Invention From First Principles via Topologies of Interaction

by

Brian Charles Williams

S.B., Massachusetts Institute of Technology

1981

S.M., Massachusetts Institute of Technology

1984

Submitted to the

Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

Massachusetts Institute of Technology

June, 1989

Signature of Author _____

Department of Electrical Engineering and Computer Science
April 25, 1989

Certified by _____

Randall Davis
Professor of Management Science
Thesis Supervisor

Accepted by _____

Arthur C. Smith, Chairman
Departmental Committee on Graduate Students

A Theory of Invention from Fundamental Principles of Physics Based on Topologies of Interaction

by

Brian Charles Williams

Submitted to the Department of Electrical Engineering and Computer Science
on April 25, 1989 in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in
Artificial Intelligence

Abstract

One hallmark of intelligence is the ability to create. This thesis explores one aspect of creativity – invention. The central question addressed is – how might an inventor use his knowledge of fundamental principles of physics (called first principles) to construct innovative devices? An inventor who is skilled at constructing innovative designs is distinguished, not by the first principles known, but by the way he uses these principles and how he guides the search for novel devices among an overwhelming space of possibilities. The goal of this research is a principled theory that accounts for the novel design of continuous physical systems.

In this thesis we view the art of design from a historical perspective, analyzing inventions from the past to gain insight into how modern inventions might arise. To understand how simple devices arise from first principles, we have analyzed a set of ancient fluid devices (300 B.C.), that were fundamental to the development of feedback control. To understand how simple devices evolve into sophisticated designs, we have explored a progression of high performance MOS buffer designs that played a fundamental role in the development of high performance dynamic memories.

Central to our theory is a representation that focuses the search for innovative designs by highlighting fundamental differences between devices in terms of how they work. A device achieves a desired behavior by constructing a set of interactions between quantities and then modulating these interactions over time. Thus to identify fundamentally different ways of producing a desired behavior, the designer focuses on the time-varying topologies of interactions between quantities that capture how physical devices work.

Our approach (called Interaction-based Invention) involves constructing a topology of interactions that both produces the desired behavior and makes evident a structural topology of physical components that implements those interactions. Our approach to focusing the search for innovative devices combines two strategies that are based on the following insights:

- To focus the search for a desired interaction using only first principles, we first produce a coarse solution by exploring an abstract search space. This

involves identifying paths along a topology of potential interactions producible by available structures. This solution is then refined so that the combined behavior of interactions along the path produce the behavior of the desired interaction.

- Sophisticated devices evolve through a sequence of focused innovations. At each step a designer pushes a leading edge device until it breaks, and then uses the insight gained about the cause of failure to identify critical points where innovative changes are required to the device's interaction topology.

The development of these strategies has resulted in several advances in qualitative, causal, temporal and terminological reasoning. These include a hybrid qualitative/quantitative algebra (Q1), a qualitative symbolic algebra system (Minima), a theory of causal accounts as tracing dominant flows through a device (causal dominance), a representation and simulation system for describing the dynamics of devices through local interactions (concise histories), and a terminological reasoning system that supports "constructive reasoning" (Iota).

The plausibility of this approach is demonstrated by a program, called Ibis, that designs simple hydro-mechanical devices from first principles using a lumped element model. With guidance, Ibis has been able to reconstruct designs similar to ancient fluid regulation devices that were fundamental to the development of modern feedback control theory.

Thesis Supervisor: Randall Davis
Title: Associate Professor of Management Science

Acknowledgments

I would like to thank:

Randall Davis, my thesis advisor, who taught me about turning rough stones into gems. For endless stimulating discussions, encouragement, generosity, infinite patience, a saying or two, and a most excellent mustache to admire.

Members of my committee, Patrick Winston and Tomas Lozano-Perez, who asked difficult questions at just the right times. This would be a distinctly different thesis without them.

Johan de Kleer, Ron Brachman and Hector Levesque, for being both my mentors and friends, at different points in my AI career. Each has shown me how many different perspectives there are, and that all are important.

Members of the HT group, Jeff Van Baalen, Dan Weld, Mark Shirley, Walter Hamscher, Reid Simmons, Peng Wu, and Paul Resnick, for countless discussions, moral support, and most importantly, true friendship.

My parents, Carol and Herm Wickersham, and the late Robert C. Williams, for teaching me that anything is possible.

Leah Williams, who has supported me in every part of my life.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the author's artificial intelligence research is provided in part by an Analog Devices Fellowship, the Digital Equipment Corporation, Wang Corporation, Xerox Corporation and the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-85-K-0124.

To Leah

Contents

1 Introduction 12
- 1.1 What Makes a Device Innovative . . . 16
- 1.2 Why Invention? . . . 23
- 1.3 The Task Explored . . . 25
- 1.4 Elements of the Invention Process . . . 27
- 1.5 Interaction-based Invention . . . 29
- 1.6 First Principles of Interaction-based Invention . . . 32
- 1.7 Strategies for Invention . . . 35
- 1.8 Representing Interaction Topologies . . . 45
- 1.9 An Inventor's Procedural Skills . . . 47
- 1.10 Road Map . . . 50

