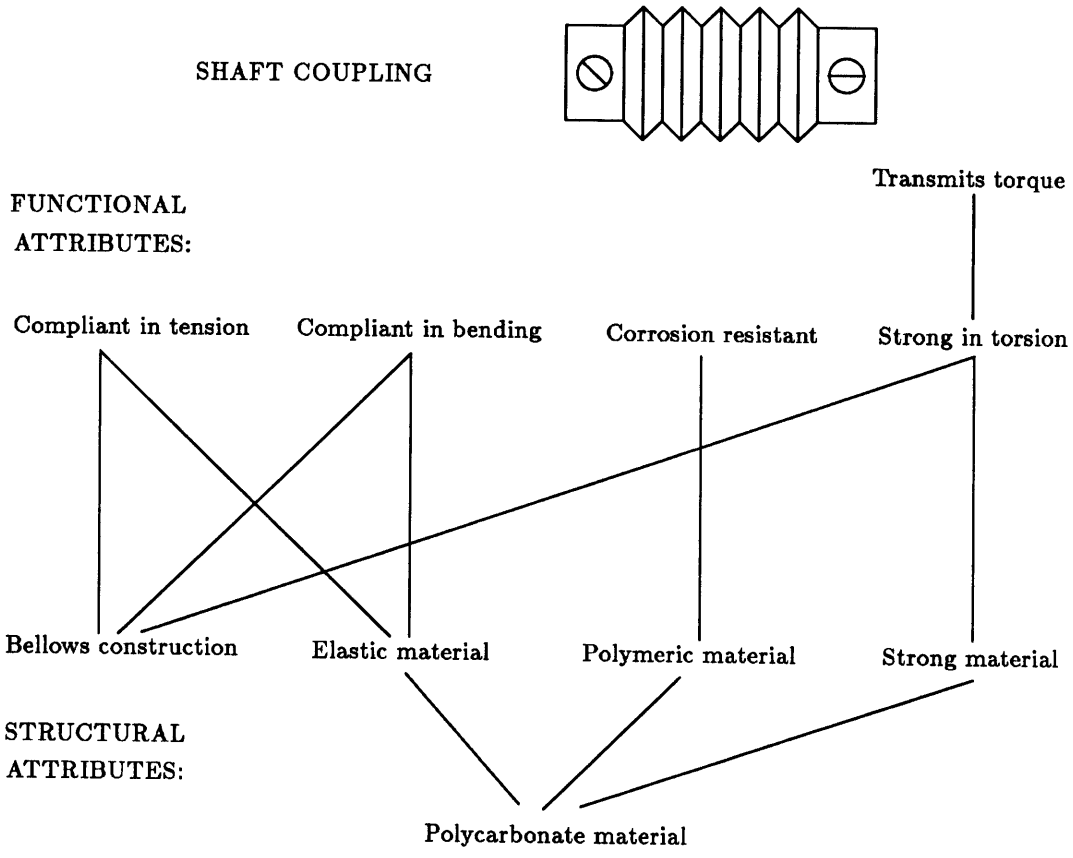

Figure B.1: Shaft coupling knowledge.

B.3.2 Design Procedure

Imagine that a design system has a representation of many devices as described above. Now this knowledge can be used to synthesize new devices. As described in the introduction, the system must transform a functional specification to a structural description. The input functional specification can be thought of as a set of requirements, each one of which must be fulfilled for the design to be functional. For example, an input to a design system might be the requirement that a design have both function A and function B. Now, if the system knowledge base contains an existing device in which function A is caused by structures 1 and 2, then the system can substitute structures 1 and 2 for function A in the new device specification, such that the requirement for the device is now that it have structure 1, structure 2 and function B. This new set of requirements can be thought of as a new design specification. The system can then find structural causes for function B in its knowledge base. And, the system might find more detailed structures which cause structures 1 and 2. This procedure continues until no further refinement can be performed. Notice that in general the system knowledge base will contain more than one device with a particular function, and therefore in general more than one structural cause for a particular function. When the system refines the input specification under these conditions, it creates a new specification for each possible structural cause. This procedure results in many possible structural specifications for a particular design description. Because many of the refined descriptions are not physically realizable designs, there must be a filtering mechanism in the system which removes or resolves conflicting designs. If this filtering is accomplished between each step of the refinement process, the combinatorial explosion of potential designs can be harnessed.

B.4 AN EXAMPLE

We have implemented a program which does novel combination in the domain of mechanical fasteners. There are hundreds of thousands of different kinds of mechanical fasteners ranging from velcro to railroad spikes. Since novel fasteners are characteristically formed from structures extracted from known fastener designs, the fastener domain is especially appropriate for implementing novel combination. This section describes the fastener example by outlining the task, the knowledge base, the methods, and the output of the program.

B.4.1 The Task

The program task is to propose a fastener design which will meet a particular set of functional requirements. The functional specification is made within the context of the situation described in figure B.2. The fastener must hold plates A and B together, be actuated from the front, and be inserted in the hole. In addition, we can specify that the function of the fastener must not depend on a second device such as a nut. A variety of actuators are assumed to be available, and modifications to the hole are allowed. Since we are interested in conceptual design, issues such as the actual size of the hole or fastener are not of primary importance. An example solution to a typical problem specification might be a machine screw or a cotter-pin.

B.4.2 System Knowledge

The knowledge base of the program consists of descriptions of seven common fasteners: a cotter-pin, a roll-pin, a machine screw, a hex bolt, a socket-head screw, a dowel pin, and a "christmas tree" (barbed) fastener. Figure B.3 is an illustration of these fasteners.

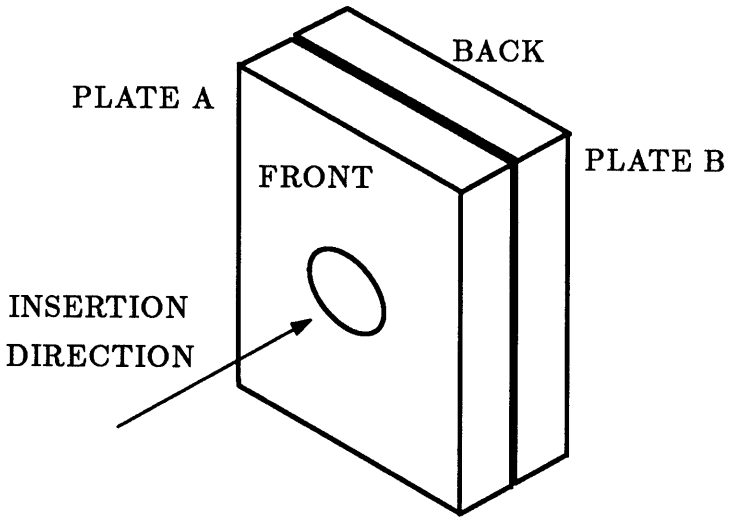

Figure B.2: Fastener task situation.

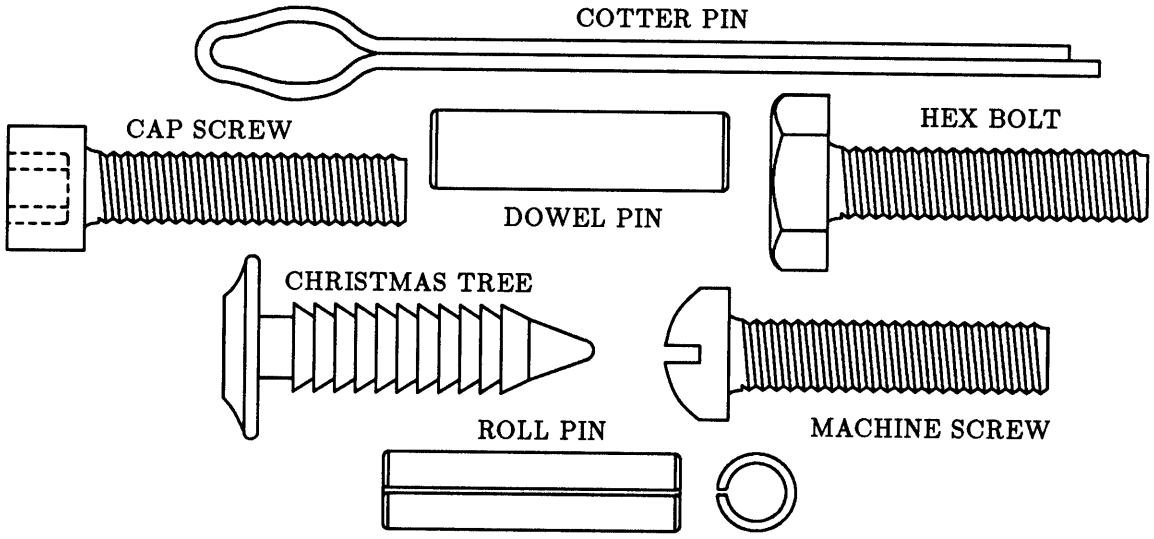

Figure B.3: Fasteners in the knowledge base.

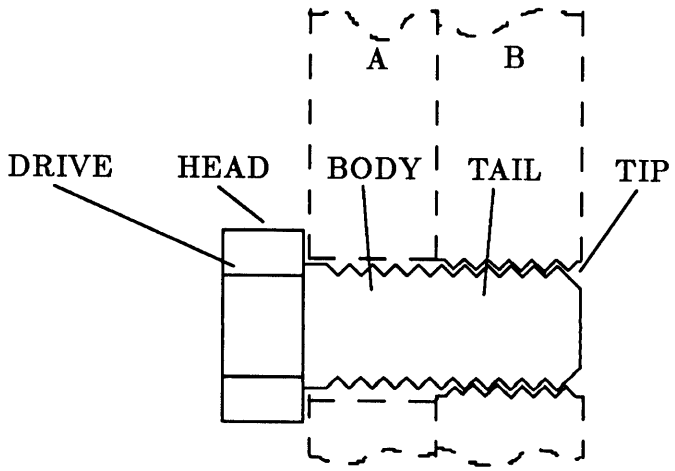

Figure B.4: Structural framework for a fastener.

The structural description of every fastener is specified in terms of a fixed framework. There are five essential structural features for a fastener which will fit the problem situation. The fastener must have a **drive**, which is the structure which forms the interface to the fastener actuator. It must have a **tip**, which is the structure which enters the hole first when the fastener is actuated. It must have a **body**, which is the structure which occupies the hole in plate A. And it must have both a **head** and a **tail**. The head and tail are the structures which provide the tensile reaction forces to plate A and plate B when the plates are pulled apart. In the case of the hex bolt in figure B.4, the drive is the hexagonal shape of the end of the fastener, the head is the larger diameter shoulder section, the body is a section of threads which fit loosely in the hole through plate A, the tail is another section of threads which mate with threads in the hole through plate B (threads are an allowable hole modification), and the tip is the chamfered section at the end of the fastener.