I Strategies for Invention 52

2 Interaction-based Invention 53
- 2.1 Detailed Account of the Punch Bowl Design . . . 53
- 2.2 Interactions . . . 57
- 2.3 Invention as Transforming Interactions . . . 63
  - 2.3.1 The Transformations . . . 64
  - 2.3.2 Rationale for the Transformations . . . 67
  - 2.3.3 Analogy to FSM Design . . . 68
- 2.4 What the Transformation Process Entails . . . 70
  - 2.4.1 First Transformation . . . 71
  - 2.4.2 Second Transformation . . . 75
  - 2.4.3 Third Transformation . . . 79
- 2.5 Summary . . . 92

3 Tracing Paths of Interaction 94
- 3.1 Motivation for the Approximate Search Space . . . 96
  - 3.1.1 Properties of the Devices Being Invented . . . 96
  - 3.1.2 The Most Constraining Properties . . . 97
  - 3.1.3 The Eliminated Cost . . . 99
- 3.2 Representing What Interacts . . . 100
  - 3.2.1 Background: Describing Structure . . . 100
  - 3.2.2 Pictorial Descriptions of Interactions . . . 101
  - 3.2.3 Primitive Topologies for Models and Laws . . . 104

3.2.4 Using Primitive Topologies to Build Descriptions 107
3.3 Using a Topology to Represent the Space of All Interactions 109
3.3.1 The Topology of Existing Interactions 110
3.3.2 The Topology of Potential Interactions 114
3.3.3 Constructing the Complete Space of Interactions 122
3.4 Invention as Augmenting Interaction Topologies 124
3.5 Applying the Strategy to the Punch Bowl Problem 130
3.5.1 Constructing the Search Space 131
3.5.2 Constructing the Coarse Solution 131
3.5.3 Augmenting the Device's Interaction Topology and Physical Structure 136
3.5.4 Testing if the Structure is Physically Realizable 148
3.5.5 Testing if the Desired Interaction is Produced 148
3.6 Refining Failed Solutions 151
3.7 Examples of Ancient Inventions 162
3.8 Path Tracing Criteria: Rationale and Limitations 178
3.8.1 Rationale for the Path Tracing Criteria 178
3.8.2 Limitations of the Path Tracing Approach 180
3.9 Limitations and Future Work 182
3.10 Summary 186

4 Conclusion **189**
4.1 Contributions 189
4.2 Related Design Work 192
4.2.1 Design Compilation 194
4.2.2 Library Design 195
4.2.3 Extensions to Basic Library Design 195
4.2.4 Design Debugging 196
4.2.5 Approaching Design from First Principles 199

II Representations, Principles and Procedural Skills 203

5 Representing Intensional Interactions: The Q1 Algebra 204
5.1 Requirements for a Good Representation of Interactions 206
5.2 A Qualitative Algebra on Signs 208
5.3 Inadequacy of Existing Qualitative Algebras 210
5.4 Q1 – A Hybrid Algebra 212
5.5 Semantics of Q1 Equations 214
5.6 Using Q1 to Define Traditional Qualitative Representations 217
5.7 Summary 224

6 Minima: Qualitative Algebraic Skills 225
6.1 A Controversy and Highlights of the Solution 226
6.2 First Principles of Q1 228
6.3 Consequences for Manipulating Interactions 230
6.3.1 Weaknesses of $\mathcal{S}'$ 230
6.3.2 Commonalities Between $\mathcal{S}'$ and $\Re$ 231

6.3.3 Properties of $\mathcal{S}'$ not in $\Re$ 233
6.3.4 Relating $\mathcal{S}'$ and $\Re$ 234
6.4 MINIMA – Algebraic Manipulation Skills 236
6.4.1 Simplification and Canonicalization 238
6.4.2 Equation Solving and Factoring 248
6.5 Example: Composing Interactions 250
6.6 Summary 251

7 First Principles of Physics **254**
7.1 The Appropriate Abstraction for Principles 254
7.2 Notation for Principles 256
7.3 First Principles of Hydraulics 259

8 Iota: Constructive Reasoning Skills **265**
8.1 Iota – Abstract Specification 268
8.1.1 Objects Manipulated 269
8.1.2 Queries 269
8.1.3 Definitions and Assertions 271
8.2 Using Iota to Connect Interactions 277
8.3 Algorithms for Answering Queries 282
8.4 Summary 291

List of Figures

1.1 Philon's oil lamp. 13
1.2 Simplified topology of interactions for Philon's oil lamp. 13
1.3 Interaction-based Invention applied to Philon's lamp. 15
1.4 Ktesibios' water clock. 16
1.5 Heron's weight regulator. 18
1.6 Graphs of interactions between quantities traced by explanations of how ancient fluid devices work. 20
1.7 Understanding how a device works requires making the interaction topology explicit. 22
1.8 Example of library design applied to constructing an inverter. 24
1.9 An account for the invention of a device similar to Philon's lamp according to interaction-based invention. 31
1.10 Principles required for different stages of interaction-based invention. 33
1.11 Examples of innovation from first principles alone. 36
1.12 Examples of evolutionary design through successive innovations. . . . 37
1.13 Tracing a path through the topology of potential interactions. 39
1.14 Distinct paths correspond to innovative alternatives. 40
1.15 Evolutionary progression of MOS memory buffers with modifications indicated. 42
1.16 Each step in the evolution involves an innovation, consisting of a change to the interaction topology. 44
1.17 Major components of Interaction-based Invention. 51