There are eight high-level functional attribute classes which are commonly used when describing the function of a fastener. They are fastening strength, retractabil-

ity, required modifications to the hole, degree to which access to the back of plate B is required, ease with which the fastener can be actuated, whether or not the fastener protrudes from plate A, whether or not it protrudes from plate B, and the precision with which it laterally locates plate A with respect to plate B. In specifying a new fastener or describing an existing fastener, these attribute classes take on discrete values of low, medium and high, or yes and no. Several intermediate functional attribute classes are also introduced. An example of these classes is head strength or tail strength. A complete description of an existing fastener is shown in figure B.5. Note that the names for the functional classes have been abbreviated. Also note that the causal network structure is a network of the type described in the section on knowledge representation. One of these networks exists for each of the seven existing fasteners. As an example of the causal relationships contained in the network, consider the case of the fastener strength of the hex bolt. Strength is determined by head strength and tail strength, which in turn are determined by the structure of the head and the tail.

B.4.3 Methods

Imagine that we want the system to design a fastener which has high strength, is retractable, and is highly precise. Assume we don't care about the other device attributes (notated by *dc*). Then, in the notation of the knowledge representation language, we want *(strength hi) and (retractability yes) and (precision hi) and (back-access dc) and (actuation-ease dc) . . . (tail-out dc).* The program first looks for known fasteners which have high strength. For each known fastener with high strength, the causes of the strength node are retrieved. For each unique set of causes a new specification is created with the set of causes replacing the strength attribute in the initial fastener specification. For example, in using the hex bolt

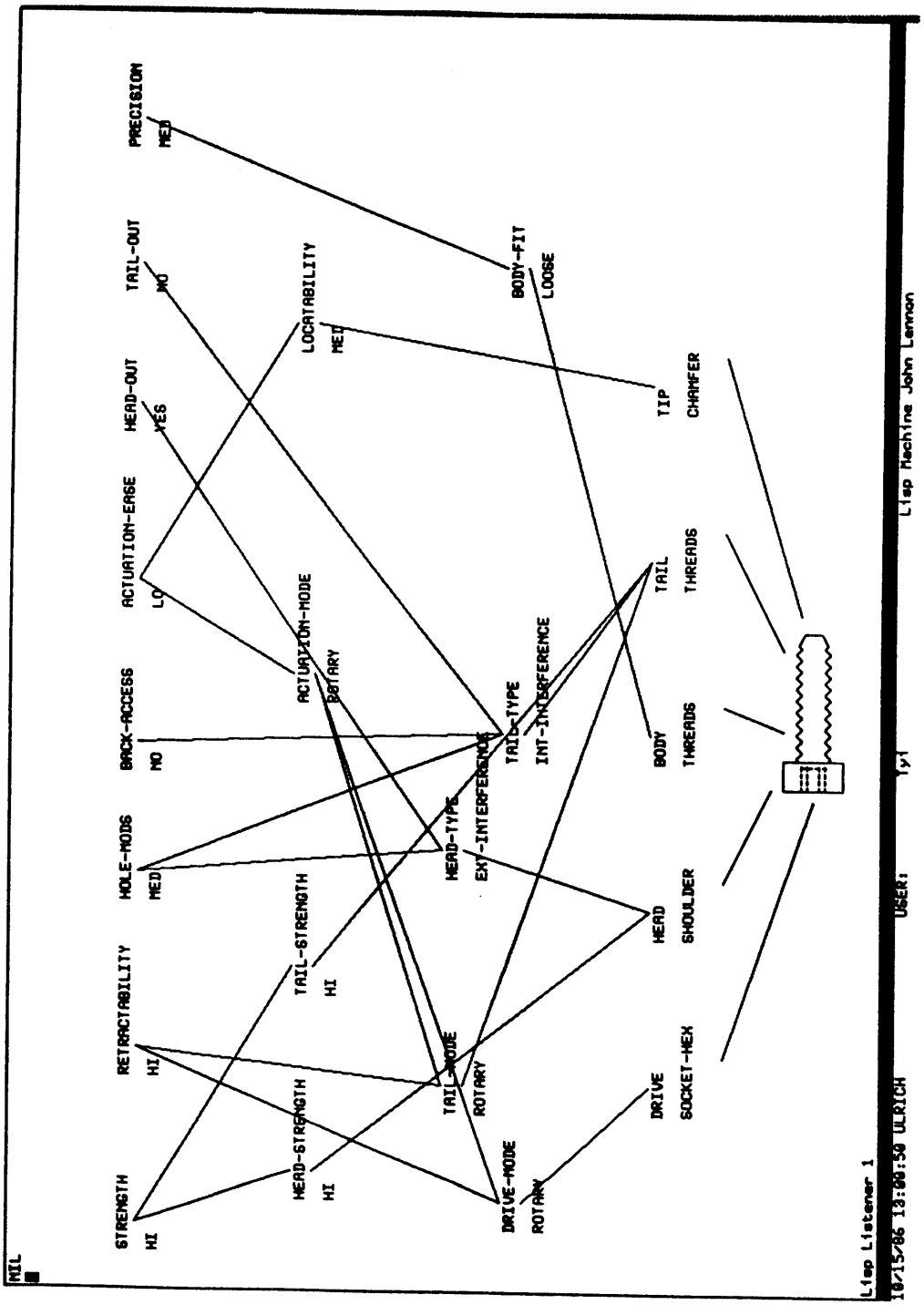

Figure B.5: The knowledge base description of a fastener

knowledge, the system would substitute *(head-strength hi) and (tail-strength hi)* for *(strength hi)*. The program then repeatedly expands these new fastener specifications until the fastener specification is a structural description in terms of the five lowest-level structural attribute classes. Between each step, the program checks to see if a fastener is acceptable. An acceptable fastener is one in which there are no conflicting structural attribute values. The structure is in conflict if there is more than one different value for a single structural attribute (ie. *(drive slot) and (drive external-hex)*).

B.4.4 Output

The result of applying the above design procedure to an input specification is a set of potential designs, each of which consists of a set of five structural attributes corresponding to the five parts of the fastener structure. Each of these sets of structural attributes can be graphically represented as shown in figure B.6. Figure B.6 shows the fasteners which result from running the example described above. Figure B.7 is a selection of fastener designs that were generated from several different input specifications. (The graphical descriptions are generated from the structural description by a collection of ad hoc procedures.)

B.5 DISCUSSION

In this section we discuss some observations of the performance of the fasteners program and point out the direction of our future work.

- The program does design novel fasteners. In figure B.6 notice that there are several novel precision screws. One of these resembles the commercially avail-

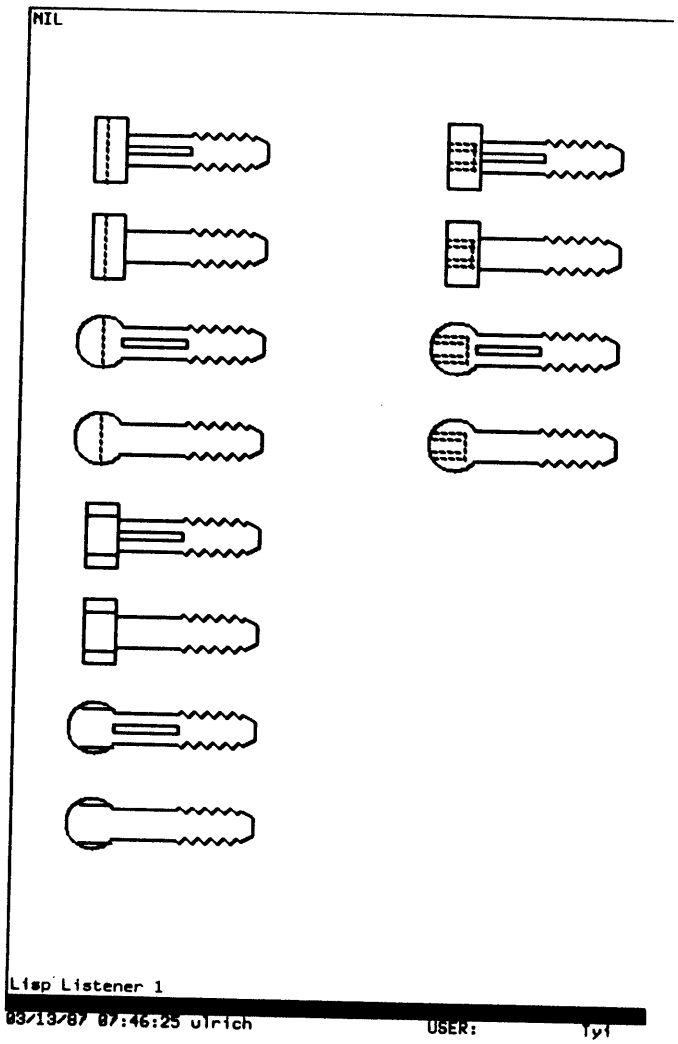

Figure B.6: Example output.

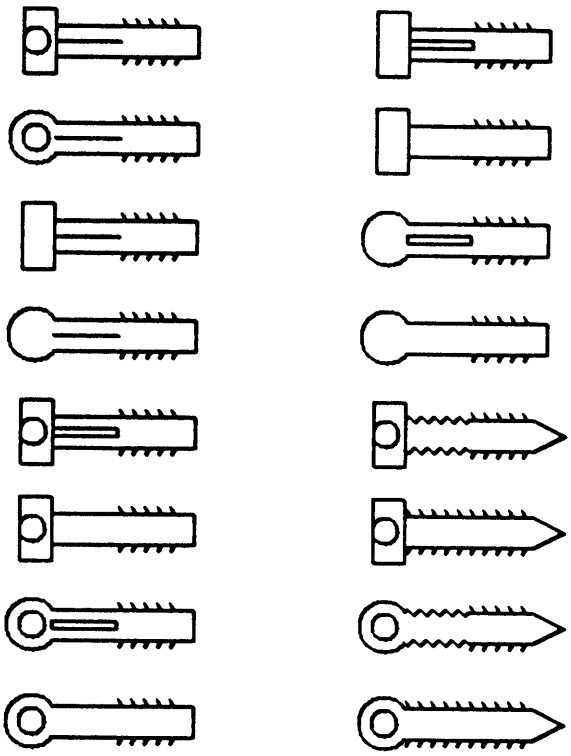

Figure B.7: Other example fasteners.

able shoulder screw. While this screw is common in engineering practice, it is not one of the initial fasteners in the system knowledge base.

- Notice that there is a fixed framework for the form of the solution in the fasteners example. For example, if the system were designing pizzas, the framework would require a crust, sauce, cheese and toppings. The existence of this framework makes the knowledge and solutions easy to describe. If the system were to design sculpture, there would be very little constraint on the framework of the solution. The fasteners domain as we have constrained it is more like pizza than sculpture. Most design problems fall somewhere between these two extremes. The degree to which a rigid framework for the solution to a problem exists is a good measure of the difficulty in applying novel combination.

- The issue of context is pervasive in artificial intelligence research in general and in our research in particular. Notice that the propositions made in our knowledge base are not only made within the context of the task situation in figure B.4, but also imply an infinity of assumptions like standard environmental conditions, moderately elastic materials etc. The use of this knowledge outside of the rigid confines of the fasteners domain would in general result in faulty designs.

- The knowledge base for the fasteners program was devised and coded by hand. This is a tedious process for all but the smallest of knowledge bases.

- The fastener system as currently designed will not increase in performance over time. This is because there is no feedback mechanism such that the system can learn new causal relations with experience.

- Conflicting fastener specifications which arise in the process of refining the input specification are currently eliminated. There are cases where instead of eliminating conflicting descriptions, novel designs could be produced by resolving the conflicts with a novel structure.