2.1 Initial setup for the punch bowl problem. 54
2.2 Using a pipe to restore the punch bowl level. 56
2.3 A state transition representing fluid restoration. 61
2.4 Design transformations between interactions and to physical structure. 65
2.5 Analogy between FSM design and transformations of interaction-based invention. 69
2.6 Integration Rule 73
2.7 Desired dynamic interactions for punch bowl example 74
2.8 Mapping a logical expression to its corresponding digital circuit. . . . 80
2.9 A graph of interactions for the pipe solution to the punch bowl problem. 81
2.10 Fluid Container Interactions 86
2.11 MOS Inverter 88

3.1 Examples of components and terminals. 100
3.2 Example of shared-node connection between two components. 101

3.3 Topology of interactions for the vat highlighting what interacts. . . . 103
3.4 Primitive topologies for simplified models of fluids. 105
3.5 Topology of interactions for simplified laws of fluids. 106
3.6 Topology of existing interactions for initial specification of punch bowl problem. 108
3.7 Paths through the topology of existing interactions for the punch bowl problem. 112
3.8 Identify access paths and access variables for desired interaction. . . . 113
3.9 Build new interactions between access variables. 114
3.10 New interactions produced by connecting a pipe. 115
3.11 Topology of potential interactions for the simplified fluid models and laws. 118
3.12 Interrelating existing and potential interactions. 123
3.13 Identify needed interactions by tracing paths along existing and potential interactions. 126
3.14 Making augmentations to the device's topology of existing interactions, by augmenting its physical structure. 127
3.15 Identify places to inject new interactions for the punch bowl problem. 133
3.16 Identify augmentations to the interaction topology for the punch bowl problem. 135
3.17 A topology which is a coarse solution to the punch bowl problem. . . 137
3.18 Making augmentations to a device's topology of existing interactions, by augmenting its physical structure. 138
3.19 Topologies and physical structure for punch bowl problem before making augmentations. 139
3.20 Introduce a single interaction to the topology of existing interactions 141
3.21 Introduce the remaining potential interactions into the topology of existing interactions. 142
3.22 Introduce a single link to the topology of existing interactions 144
3.23 Introduce the remaining links interactions into the topology of existing interactions. 146
3.24 Proposed augmentations to physical structure to solve the punch bowl problem 148
3.25 Instantiation of proposed interaction topology for relating variables of the desired interaction. 149
3.26 Accumulated expressions during propagation 152
3.27 Sequence of interactions combined to verify that the interactions along the path produce the desired interaction. 153
3.28 A path for the punch bowl problem, resulting in an interaction topology that can't be built. 155
3.29 Proposed paths sometimes leads to inconsistent structures. 157
3.30 The remaining proposed structure after removing an inconsistent link. 158
3.31 Instantiation of proposed interaction topology involving an additional container and pipe. 160
3.32 Proposed augmentations to physical structure using a closed container. 161
3.33 Topologies of interaction for models and laws of fluids. 165
3.34 Additional topologies of interaction for various fluid container models. 166

3.35 Topologies of interaction for models and laws of mechanics. 167
3.36 Topology of potential interactions for the hydro-mechanical models and laws. 169
3.37 Topology of existing interactions, using hydro-mechanical principles, for the initial specification of punch bowl problem. 170
3.38 Identifying places to inject new interactions for the punch bowl problem using hydraulic and mechanical properties. 171
3.39 Hydro-mechanical Design 1 172
3.40 Augmentations to interaction topology for hydro-mechanical design 1. 173
3.41 Hydro-mechanical design 2 174
3.42 Augmentations to interaction topology for hydro-mechanical design 2. 175
3.43 Hydro-mechanical design 3 177

5.1 Definitions for sign addition and subtraction used in confluences. . . . 209
5.2 Definitions for sign multiplication ($\otimes$) and division ($\oslash$) used in $Q1$. . 212
5.3 Expressions in $Q1$ become successively more abstract as real operators are replaced with sign operators. 214
5.4 Replacing an expression's real operators with sign operators increases the opportunity for ambiguity to arise. 215
5.5 Demonstration that clean boundaries are lost through over-abstraction when using only a sign algebra. 219
5.6 A mosfet's state space, defined by $V_{G,S}$ and $V_{D,S}$, is partitioned into three distinct regions of operation. 220

6.1 Simplification rules used to reduce quadratic sign polynomials to a canonical form. 243

8.1 Simple definitions for hydraulics 278

Chapter 1

Introduction

"Discoveries and inventions arise from
the observation of little things"
– Alexander Graham Bell

One hallmark of intelligence is the ability to create. In this thesis we explore one aspect of creativity – the process of invention. A central question we address is:

> How can an inventor use his understanding of basic physical principles to develop innovative devices?

This thesis presents a theory that captures several key aspects of the design process used to construct innovative devices, such as the ancient oil lamp shown in Figure 1.1. Called *Interaction-based Invention*, it is based on the following insights:[1]

[1]Throughout this thesis we use the term "invention" to highlight our specific focus within design on the process of innovation from basic physical principles, and use "inventor" to refer to the expert designer who is particularly skillful at constructing novel devices.