Some of these issues point to rich areas for future work. Four of these areas are feedback, conflict resolution, context, and abstraction and analogy.

B.5.1 Feedback

The conceptual design procedure we have described does not improve in performance over time. Improvement is not possible without feedback, or evaluation of the results of a design. Since in general, the knowledge base is only heuristically suggestive of causal design relations, it can be improved or repaired by processing the results of a functional performance evaluation of a design. This performance evaluation could be provided by building and testing each potential design, but this would be expensive in time and resources. Alternatively, a simulator could evaluate design performance. This approach requires that the simulation be at a fundamentally more refined level of detail than the design knowledge (an example of this sort of simulation is finite element analysis). In many cases this simulation could be provided by a human designer, since people have a very powerful ability to envision device performance in many domains.

B.5.2 Conflict Resolution

As noted, the current system eliminates any potential designs with conflicting specifications. Novel designs are often bypassed through this elimination process. Instead of simply eliminating conflicting designs, we would like to resolve the specification

conflicts with new structures. In order to resolve conflicts, a designer must recognize that the function of conflicting structural attributes can often be met by a new structure which results from the elimination of the non-essential characteristics of each of the conflicting structural attributes. In general this ability requires reasoning at a more refined level of representational detail than is expressed in the conflicting design specification. Future research will address the issue of multiple levels of representational detail being applied when appropriate and necessary.

B.5.3 Context

We state that the use of the represented design knowledge outside of the fasteners domain would result in faulty designs. This is because there are many implied conditions on this knowledge. To allow the fastener knowledge to be applied in other domains, it could be grouped together as a class with the implied conditions explicitly represented as attributes of that particular class. Then, when a part of a description is applied in a new domain, this contextual information could be added as an explicit condition on the use of the knowledge.

B.5.4 Abstraction and Analogy

The design procedure we have described uses very superficial analogical reasoning. Much of highly novel conceptual design relies on the ability of the designer to recognize deep similarities between the current design specification and existing designs. This process can be described as abstraction and analogy. Devising representations and procedures for performing abstraction and analogy is a long term goal of our work, which will be essential for highly novel conceptual design.

B.6 RELATED WORK

Our research is built upon the ideas of several researchers. Some of the fundamental ideas on conceptual design were developed with insight provided by Lenat [1,2]. The idea of design reasoning from structure and function was introduced by Freeman and Newell [3] and subsequently used for diagnosis by Davis et al [4,5]. Winston's work on learning device descriptions inspired both the concept of using device descriptions to do design, and the structure of our knowledge representation [6,7]. Simon has written about design learning through simulation [8]. Dyer et al have investigated an alternative approach to invention using qualitative analysis of modified known devices [9].

B.7 REFERENCES

1. Lenat, D., "The Ubiquity of Discovery," *Artificial Intelligence*, **9** (1978), 257-285.

2. Lenat, D., "Why AM and EURISKO Appear to Work," *Artificial Intelligence*, **23** (1984) 269-294.

3. Freeman, P. and Newell, A., "A Model for Functional Reasoning in Design," *Proceedings of the Second IJCAI*, London 1971.

4. Davis, R., et al, "Diagnosis Based on Structure and Function," *Proceedings of AAAI-82,* August 1982.

5. Davis, R., "Diagnostic Reasoning Based on Structure and Behavior," MIT AI Lab Memo 739, June 1984.

6. Winston, P., "Learning New Principles from Precedents and Exercises: The Details," MIT AI Lab Memo 632, November 1981.

7. Winston, P. et al, "Learning Physical Descriptions from Functional Definitions, Examples, and Precedents," MIT AI Lab Memo 679, January 1983.

8. Simon, H., *The Sciences of the Artificial*, Second Edition, The MIT Press, 1981, p. 17-22.

9. Dyer, M., et al, "EDISON: An Engineering Design Invention System Operating Naively," UCLA AI Lab Technical Report UCLA-AI-85-20.

Achieving Multiple Goals in Conceptual Design

Appendix C

NOTE:

This appendix chapter is a version of:

Ulrich, K.T. and W.P. Seering, "Achieving Multiple Goals in Conceptual Design," in Yoshikawa, H. and D. Gossard (editors), *Intelligent CAD*, North-Holland Publishing Co., U.K. 1988.

C.1 ABSTRACT

We are interested in developing ideas that will lead to computational tools for conceptual design. Most conceptual design tasks require the simultaneous achievement of several goals. We propose that the explicit separation of design reasoning into computational modules we call *perspectives* clarifies the operation of the program and makes design knowledge easier to encode, maintain and transport. We illustrate this concept with an explanation of a program that solves a simple design problem.

We analyze this program and discuss the criteria for application of the *perspectives* scheme to other problems.

C.2 INTRODUCTION

This paper describes an organizational structure for computer programs that solve iterative design problems requiring the satisfaction of multiple goals. Our objective is to develop ideas that will lead to computational tools for conceptual design. Our approach is to define an information processing task relating to conceptual design and then to explain how the task can be accomplished computationally.

This introductory section motivates the research issue and explains our general problem solving strategy. Section 2 defines the key concept of design using *perspectives.* We illustrate this concept in section 3 with an implemented program that solves a simple design problem. Section 4 is an analysis of the results of this implementation and a discussion of the criteria for application of our approach to other problems. We conclude by discussing our current research efforts and outlining related work by others.

C.2.1 Conceptual Design Problems Have Multiple Goals

One pervasive attribute of conceptual design problems is the diversity of typical design goals. Consider the following specification for an automobile speedometer.

- Produce an angular deflection of a needle proportional to the angular velocity of an input shaft.
- The input shaft is located at the automobile transmission housing. The indicator needle is located at the dashboard.

- The speedometer should have a low-pass frequency response with a cut-off frequency of 0.5 hz.
- The speedometer should last 200,000 kilometers.
- The speedometer should be easy to assemble.
- The total cost should be less than US$5.00.

Each of these six goals must be met for a design to be successful. Note that evaluating these different goals requires the ability to reason about highly distinct attributes of a design.

C.2.2 Design And Debug Is One Solution Technique

There are several approaches a designer (human or computer) can take to meet design goals like those for the speedometer. If the problem can be mathematically parameterized, then mathematical programming techniques can be used to find acceptable solutions. Unfortunately, most conceptual design problems are not easily parameterized, and most likely the designer will have to use some sort of iterative approach. In one scenario, a designer might first devise some scheme to convert angular velocity to angular deflection and then adjust the configuration and geometrical properties of the design components to achieve the remaining goals. We call this general approach *design and debug.* The process can be thought of as consisting of two phases. First the designer generates a solution that is in some respect close to a final solution. Second, that solution is modified to meet the multiple goals of the problem.

There are many ways to generate a solution that is almost satisfactory. A past solution to a similar problem can be retrieved. A general design template can be

filled in for the class of problem specified. Pieces of several known devices can be combined. A sequence of components with known input-output relations can be assembled to achieve a global input-output relation. In general these techniques will yield a promising but faulty design. The balance of this paper focuses on a way of organizing the modification or debugging of this initial design.

C.3 KEY CONCEPT

Each of the goals in the speedometer problem requires the designer to reason from a different point of view, using a different abstract description of the design. We call a particular abstraction of a design, used to reason from a particular point of view, a *perspective*. In the speedometer example, the perspectives a designer might incorporate include the geometric, dynamic, manufacturing, reliability, and cost abstractions of the design. Each of these perspectives highlights certain properties of the design components and suppresses others. We propose that the explicit separation of these perspectives in computer programs that perform design is an important organizational strategy.

C.3.1 Perspectives Allow Control of Debugging

The separation of perspectives is particularly important for debugging a design. Imagine that a designer only cares about two different design goals and can assess how close a design is to meeting these two goals. The debugging process is represented in figure C.1. The horizontal axis represents the difference between a current design and goal 1. The vertical axis represents the difference for goal 2. Each step in the debugging process is represented by an arrow. There are at least two fundamental approaches to reaching the design goal. The designer can first achieve one goal and

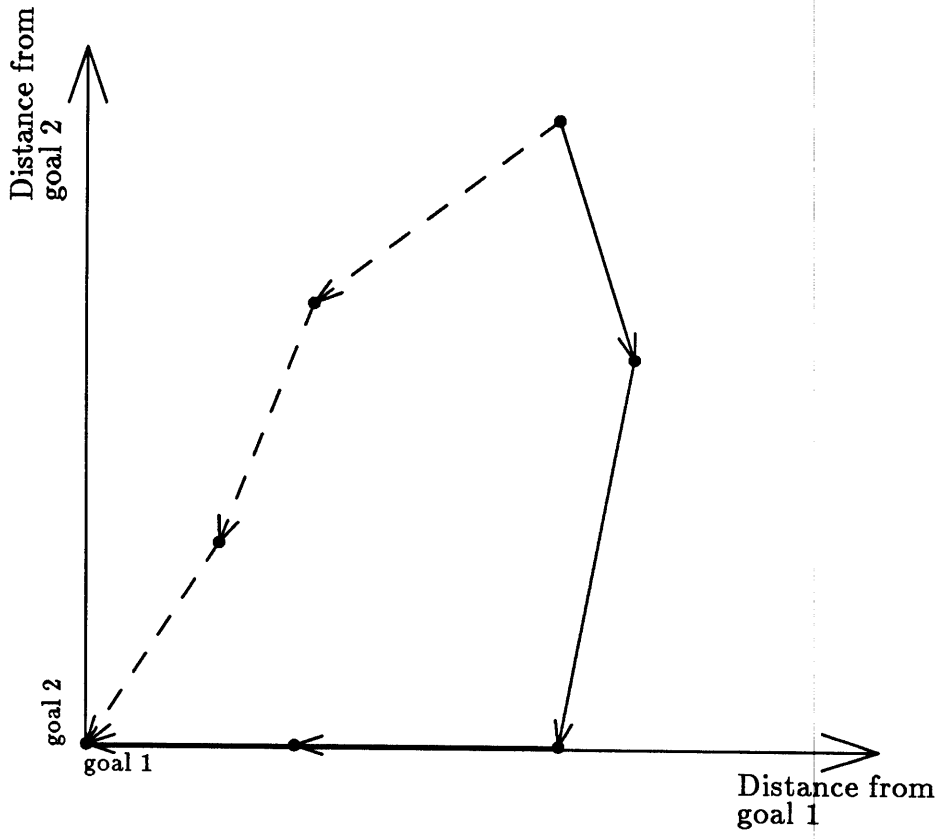

Figure C.1: Debugging to meet two goals

then make changes to the design that preserve the validity of the design with respect to that goal while improving the design with respect to the other goal (solid line). Alternatively, the designer can make changes that improve the design with respect to both goals simultaneously (dashed line). These debugging strategies are made possible by the explicit control of the focus of debugging effort. This control is made possible by the separation of debugging effort into perspectives.

C.4 AN EXAMPLE

We observed that the speedometer design problem is specified by a collection of goals each requiring the ability to view the design from a different perspective. In this section we explain an organizational structure for computer programs that perform this sort of design reasoning. This structure contains computational modules corresponding to each separate perspective required for the solution of the design problem. We illustrate this idea with an example in a simple design domain. As a base case, we consider the problem of meeting *two* design goals simultaneously.

C.4.1 A Domain With Two Distinct Design Perspectives

We have constructed a simple domain to explore the idea of using separate perspectives to solve design problems. The domain includes two distinct perspectives of designs. The design problem is to construct a sequence of blocks chosen from among the blocks shown in figure C.2. The problem can be thought of as an abstraction of the problem of arranging modular material handling equipment that must fit in a certain location and cost a certain amount. The following rules apply.

- Sequences consist of six blocks.
- The *pipe segments* (thick lines within blocks) associated with adjacent blocks must match at the block interface.
- Blocks may be translated but not rotated.
- Any number of blocks from any of the twelve types may be used in a sequence.
- Blocks may not overlap.

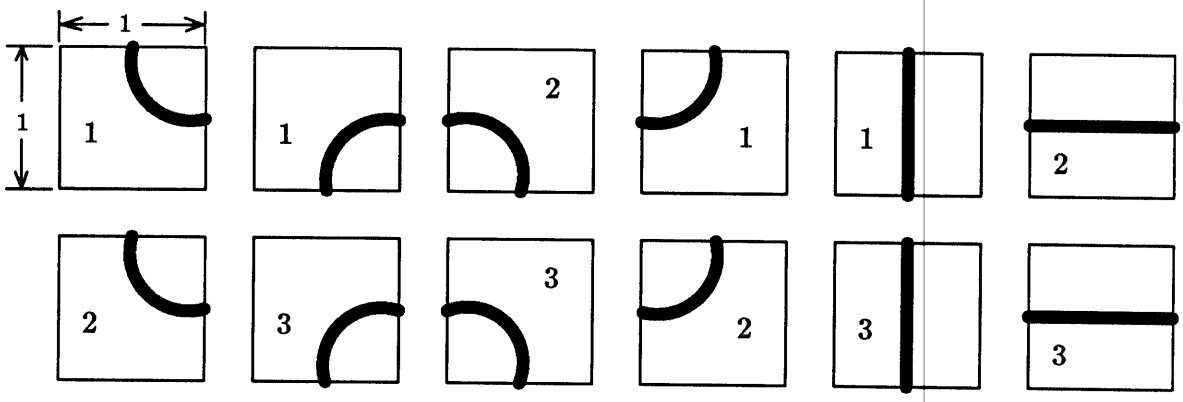

Figure C.2: Components available in the blocks domain.

Figure C.3 is an example of a valid design. Notice that each block has associated with it an integer from 1 to 3 and a *pipe segment.* These attributes are meant to simulate two different design properties of a design component (roughly analogous to geometrical and cost properties of mechanical components). There are two design goals that must be met for a sequence of blocks to be a successful design. First, the **centroid** of the area defined by the block sequence (with respect to the lower left-hand corner of the first block) must be located in a specified region. Second, the integer values of the blocks must **sum** to a specified value. So, in the case of the block sequence in figure C.3, the sum is $1+3+1+2+2+1=10$, and the centroid is located (within .01) at $x = 1.83$ and $y = -.67$.

C.4.2 Using Perspectives in Design and Debug

We solve this design problem by separating the debugging part of the program into two design perspectives–one corresponding to the geometrical properties of the sequence and the other corresponding to the numerical properties of the sequence. Figure C.4 illustrates our program architecture. Each of the rectangular boxes in

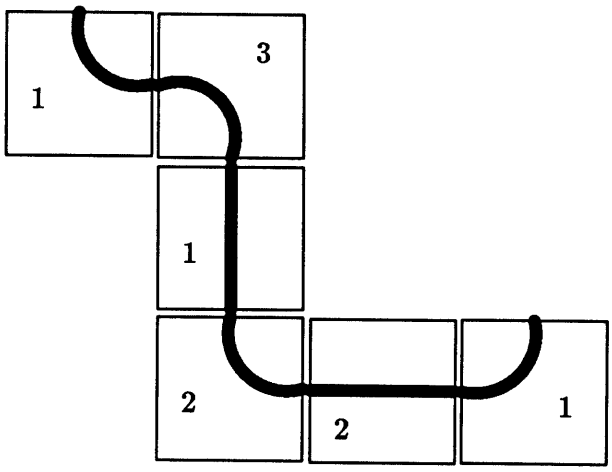

Figure C.3: A valid design.

the diagram represents a computational module. The left and right hand sections of the figure correspond to the two different design perspectives used to solve the problem. The following description is an overview of the program operation. The five paragraphs after the overview describe each part of the program in detail.

- The program accepts as inputs an initial design, the desired x and y location of the centroid, and the desired sum of the block numbers.

- Two separate abstractions of the initial device are made corresponding to the two separate design perspectives.

- The abstract descriptions of the initial designs are modified by debugging procedures associated with each separate perspective.

- The modified abstract descriptions are then instantiated into completely specified designs by instantiation procedures associated with each perspective.

- The instantiated designs from each perspective are collected together and evaluated according to how well they meet the two design goals.

- If no new design completely meets the design goals, one of the new designs is chosen as the next initial design and the process is repeated.

In this scheme, the debugging procedures are explicitly separated into two perspectives. These perspectives are organized such that the debuggers are not explicitly controlled–both debuggers are invoked on every iteration cycle.

C.4.3 Initial Design and Abstractions

The program accepts as input an initial design, a shape goal (the x-y location of the centroid), and a number goal (an integer value between 6 and 18). In this case the initial design is generated randomly, although in most mechanical design problems, the initial design would be generated by some other procedure. The first program operation is to create abstract descriptions of the initial design corresponding to the two design perspectives. In one case a geometrical abstraction is made which suppresses the numerical properties of the blocks. In the other case, a numerical abstraction is made which suppresses the geometrical properties of the blocks. These abstractions correspond (in figure C.4) to the left and right arrows leaving the initial design. The geometrical abstraction consists of a list of six pairs of directions corresponding to the input and output directions of the pipe segments associated with each of the six blocks. The numerical abstraction consists of a list of six integers corresponding to the integers associated with each of the six blocks.

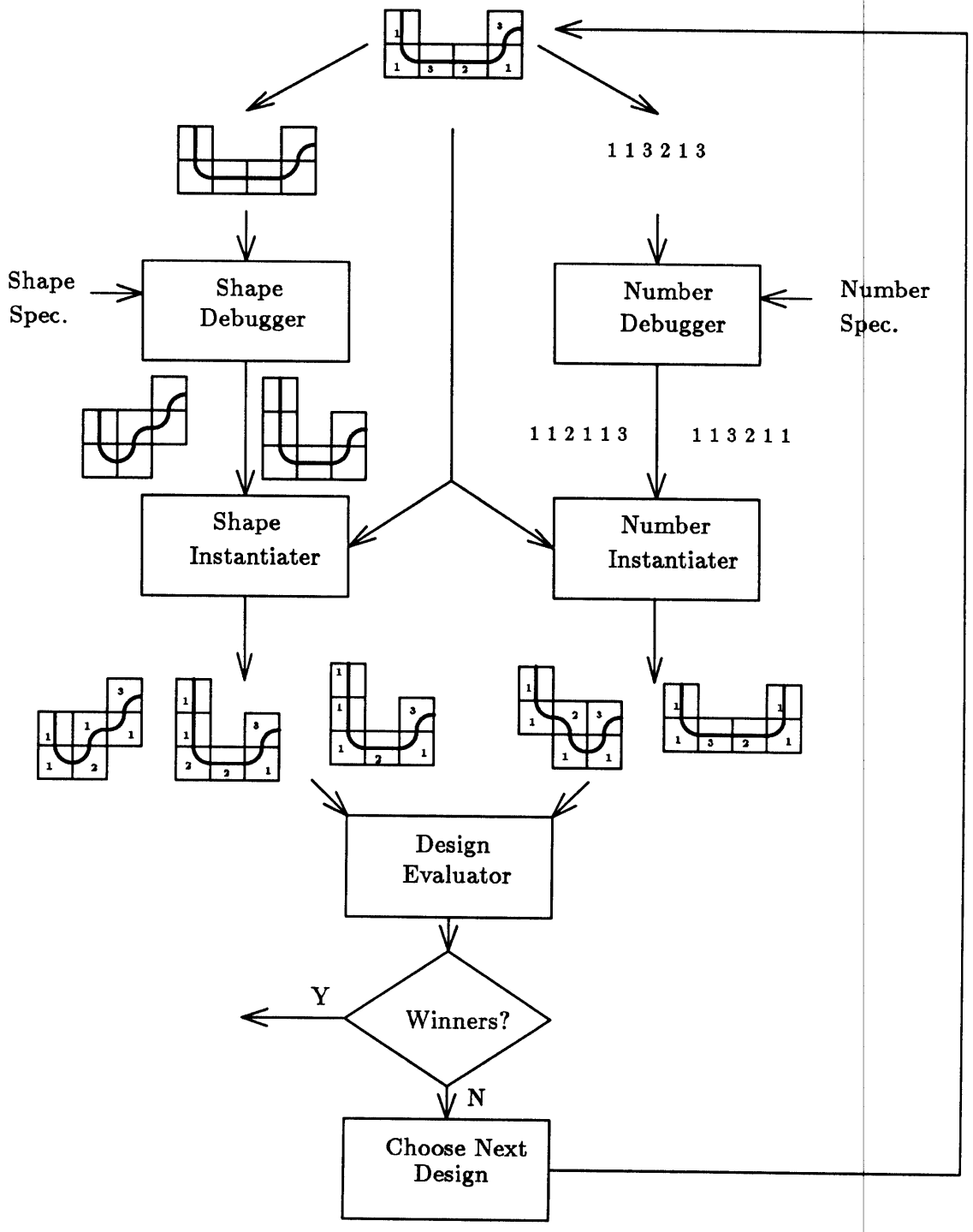

Figure C.4: Program architecture

C.4.4 Debugging Procedures

The debugging operations occur within separate design perspectives. In the geometrical perspective, a shape debugger takes as input the geometrical abstraction of the initial device and the shape specification (given as a program input). The debugger computes the difference between the desired centroid location and the actual centroid location and produces as output several geometrical abstractions of designs that have improved shapes (whose centroids are located closer to the specification). In the numerical perspective, a number debugger takes as input the numerical abstraction of the initial device and the number specification. The debugger computes the difference between the desired sum and actual sum and produces as output several numerical abstractions of devices that have improved numerical properties. In each case, the debuggers operate on abstract descriptions and produce abstract descriptions. The abstract descriptions they produce may or may not actually be realizable and may represent many different possible instantiations.

The design knowledge associated with the debuggers will vary from perspective to perspective. In the block design problem the shape debugger simply tries all possible substitutions of two adjacent blocks in the abstract block sequence description and keeps those descriptions that have improved centroid locations. The debugger changes two adjacent blocks because that is the smallest design change that can influence the location of the centroid. The shape debugger will typically produce about 20 abstract descriptions.

The number debugger makes changes to single numbers and pairs of adjacent numbers in the list of six numbers. The debugger makes all changes of these two types that improve the design sum. These changes are guided by the error between the design goal and the actual sum of the current abstract description. The number

debugger will typically produce about 10 abstract descriptions.