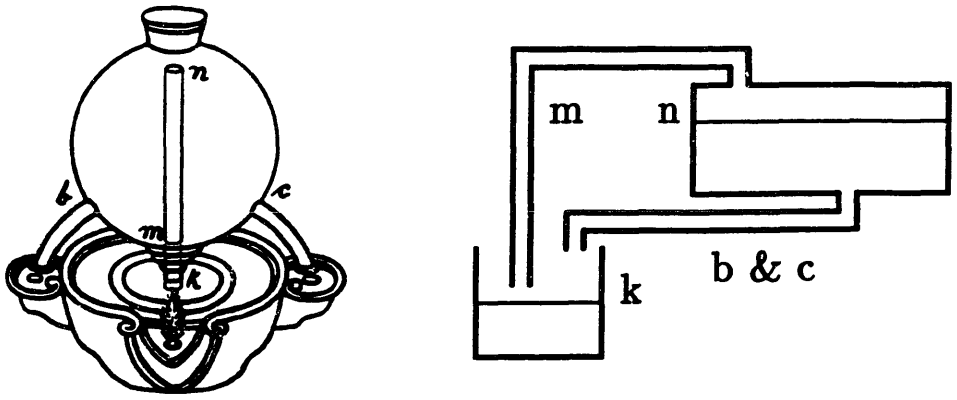


Figure 1.1: Philon's oil lamp. The lower container is the lamp, which holds a floating wick. The upper container holds a supply of oil, used to restore the oil level in the lower container.

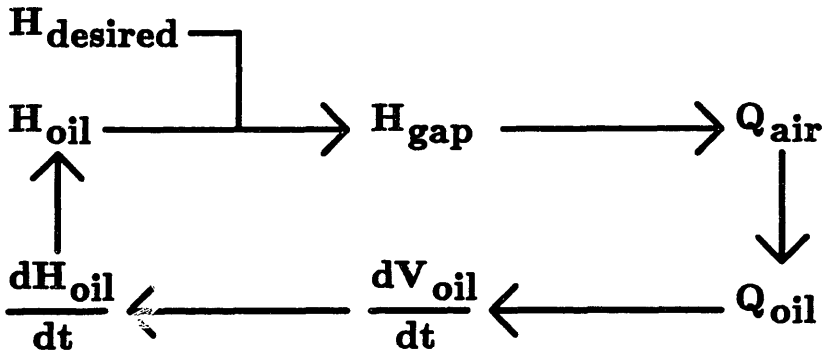


Figure 1.2: Philon's oil lamp is innovative in the way it works. The sequence of interactions constructed by the lamp to produce the desired behavior is the first sequence to explicitly construct a feedback loop. These aspects of the innovation are conveyed by the graph shown above.

- One way a device is innovative is if it works in a way fundamentally different from those seen before.

- In order to identify fundamentally different ways of producing a desired behavior, the inventor focuses on the time-varying topologies of qualitative interactions between quantities (Figure 1.2).

- To develop novel devices, invention involves constructing a topology of interactions that both produces the desired behavior and makes evident a topology of physical devices that implements those interactions (Figure 1.3).

Invention involves much more than applying accumulated design experience to routine problems. To develop a robust theory of invention – one that captures an inventor's ability to exploit fundamentally new technologies and to handle extraordinary design situations – we explore the *art of invention from a historical perspective.* In ancient times innovative devices were developed from a basic knowledge of physics. At that time little else was known; a body of design experience had not yet been accumulated. We claim that this process of constructing innovative devices from fundamental principles of physics is also a crucial component of modern invention. Thus an important way to understand how modern inventions are developed is to first understand how innovative devices have arisen in the past. To this end our theory of interaction-based invention is based on insights derived from case studies of historically significant devices, invented in ancient (300 B. C.) and recent times (1970-80). These insights have allowed us to capture key aspects of the process by which *innovative devices first emerge from basic observations of physics, and then evolve over time into sophisticated high performance devices.* Before sketching this approach it is instructive first to consider examples of the types of devices our theory can produce.

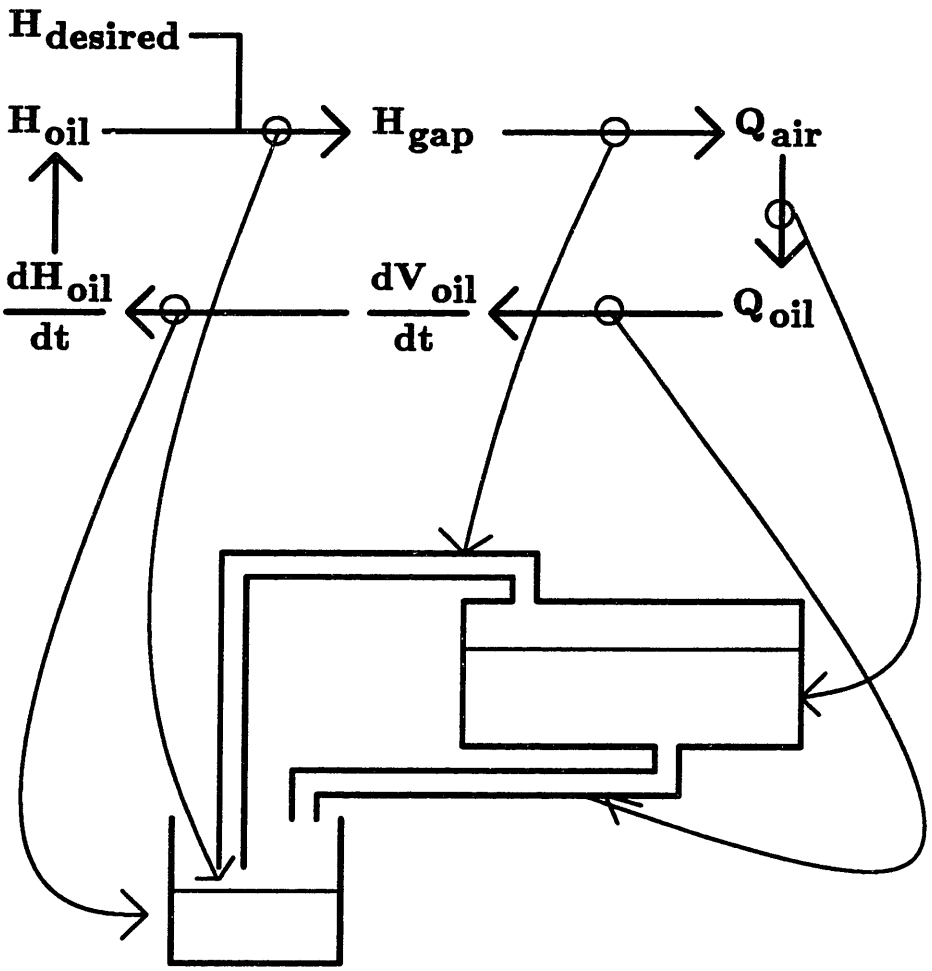


Figure 1.3: Interaction-based Invention applied to Philon's lamp. This involves constructing a topology of interactions that produces the desired behavior of restoring oil level, and makes evident its corresponding physical structure.

1.1 What Makes a Device Innovative

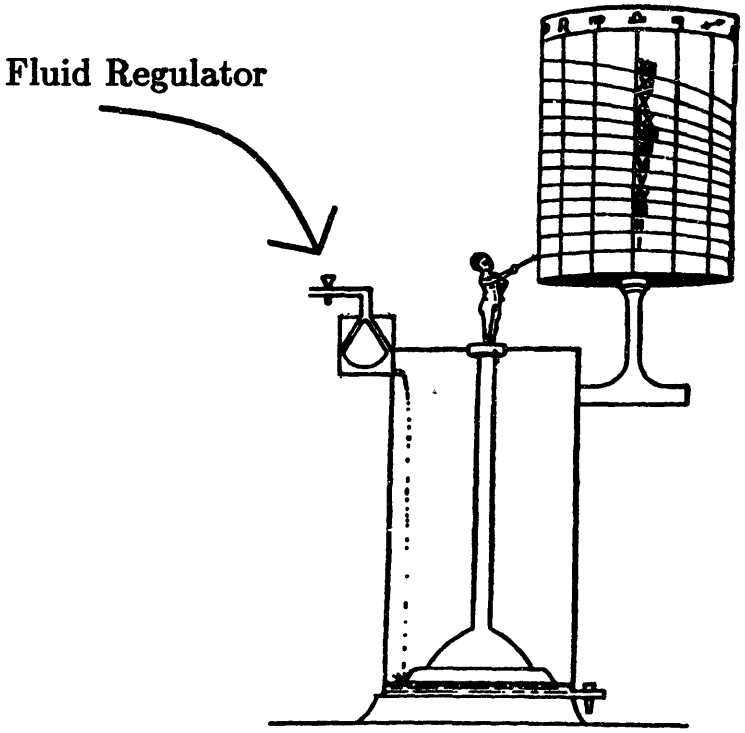


Figure 1.4: Water clock with fluid regulator, developed by Ktesibios of Alexandria in the third century B. C.

Consider the problem, faced by ancient inventors, of building a water clock (Figure 1.4).[2] The key to developing a water clock is to produce a steady flow of water that can be accumulated to measure time; this is called the fluid regulation problem. A common way to accomplish this is to put a hole at the basis of a container. The flow out of this hole is then made constant by maintaining a constant height of water in the container. Inventors of the era knew about many of the properties of fluids (due in part to Archimedes), as well as simple devices such as vessels, pipes, valves, floats,

[2]The examples of feedback control used in this section are taken from "The Origins of Feedback Control" by Otto Mayr[25]. This book has provided a valuable source of inspiration in exploring the evolutionary design process.

scales and linkages. The problem then becomes, how can we maintain a constant height of water in a container using these simple devices and principles?

Before continuing to the next page, you may wish to take a few minutes coming up with your own solution to the problem.

The Egyptians and Arabs came up with many solutions to this and similar problems. We will consider two of the most innovative designs, both based on the hint on the previous page.

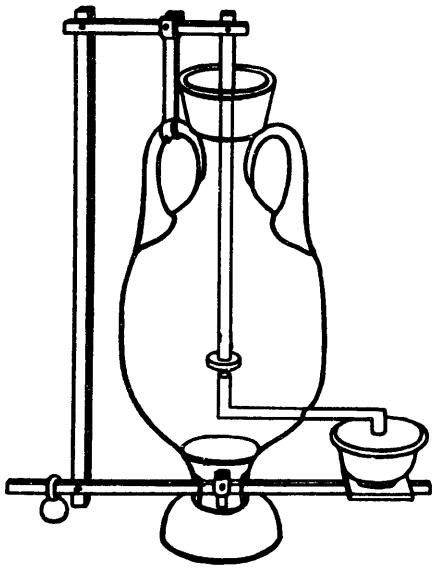


Figure 1.5: Heron's weight regulator.