C.4.5 Instantiation procedures

The next step in each of the perspectives is to instantiate the abstract design descriptions that have been produced by the debuggers. Note that an abstract description could be instantiated many ways. For example a list of six integers corresponding to the numerical design perspective can represent many different complete designs (because there are several different blocks with the same numerical properties). The key idea behind the instantiation procedures is to only consider those design instantiations that could be the result of small changes to the initial device. This is an important step in reducing the space of possible designs, and the reason that the instantiation procedures accept as inputs the initial device descriptions as well as the abstract descriptions produced by the debuggers. The instantiation procedures simply compare the initial design description with an abstract description generated by the debugger. The design components that differ from the initial design are identified. The instantiater then finds all possible components (from those in figure 3) that have the same abstract description as the differing components in the design generated by the debugger. All of these components that are compatible (meet the interface constraints) with the initial device are substituted for the differing components in the initial device description.

C.4.6 Evaluation

Next, all of the instantiated designs are evaluated and ordered. The evaluation procedure must weigh the relative importance of meeting a goal in each of the design perspectives. For example, if the numerical goal is very easy to meet (if every different

shape of block is available in every numerical value) then closeness to the numerical goal should not be as important as closeness to the geometrical goal. In the example we describe, we use as the evaluation function, the sum of the x centroid error, the y centroid error, and the sum error.

C.4.7 Choosing the Next Design

If the best design produced in a debugging iteration does not meet all of the design goals then one of the new designs is chosen as a new initial design for another iteration. The new design can be chosen many different ways. Two procedures we have used work well. In one case, we simply choose at random from the best five new designs. In the other case, we keep the best five designs that have ever been produced since the first debugging iteration. The choice procedure picks randomly from these five. The introduction of some randomness helps to prevent program looping. The two choice strategies we describe are analogous to the search control strategies *hill climbing* and *best first.*

C.5 ANALYSIS

The blocks problem is a constructed domain. We have designed it to preserve the properties of conceptual design that are relevant to debugging to meet multiple goals, and to suppress some of the difficult implementation issues inherent in more realistic design problems. In this section we discuss the results of our implementation and justify our focus on perspectives as an organizational paradigm. Then, we analyze the assumptions both explicit and implicit in the blocks problem by outlining the criteria for application of the perspectives structure to other domains.

C.5.1 This Approach is Successful in the Blocks Domain

The concept of perspectives and our program architecture are very successful in the blocks design domain. A wide range of specifications for the centroid location and the integer sum were tried. Also, the assortment of types of blocks available as design components was varied. These changes control the relative difficulty of meeting the two design goals and the overall difficulty of the design problem. The problem difficulty can be adjusted from approximately 1 solution in 200,000 random designs to 1 solution in 50 random designs. In the easy case, the design program finds a solution 96 percent of the time, in an average of about 3 iterations. In the hardest case, the program finds a solution 30 percent of the time, in an average of about 25 iterations. These statistics are not important except to illustrate that if we can attack real design problems analogously, our programs should be successful. An important point is that the success of this particular program is dependent upon many factors including the power of the debugging procedures.

C.5.2 Perspectives Make Sense as an Organizational Tool for Enforcing Debugging Modularity

The concept of debugging from separate perspectives is important because it is an organizational tool for enforcing modularity. The shape debugger and number debugger are designed independently of the rest of the system. The only design knowledge located in the shape debugger is shape knowledge. The only design knowledge located in the number debugger is number knowledge. This makes the debuggers easy to design. If the debugger had to consider all perspectives at once, it would be nearly impossible to develop and to maintain. This organizational strategy makes design knowledge in one debugger applicable to a variety of problems. For example,

if we develop a geometry debugger that takes as input a network of two dimensional shapes and makes changes so that they fit in a specified two dimensional envelope, we can use it for office layout, circuit board layout, and simple mechanical design. To use this generic geometry debugger we only need write an ad hoc instantiation program that takes an initial design description and a debugged abstract description and produces possible design instantiations. This organizational strategy also makes the design knowledge explicit. It is easy to understand why a debugger does or does not work.

C.5.3 Criteria For Successful Application To Other Domains

In this section we describe the conditions under which our design strategy can be applied to other problems. Design problems to be approached with design and debug using perspectives need to have the following characteristics.

- There must be a multi-valued measure of design quality. This assumption corresponds to the ability to measure some sort of distance from each of the design goals. This is a requirement for any design technique in any domain that cannot be exhaustively explored. Most designs can be evaluated with more than a binary acceptance-rejection function, but formalizing these evaluation functions is often difficult.

- Debugging operations must correspond to small changes in the quality of the design. This is a requirement for any iterative approach. This property of debuggers and evaluation functions ensures that designs that almost meet design goals can be debugged in a single step. Problem formulations without this

property are difficult to solve because the evaluation function does not accurately reflect the degree of difficulty involved in debugging a design–a design with a poor evaluation may be one debugging step from success, while a design with a good evaluation may be several steps from success.

- There must be a formalizable set of debugging procedures for modifying an abstract description of a design.

- There must be a correspondence between the components used to build complete design descriptions and the primitive elements used to make abstract descriptions of designs. This is important since the debuggers make changes to abstract descriptions. These changes must correspond to changes that can be made to the complete design description.

C.5.4 The Cost of Using a Non-directed Control Strategy

Although the concept of perspectives is important in conjunction with any control structure, there is a constant-factor cost in computation time for the program architecture we have described in comparison to a debugging approach that explicitly controls the order in which designs are modified to meet goals. In our scheme, at each design iteration, each of the debuggers is invoked and all of the resulting designs are evaluated. This takes more computation time than if the focus of debugging were explicitly controlled. This cost in computation time is only a small constant multiplier (approximately 2 or 3) and results in a simpler program control structure.

C.6 EXAMPLE EXTENSION TO ANOTHER DOMAIN

The blocks design problem was an exercise to help understand debugging using perspectives. It has illuminated the requirements for the successful solution of other design problems. Our current efforts focus on the solution of single-input single-output dynamic systems design problems that require the satisfaction of multiple goals. We illustrate how such problems might be solved with a solution to the speedometer problem, and then discuss the current status of our efforts.

C.6.1 The Speedometer Revisited

Imagine that we have simplified the speedometer problem to consist of the two goals that there must be a linear relationship between the angular deflection of the indicator needle and the angular velocity of the input shaft, and that the needle must be located at the dashboard while the input shaft is at the transmission. First imagine that the system is given an initial design consisting of the speedometer needle connected directly to the input shaft. One perspective corresponds to the dynamic behavior of the device. The other perspective corresponds to the geometry of the device. The behavior debugger uses a dynamic system abstraction of the device and observes that the needle-shaft relationship is linear between the *velocity* of the needle and the input velocity. The debugger invokes a dynamic system debugging rule that says if a device measures the derivative of the desired quantity, then add compliance from the output to ground, and viscous damping between the input and the output. The shape debugger observes that the device does not connect the geometrical location of the input to the geometrical location of the output, and proposes that

something long and flexible be placed between input and output. These changes to the device abstractions are instantiated, and evaluated. (See figure C.5).

The design with a spring and damper added is chosen for the next iteration. The behavior debugger is satisfied that the design meets the behavior spec. The geometry debugger observes that the design does not geometrically connect input to output and invokes the long flexible component modification again. On this iteration, one instantiation of this modification is a device with a flexible shaft between the input shaft and the damper. This design meets all of the goals and the design is completed. In this scenario we have only illustrated one of the many possible outcomes of the design procedure. In practice other successful designs would probably be generated.

C.6.2 Status of Research

All of the work relating to the blocks domain has been implemented. We have implemented chunks of these ideas in the domain of single-input single-output dynamic systems. In particular, we have developed a technique for generating a description of a system with correct input-output behavior [1].

C.7 RELATED RESEARCH

This section is presented only as a set of pointers to related literature, rather than a comprehensive discussion of related research efforts.

Sussman introduced the concept of design and debug in his doctoral thesis on planning [2]. Davis used the concept of multiple device representations to do electronic troubleshooting [3]. We have explored the idea of using knowledge of existing devices to design new devices in the domain of mechanical fasteners. This approach may be a good way to generate an initial device description that is subsequently

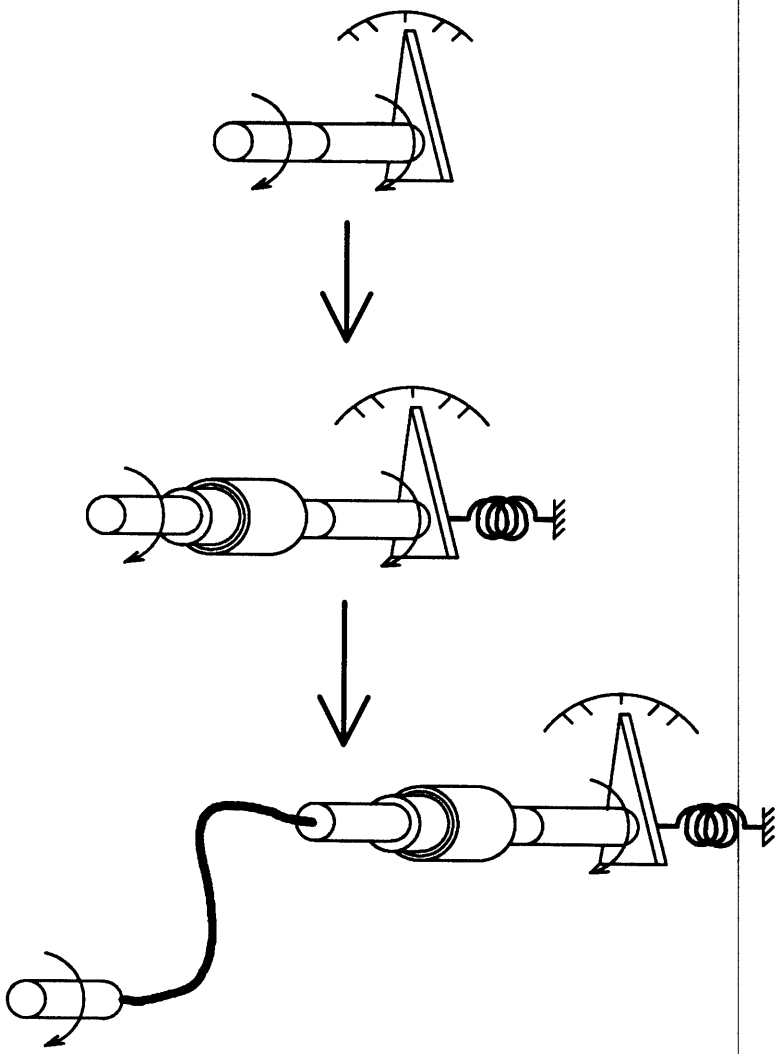

Figure C.5: Speedometer debugging

debugged [4]. Williams uses qualitative reasoning about causal mechanisms in electronic devices to guide debugging [5].

C.8 REFERENCES

1. Ulrich, K.T., *Computation and Pre-Parametric Design,* Massachusetts Institute of Technology Artificial Intelligence Laboratory Technical Report 1043, August 1988.

2. Sussman, G.J., *A Computer Model of Skill Acquisition,* Elsevier, Inc., 1975.

3. Davis, Randall, "Diagnostic Reasoning Based on Structure and Behavior," in *Qualitative Reasoning About Physical Systems,* D.G. Bobrow ed., The MIT Press 1985.

4. Ulrich, K.T. and W.P. Seering,"A Computational Approach to Conceptual Design," *Proceedings of the International Conference on Engineering Design,* August 1987, Boston, MA.

5. Williams, B.C., "Principled Design Based on Topologies of Interaction," Massachusetts Institute of Technology Department of Electrical Engineering and Computer Science, Ph.D. Thesis, 1988.

What Makes a Mechanical Design Interesting?

Appendix D

This entire research project was initially motivated by a desire to computationally generate interesting designs in order to aid engineers in the early stages of a design project. As a result of this objective, I gave a moderate amount of thought to what designers mean by *interesting designs.* I use the word interesting as a synonym for creative or clever, and mean it to also include the concept of elegance and novelty.

Generating interesting designs is desirable not only because these designs will often be better designs than the obvious solution, but also because they lead to a new way of thinking about the problem. Too many alternatives can swamp a designer's evaluation capacity, however in industrial practice this problem rarely materializes— designers typically generate less than 5 alternative concepts for a particular problem. A designer can always arbitrarily truncate a set of possible design concepts, so computationally generating additional interesting concepts can never hurt.

This appendix chapter is the result of my preliminary thinking on what factors

contribute to making a design interesting. Hopefully it will provide some seeds from which other researchers will derive useful direction. This chapter is not meant as a strong claim of any kind, but more as a few ideas that are perhaps worth the reproduction costs required to preserve them.

D.1 ORIGIN OF THESE CLASSIFICATIONS

My (admittedly informal) method for generating a classification of interesting designs was to list 100 designs that I thought were interesting and then to try to devise a scheme that would categorize the sources of interestingness in the designs. My goal was to develop a classification that would divide the designs into a small number of groups. These categories are clearly artifacts of my own perception of what makes a design interesting, since the 100 initial designs were the result of my own thinking. Some of the designs that I think are interesting are very common— the bicycle wheel for example. My criteria for interestingness are based on the invention or initial appearance of the design. In other words, I would have considered the bicycle wheel a very clever design when it was first developed.

D.2 FOUR CATEGORIES OF INTERESTING DESIGNS

Four categories classify my 100 interesting designs. In this section, I devote a subsection to each category. Each subsection will describe the category and give several examples.

D.2.1 Novel Combination

Novel combination is the result of extracting individual structural attributes from each of several known devices and combining these attributes in novel ways to create structural descriptions of new devices (Appendix B is devoted to this idea). The following three examples illustrate this concept. In some of these examples I invent a historical myth that could account for the development of the design.

Fastener

Figure D.1 is a drawing of a plastic fastener used in high-volume manufactured parts. It is the result of combining the ease-of-insertion attributes of a barbed christmas tree fastener and the removability attributes of a threaded fastener. Because its threads are flexible it can be pressed into a hole; because it has threads it can be unscrewed. (This example is discussed in more detail in Appendix B).

Bicycle wheel

The bicycle wheel combines several attributes from other designs in new ways. First, it uses the cushioning properties of inflated flexible chambers (from cushions, pads, or balls perhaps). Second, the concept of suspending the bicycle hub from the top of the upper half of the rim with flexible tension elements (spokes) is borrowed from suspension bridge design. Combining these design fragments together yields the modern bicycle wheel.

Threading tap

A threading tap can be thought of as a cutter along a helical path. The concept of a helical path comes from the screw itself. The cutter design incorporates gradually

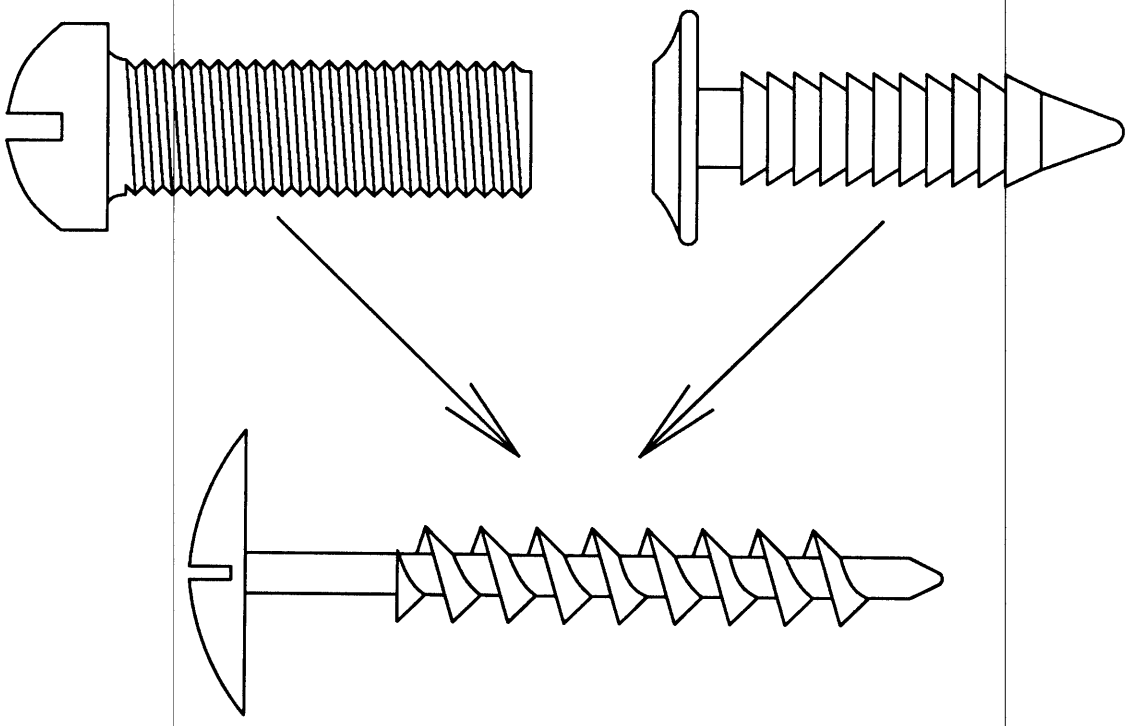

Figure D.1: Easily insertable and removable fastener

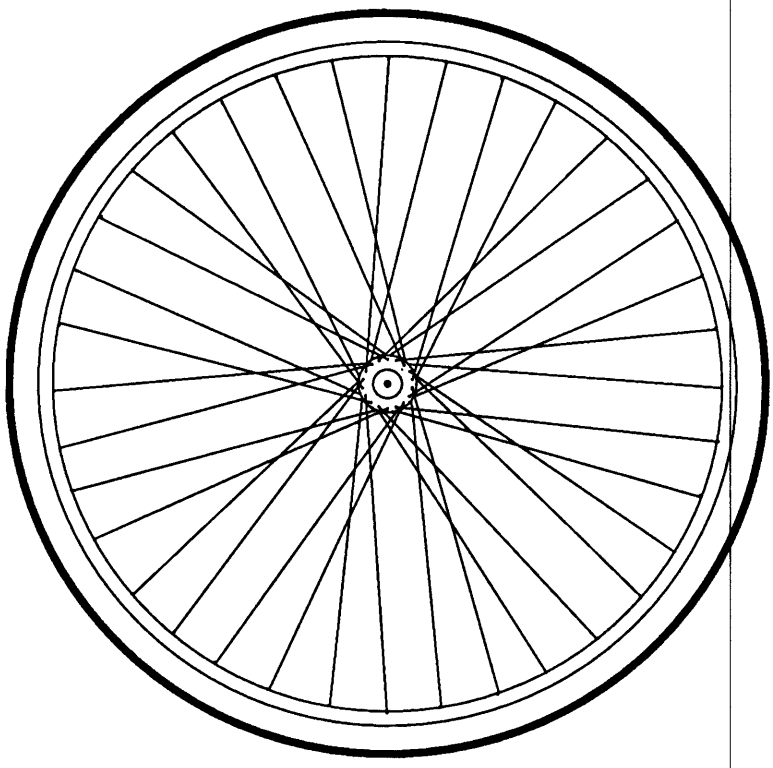

Figure D.2: Bicycle wheel

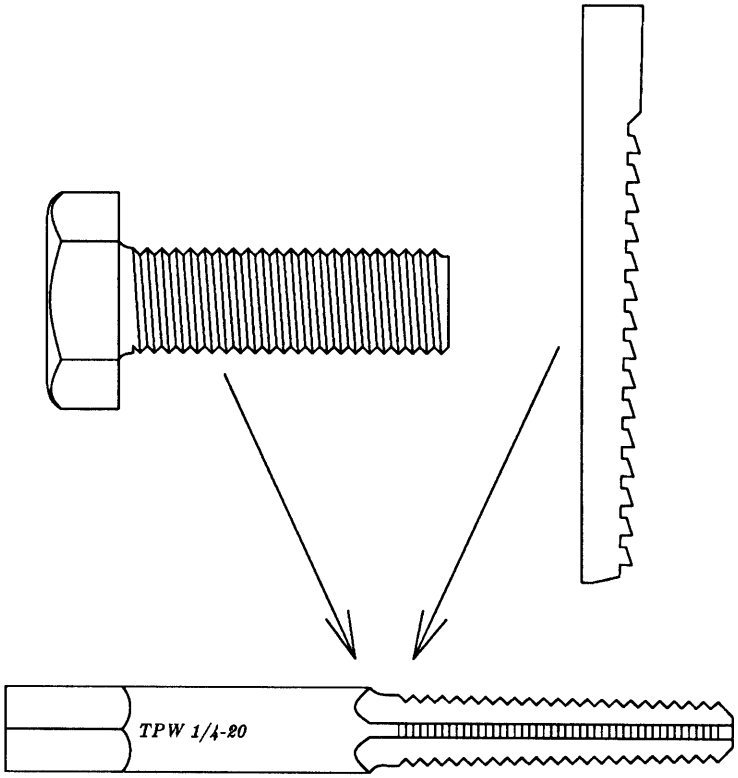

Figure D.3: A threading tap

increasing tooth sizes, an idea that could come from the keyway broach. This design is not at all obvious as indicated by the surprise of people who first see how threaded holes are made.

D.2.2 Geometrical Exploitation

Geometrical exploitation is a category of design in which a particular structural configuration is found that, because of the way its properties change with geometry, is able to meet two or more seemingly conflicting specifications. These designs exhibit a kind of shape optimization.