Figure 1.5 shows a device that maintains a constant fluid height by weighing the contents of the container, developed by Heron of Alexandria in the first century A. D.. The device consists of a a small container whose height of fluid is being regulated, a large container that supplies fluid, together with a scale, weight, linkage, valve and pipe that regulate the fluid level. The device maintains a constant fluid level as follows: If the fluid level in the small container is too low, then the small container will be too light, and the weight at the end of the scale will move downward. Through a set of linkages this causes the valve inside the large container to open, allowing fluid to flow into the small container. This causes the container's height of fluid and weight to increase. Eventually the fluid reaches the desired height, at which point the scale tips, causing the valve to close.

The device, shown earlier in Figure 1.1, is by Philon of Byzantium, and is a clever means of maintaining a constant height of oil in a lamp. Although its purpose is different from the previous device, the basic problem is similar – maintaining the height of a fluid above a specified level as fluid is lost. The lamp consists of an open container of oil with a floating wick, plus a closed container and a series of pipes that store and supply oil as it is consumed.

Assume the closed container n is full of oil, and the height of oil in the open container is sufficient to cover the opening of the vertical pipe m. When the flame consumes enough oil to expose the lower end of the pipe m, air flows through the pipe, letting oil flow through the capillary tubes b and c, refilling the open container. When the oil rises enough to cover the lower end of m, air cannot enter n to replace oil leaving, thus the outflow through b and c stops.

An important component of what makes these devices innovative is that they work in a way that is qualitatively different from preceding inventions of the time. But what do we mean by working qualitatively differently? In fact what do we mean by how something works?

One of the major perspectives taken in this thesis is the notion that *how a device "works" is captured by the sequence of interactions between variables that achieve the desired behavior.* These are interactions traced in the explanation above of each device's behavior, and are depicted graphically in Figure 1.6.

The devices are "qualitatively different" in the sense that the paths traced by the interactions are different from their predecessors (i.e., the *topology of interactions* is unique). Both devices are considered innovative for their time because they use the difference between existing and desired fluid height to control the rate that fluid flows into the container. The sequence of comparison followed by adjustment (i.e., feedback control) had not appeared in earlier designs. Graphically, they are the first devices whose interaction topologies contain a loop.

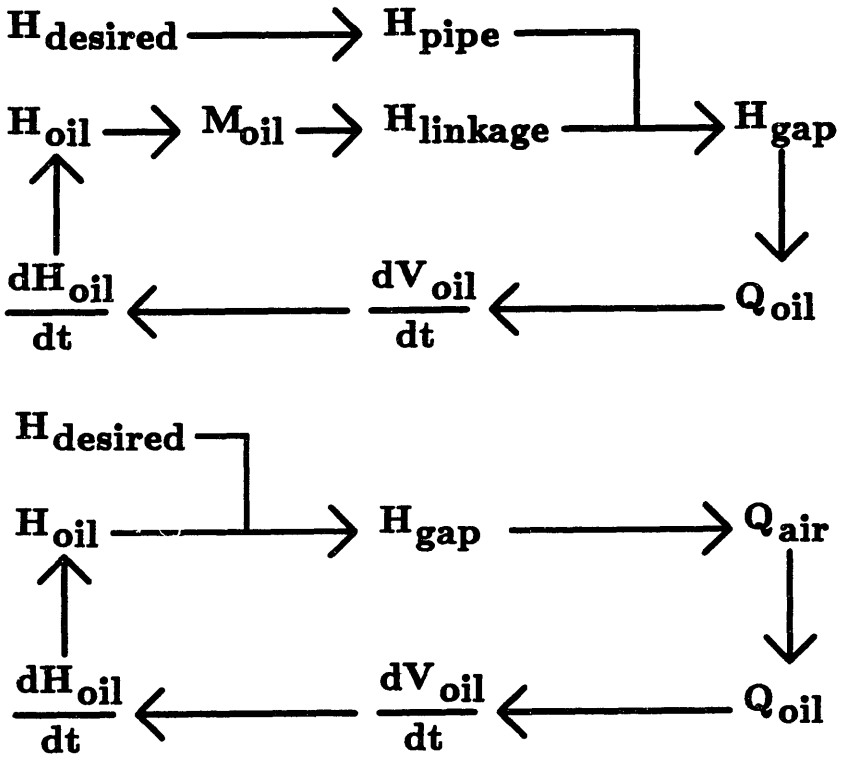


Figure 1.6: Graphs of interactions between quantities traced by explanations of how ancient fluid devices work. The upper graph is that for Heron's weight regulator and the lower is that for Philon's lamp. The topology of these interactions captures qualitative differences in how the two devices work. Although both graphs contain a loop, the sequence of variables related are different.

Note that these graphs provide only part of the picture – they highlight only what variables interact. A topology of interaction also includes a qualitative description of *how* the variables interact. In addition, these graphs are simplified versions of the interaction topologies required to understand the device's behaviors in detail. Chapter 3 provides examples of the topologies actually used.

The particular path of interactions used to relate height and rate of inflow is also innovative in each case. Heron measures the fluid height indirectly through container weight, which is used to control the rate of water inflow through a mechanical linkage. For Philon's lamp the innovative component of the path is the interactions used to control flow of oil into the lamp. Preceding designs controlled this flow directly through a valve. Philon's lamp controls this flow indirectly – using oil height to control air flow into the closed oil container, which in turn controls the flow of oil into the lamp by exploiting conservation of flow for a closed container. Thus the devices are distinguished in that their topology of interactions are different.