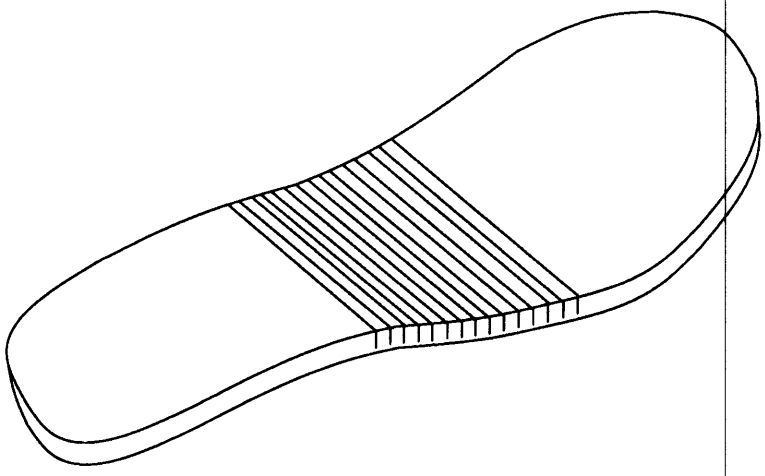

Figure D.4: Novel ski boot sole

Cross-country ski boot sole

Figure D.4 illustrates the design of a novel ski boot sole (designed by Steven D. Potter while a graduate student at MIT). Cross-country ski boots ideally are very flexible in straight bending, but are very stiff in torsion. This sole design achieves those seemingly contradictory objectives by making many narrow parallel slits most of the way through the top side of the sole and perpendicular to the major axis of the boot. This configuration is remarkably stiff in torsion (approaching the stiffness of a solid sole of the same material), while extremely flexible in bending.

I-beam

The I-beam is designed such that most of the cross-sectional area of the beam is located a great distance from the neutral bending axis. This is accomplished by using a thin web to separate two flanges. This configuration achieves high bending strength and stiffness with minimum weight. See figure D.5.

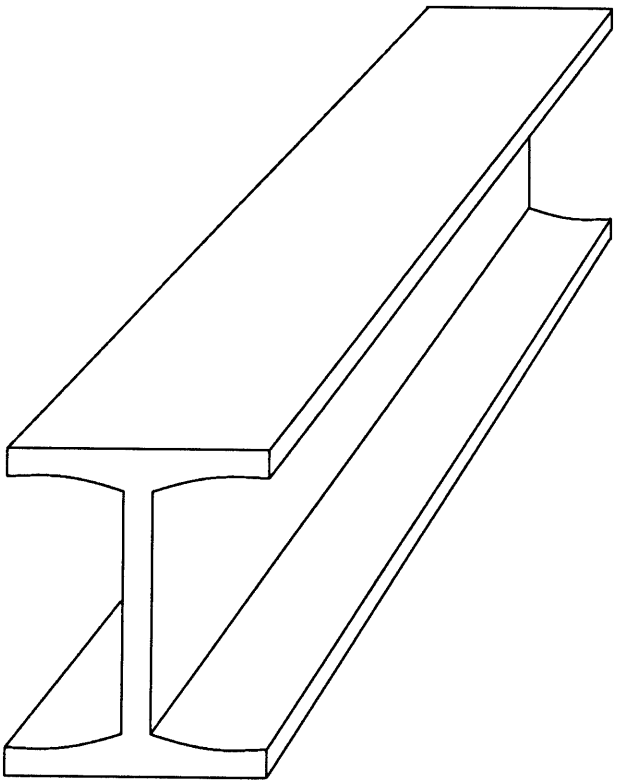

Figure D.5: I-beam cross section

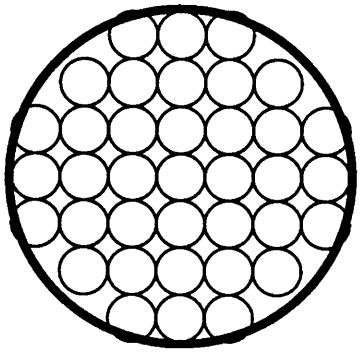

Figure D.6: Stranded cable

Stranded cable

The stranded cable is a design that achieves both high tensile strength and great flexibility. These goals are met by maximizing the cross-sectional area of the cable, while minimizing the bending stiffness of any particular strand. The result is an almost full circular cross section made of thin flexible strands.

D.2.3 Application of Natural Phenomena

Many interesting devices exploit some natural phenomenon. These phenomena are often material properties. Three examples are strain gages, thermocouples, and fluorescent lights.

Strain gage

The strain gage is a device for measuring the strain (or fraction of change in length over original length) of a surface. Most strain gages are based on the fact that wire changes in resistivity as it is stretched and that this change is proportional to the strain in the wire. Wire strain gages consist of several zigzags of fine wire bonded

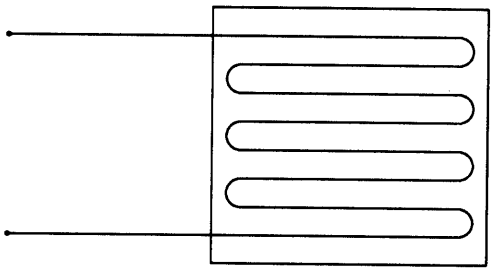

Figure D.7: Strain gage

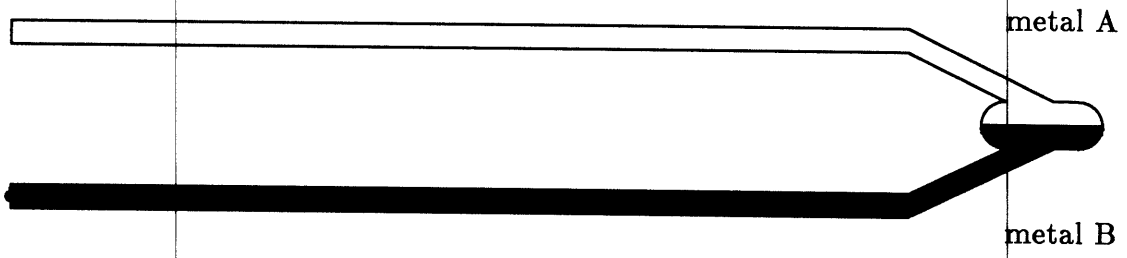

Figure D.8: Thermocouple

to a paper backing. They are glued to a surface and integrated into a circuit that measures changes in resistance.

Thermocouple

Thermocouples produce a voltage monotonically related to a temperature difference. They are based on the thermoelectric effect— junctions of dissimilar metals produce a voltage in response to temperature differences. This principle is exploited in order to build robust, inexpensive and accurate temperature sensors. The principle can also be used in the opposite direction to create a solid-state heat pump that creates a temperature difference in response to an applied potential.

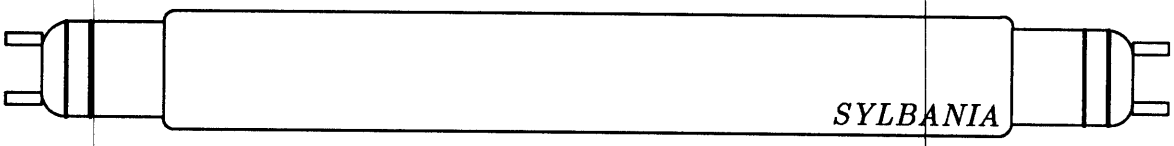

Figure D.9: Fluorescent lights

Fluorescent Lights

Fluorescent lights are based on the fact that some gases emit photons when they occupy the space between an anode and a cathode. This property is exploited to create efficient, uniform lighting in a variety of shapes.

D.2.4 Function Sharing

Function sharing is a design attribute that involves the implementation of several functions with a single functional element (chapter 4). Most designs exhibit some function sharing— these examples illustrate the concept especially well.

Resistive heating crucible

Some crucibles used in metallurgy are formed from graphite and are used as the resistive element in a resistive heating system. Figure D.10 illustrates this idea. In this case, the crucible is serving both as a container and a heating element.

Automobile body

The sheet metal body of an automobile performs many functions including structural support and the automobile electrical ground.

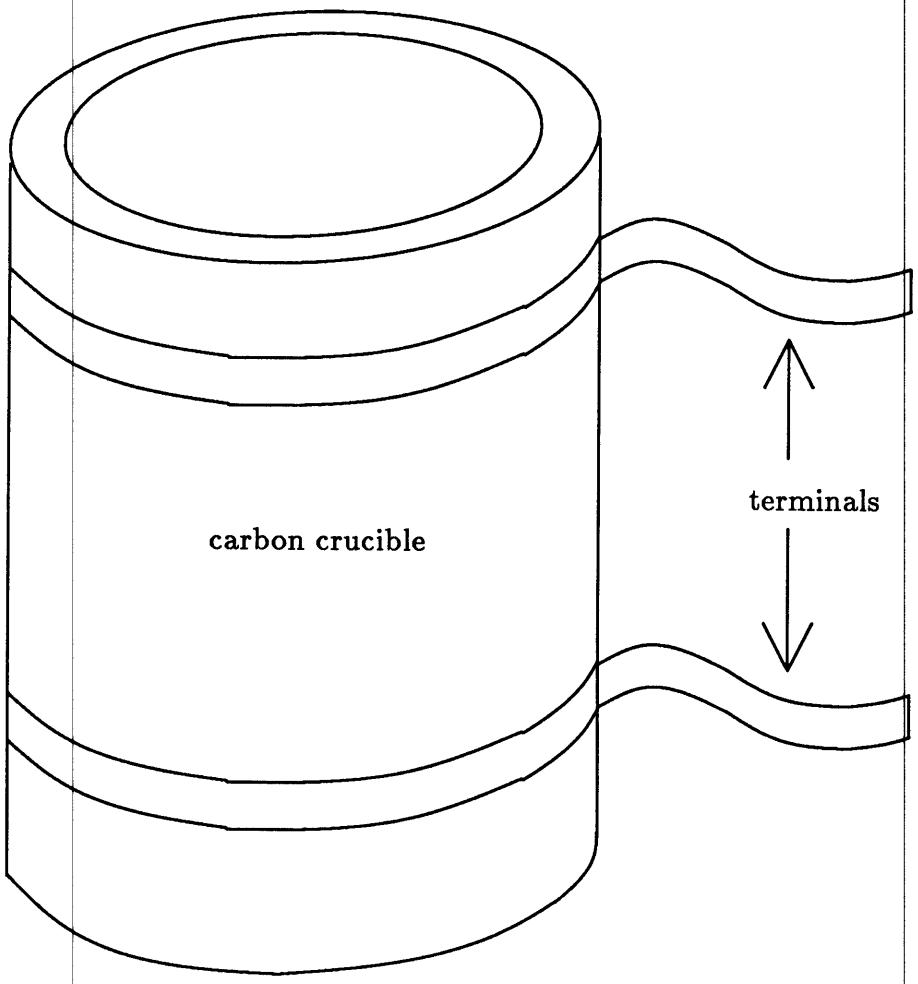

Figure D.10: Resistive heating crucible

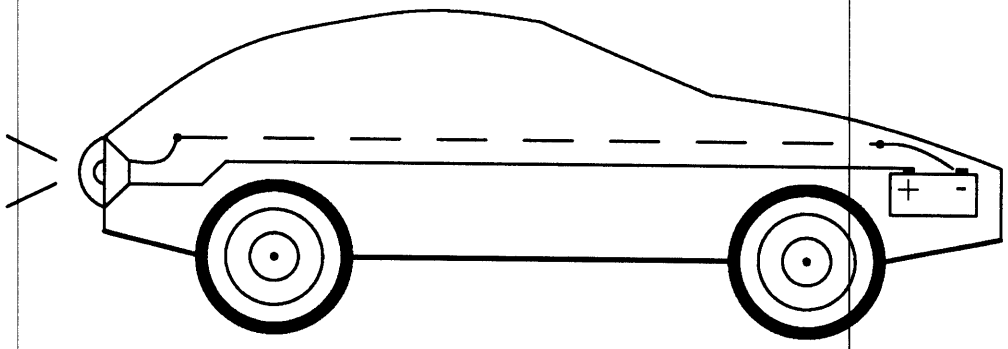

Figure D.11: Automobile body as electrical ground

Chip leads

A computer chip package usually has several leads that provide an interface to the rest of the circuit. These leads serve as electrical connections, mechanical attachment points, and thermal conduction paths.