Using only the figures of physical structure (Figures 1.5 and 1.1), it takes a while to understand how each device works. Part of the problem is that the physical structure of a device often does not make apparent the complex set of interactions it produces. For example, a single container of fluid produces a set of interactions interrelating pressures, fluid flows, fluid heights, fluid density and gravity. Thus to understand the devices above we must first make the set of interactions explicit (Figure 1.7). Also because the set of interactions produced by each component of a device is often complex, many interactions may be produced that are extraneous or parasitic to the desired behavior of the overall device. We need to focus on the interactions relevant to producing the desired behavior (Figure 1.7, highlighted interactions). In more complex devices an interaction may play several roles during different periods of a device's behavior. We need to separate out these different roles.

It is much easier to understand what is innovative about the devices above after reading the explanations of how each works, *because the explanations make the topology of relevant interactions more explicit.* The explanations do not need to use quantitative information; a qualitative description of interactions is often sufficient to understand a device's operation, and suppresses substantial irrelevant details. A representation for interactions must capture this qualitative information at the right level of abstraction.

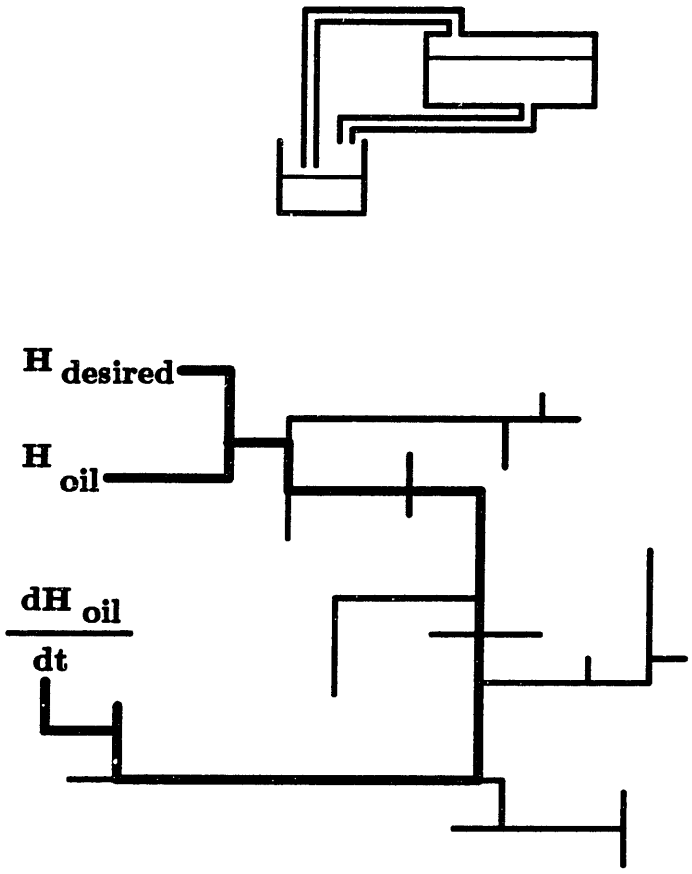


Figure 1.7: To understand how a device works we must a) make explicit the topology of interactions, and b) identify those interactions that impact the device's desired behavior (the highlighted interactions).

A central focus of this thesis is the concept of the topology of interactions and its role in highlighting significant differences in how a device works. In this section we have seen that the topology of interactions plays an important role in identifying some types of design innovations. In Section 1.5 we see that the topology of interactions is a key to constructing innovative devices.

1.2 Why Invention?

Before presenting our approach consider why a theory of invention is important, and how close we are to developing it. We claim that such a theory is central both to understanding artificial intelligence (AI), and to automating the process of design. First, with respect to AI, invention strikes at the heart of two hallmarks of intelligence – tool building and creativity. The former is the ability, not only to account for physical phenomena, but to put these phenomena to use by constructing tools. In our case the phenomena are described as basic laws of physics and the tools constructed are hydro-mechanical or electrical devices. The latter – creativity – is the ability to construct solutions to a task that are qualitatively different from those seen before. In our case the task is design, the solutions are physical devices, and the qualitative differences are a type of design innovation where devices work in fundamentally new ways – their topologies of interactions are distinct.

In addition, our theory of invention draws upon and makes substantial contributions to many areas of reasoning that are at the center of AI research. These include, but are not limited to, qualitative, causal, temporal, algebraic and "constructive" reasoning.[3] The specific contributions made to each area are highlighted later in this chapter.

[3]Constructive reasoning is a type of terminological reasoning we have developed for building interaction topologies. Terminological reasoning involves reasoning from the definitions of concepts, and is the focus of the knowledge representation community[7].

We also claim that capturing the process of innovation is at the heart of developing a robust theory of design. Furthermore, that existing design approaches are inadequate exactly because they can't innovate. To see this consider the state of the art in design research.

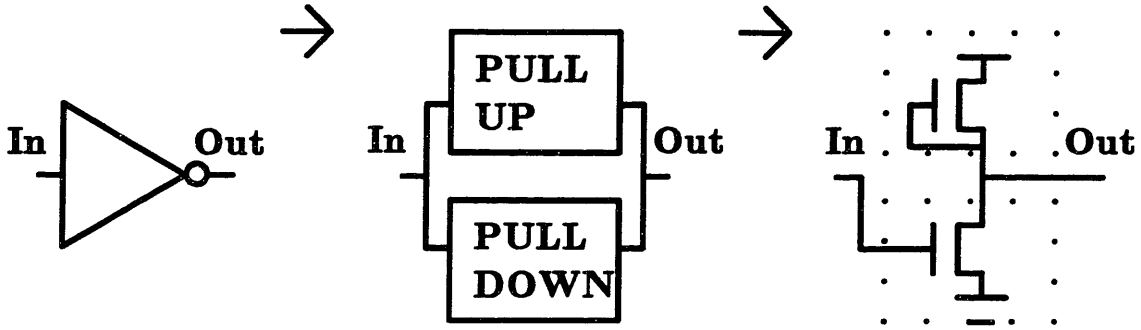


Figure 1.8: Example of library design applied to constructing an inverter.