D.3 USING THESE IDEAS

These ideas have been useful to me in two ways. First, they have yielded insights into how to teach designers to generate interesting design concepts. Second, they have spawned research directions.

D.3.1 Teaching Design

These categories of design interestingness could become prescriptions for engineers performing innovative design. These prescriptions can be much more vague than those required to implement a computer program. They might be questions or suggestions of the following form:

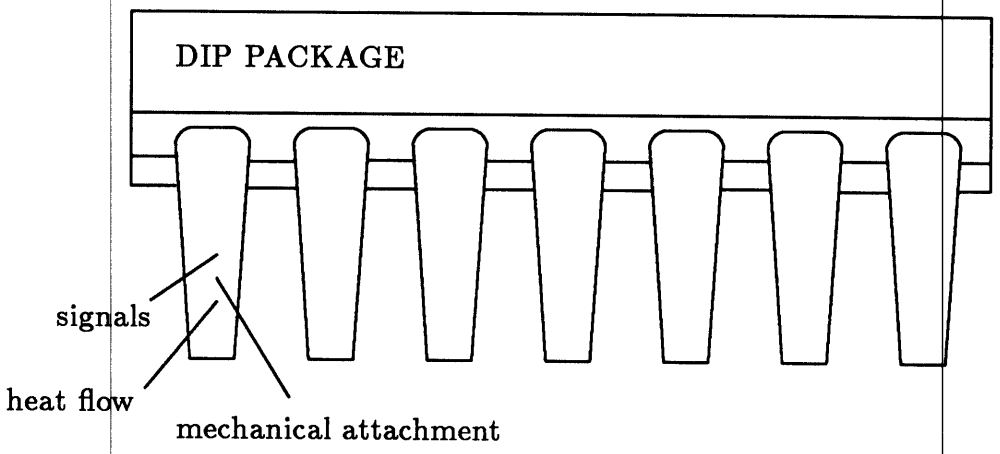

Figure D.12: Digital circuit chip leads

- Can a part be eliminated by using the secondary properties of a neighboring part to perform its function?
- Look for ways to combine two parts into one.
- Can you think of another way to solve part of the problem? Is there an existing product or machine that achieves some of the important functions of the desired device? Can attributes of these existing devices be combined to meet the specifications of the new device?
- When designing an instrument or sensor, look for a physical effect that relates some quantity of interest to something that you can measure.

D.3.2 Research Directions

The novel combination work with fasteners (Appendix B), and the function sharing work in the dynamic systems domain (chapter 4) were both the result of the thinking that led to this chapter.

Some of the other categories of interesting design provide possible future research areas. The geometrical exploitation category seems like a tractable and potentially fruitful problem; since most of the reasoning is done within a mathematical representation of the properties of the structure. That is, given a mathematical parameterization of the structure of a device, are there structural modifications that optimize the device behavior.

The physical phenomena category also seems to be amenable to some sort of automation. Because there are thousands of physical effects that could be cataloged, many useful devices could perhaps be generated simply by scanning the available effects and trying to match them to the problem at hand. In the 1950's Hix wrote a book that catalogs hundreds of unusual physical phenomena [Hix58].

REFERENCES

Antonsson87 Antonsson, E.K., "Development and Testing of Hypotheses in Engineering Design Research," *Journal of Mechanisms, Transmissions, and Automation in Design*, Vol. 109 (June 1987), p. 153-154.

Cagan87 Cagan, Jonathan and Alice M. Agogino, "Innovative Design of Mechanical Structures from First Principles," Intelligent Systems Research Group Working Paper 87-0801-2, Department of Mechanical Engineering, University of California at Berkeley.

Davis84 Davis, R., "Diagnostic Reasoning Based on Structure and Behavior," Massachusetts Institute of Technology Artificial Intelligence Laboratory Memo 739, 1984.

de Kleer85 de Kleer, J. and J.S. Brown, "A Qualitative Physics Based on Confluence," in Bobrow, D.G. (ed.), *Qualitative Reasoning About Physical Systems*, MIT Press 1985.

Doyle88 Doyle, R.J., "Hypothesizing Device Mechanisms: Opening Up the Black

Box," Massachusetts Institute of Technology, Artificial Intelligence Laboratory Technical Report 1047, 1988.

Freeman71 Freeman,P. and A. Newell, "A Model for Functional Reasoning in Design," Proceedings of the Second International Joint Conference on Artificial Intelligence, p. 621-633, 1971.

Frieling77 Frieling, M.J., "The Use of a Hierarchical Representation in the Understanding of Mechanical Systems," Ph.D. Thesis, Massachusetts Institute of Technology Department of Mathematics 1977.

Gero87 Gero, J.S. and M.L. Maher, "Future Roles of Knowledge-Based Systems in the Design Process," Carnegie Mellon University Engineering Design Research Center Report EDRC-12-13-87, Pittsburgh, PA 1987.

Glegg73 Glegg, G.L., *The Science of Design*, Cambridge University Press, 1973.

Glegg69 Glegg, G.L., *The Design of Design*, Cambridge University Press, 1969.

Hirschtick86 Hirschtick, J. "Geometric Feature Extraction Using Production Rules," S.M. Thesis, Massachusetts Institute of Technology Department of Mechanical Engineering, 1986.

Hix58 Hix, C.F. Jr., and R.P. Alley, *Physical Laws and Effects,* John Wiley and Sons, Inc., NY 1958.

Hood87 Hood, S.J. and E.R. Palmer, "The Design Analysis for Reliability Tool," IBM Corporation, T.J. Watson Research Center Technical Report RC 12647 (#56882) 4/6/87, 1987.

Ishida84 Ishida, T., H. Minowa, and N. Nakajima, "Detection of Unanticipated Functions of Machines," Proceedings of the International Symposium on Design and Synthesis, p. 21-26, July 1984, Tokyo, Japan.

Jakiela87 Jakiela, M.J., P.Y. Papalambros, "Design and Implementation of an Intelligent CAD System," Proceedings of the 1987 ASME Design Automation Conference, Vol. 1, p.339-345.

Jansson87 Jansson, D.G., "Creativity in Engineering Design: The Partnership of Analysis and Synthesis," Proceedings of the 1987 ASEE conference.

Kannapan87 Kannapan, Srikanth and Kurt M. Marshek, "Design Synthetic Reasoning: A Program For Research," Mechanical Systems and Design Technical Report 202, University of Texas at Austin, Department of Mechanical Engineering.

Lenat78 Lenat, D.B., "The Ubiquity of Discovery," *Artificial Intelligence* 9 (1978), 257-285, North-Holland Publishing Company.

Lenat84 Lenat, D.B. "Why AM and EURISKO Appear to Work," *Artificial Intelligence* 23 (1984) 269-294, North-Holland Publishing Company.

Macsyma86 *Macsyma Reference Manual*, Symbolics, Inc., Cambridge, MA 1986.

Maher87 Maher, M.L., "A Knowledge-Based Approach to Preliminary Design Synthesis," Carnegie Mellon University Engineering Design Research Center Report EDRC-12-14-87, Pittsburgh, PA, 1987.

Murthy87 Murthy, S.S. and S. Addanki, "PROMPT: An Innovative Design Tool," Proceedings of the Sixth National Conference on Artificial Intelligence, Seattle, WA 1987.

Ogata70 Ogata,K., *Modern Control Engineering,* Prentice-Hall, 1970. p. 284.

Paynter61 Paynter, H.M., *Analysis and Design of Engineering Systems*, The MIT Press, Cambridge, MA 1961.

Penberthy87 Penberthy, J.S., *Incremental Analysis and the Graph of Models: A First Step Towards Analysis in the Plumber's World*, S.M. Thesis, Massachusetts Institute of Technology Department of Electrical Engineering and Computer Science, 1987.

Prabhu88 Prabhu, D. and D.L. Taylor, "Some Issues in the Generation of Topology of Systems with Constant Power-Flow Input-Output Requirements," Submitted to 1988 ASME Design Technologies Conference, Kissimmee, Florida, September 1988.

Ressler84 Ressler, A.L., "A Circuit Grammar for Operational Amplifier Design," Massachusetts Institute of Technology Artificial Intelligence Laboratory Technical Report 807, January 1984.

Rieger77 Rieger, C. and M. Grinberg, "The Declarative Representation and Procedural Simulation of Causality in Physical Mechanisms," Proceedings of the Fifth International Joint Conference on Artificial Intelligence, Vol. 1, p. 250, 1977.

Rinderle87 Rinderle, J.R. and J.D. Watton, "Automatic Identification of Critical Design Relationships," Carnegie Mellon University Engineering Design Research Center Report EDRC-24-03-87, Pittsburgh, PA 1987.

Rinderle87 Rinderle, J.R., "Function and Form Relationships: A Basis for Preliminary Design," Carnegie Mellon University Engineering Design Research Center

Report EDRC-24-05-87, Pittsburgh, PA 1987.

Rodenberger83 Rodenberger, C.A., C.F. Herndon, S.O. Majors, and W.A. Rogers, "The Average $100,000,000 Design Engineer," Proceedings of the 1983 ASME Computers in Engineering Conference, Vol. 1, p. 33-38.

Rosenberg83 Rosenberg, R.C., and D.C. Karnopp, *Introduction to Physical System Dynamics*, McGraw-Hill Book Company, 1983.

Roylance83 Roylance, G., "A Simple Model of Circuit Design," Massachusetts Institute of Technology Artificial Intelligence Laboratory Technical Report 703, 1983.

Serrano87 Serrano, D. and D. Gossard, "Constraint Management in Conceptual Design," in Sriram, D. and R.A. Addy (Eds.), *Knowledge-Based Expert Systems in Engineering: Planning and Design.* Computational Mechanics Publications 1987.

Simon81 Simon, H.A., *The Sciences of the Artificial*, MIT Press, Second edition 1981, Cambridge, MA.

Suh78 Suh, N.P., A.C. Bell and D.C. Gossard, "On an Axiomatic Approach to Manufacturing and Manufacturing Systems," *Journal of Engineering for Industry*, May 1978, Vol. 100, p.127-130.

Sussman80 Sussman, G.J. and G.L. Steele Jr., "Constraints– A Language for Expressing Almost-Hierarchical Descriptions," *Artificial Intelligence* 14 (1980), 1-39, North-Holland Publishing Company.

Ward88 Ward, A.C., "Quantitative Inference in a Mechanical Design Compiler," Massachusetts Institute of Technology Artificial Intelligence Laboratory Memo

1062, 1988.

Williams88 Williams, B., "Principled Design Based on Topologies of Interaction," Massachusetts Institute of Technology Department of Electrical Engineering and Computer Science, Ph.D. Thesis, 1988.

Winston83 Winston, P.H. et al, "Learning Physical Descriptions From Functional Definitions, Examples, and Precedents," Massachusetts Institute of Technology Artificial Intelligence Laboratory Memo 679, 1983.