The predominant approach explored in the AI design community[26,36,30,29,31, 33,39] and the computer-aided design community[19,44,24] is library design. These appear under labels such as routine design, hierarchical refinement, or silicon compilation. The basic idea is, given a desired behavior, a design schema is looked up in the library that matches this behavior. For example, suppose we want to design an inverter (left side of Figure 1.8). Then the library is consulted, for example, coming up with a schema which uses a pullup and a pulldown (middle of Figure 1.8). This schema is then refined by successively instantiating schemas from the library, until a design is constructed that consists only of primitive components (right side of Figure 1.8).

Consider the question – how close are we to a theory of invention? Library design systems are users, but not *producers* of innovations. Specifically, nowhere have they been embodied with the ability to use fundamental principles of physics, or to focus the application of these principles on constructing innovative devices. While it is conceivable that these systems could stumble across a novel design, they have not

been embodied with the abilities that allow skilled designers to consistently come up with innovations. Thus, very little progress has been made in this earlier work towards a theory of invention.

Consider, with respect to design, the question: why is a theory of invention important? Existing automated design systems have not demonstrated the capability to construct devices that parallel the average designer for standard design problems. Often they fail to meet the specification for these problems, typically because they can't meet the minimum performance constraints. For example, the design system fails to design an inverter whose output rises and falls fast enough.

From a practical standpoint, this inadequacy cannot be overcome by simply augmenting the design libraries with innovations developed by humans. Because of the continual demand for higher performance, and rapid technology shifts, we would continually be playing catchup. Instead the design systems must be able to innovate, by exploiting new technologies and existing technologies in novel ways. An important way to accomplish this is to reason from the fundamental principles of physics that characterize these technologies. Thus, for technologies with a rapid pulse rate, *being a competent designer means being able to innovate.*

1.3 The Task Explored

In addition to the types of innovation considered (i.e., qualitative differences in how a device works), the boundaries of our theory of invention are defined by the representations used for a device's behavior and physical structure. Many innovations arise at the early stages of the design process, where the designer is faced with an enormous space of possibilities. One way the designer simplifies the process early on is by considering only the gross structure and behavior of the device. We take a similar approach in our theory of invention.

To model gross structure, we adopt the lumped-element model of circuit theory[4] and physical system dynamics[41,38] – a standard representation that is often used by engineers during the early stages of design. In this model a device's structure is represented as a network of components and connections. This representation is sufficient to capture the innovations highlighted for the ancient fluid devices in Section 1.1, while it substantially simplifies the design task by suppressing detailed spatial features.

In addition circuit theory and physical system dynamics provide general theories of physics based upon the lumped element model, that apply across domains and technologies (e.g., electrical, hydraulic, mechanical and thermal systems). Our theory of invention is able to achieve a high degree of generality by adopting the lumped element model and drawing upon the basic physical principles of these theories.

It is important to note that some types of innovation have to do with the detailed spatial structure of a device, and cannot be captured by the lumped element model. For example, an additional aspect of Philon's lamp that makes it innovative, is that the surface of the oil is used to close off the mouth of a pipe, thus acting like a valve (Figure 1.1). This innovation could be encapsulated into a lumped element, and then we could incorporate it into a design, but we could not invent the element.

The remaining constraint on our approach is the representation for gross behavior. We adopt the approach used in qualitative physics, of breaking the state space of a device (i.e., the space described by a device's state variables) into a set of regions, and describing the device's behavior in terms of quantity shifts between these regions (e.g., a fluid level shifting from too low to the desired level). This representation can be used to describe the behavior of a wide variety of devices, such as the ancient fluid devices introduced earlier, and electrical circuits used for digital applications.

In addition, although not currently pursued, in theory our approach can also handle the design of a broad range of devices, such as analog amplifiers, whose behavior

can be specified by qualitative or quantitative descriptions of non-linear, differential equations.[4] However, our approach cannot handle the design of devices whose behavior must be described by precise, time-varying functions, such as sine generators and radio transmitters. Given these restrictions on structural and behavioral representations, we are ready to sketch the major elements of our theory of invention.

1.4 Elements of the Invention Process

How do we account for inventions like Heron's weight regulator or Philon's lamp (Section 1.1)? To be innovative an inventor must certainly draw upon a solid understanding of fundamental principles. To emphasize their importance, this type of design is sometimes referred to as *design from first principles.* An important objective of an engineering education is to teach the student a broad set of first principles in mathematics and physics. Mathematics provides a set of domain independent principles for characterizing behavior, while physics provides the means of establishing the link between these mathematical concepts and properties of a particular technology being explored. Together the principles of mathematics and physics provide an engineer a foundation for analyzing the behaviors of new technologies in a broad set of design situations, as well as to study the basic phenomena underlying a familiar technology in order to identify new solutions.

But this foundation is not enough. A naive designer quickly becomes overwhelmed by the number of possibilities that result from the generality of the basic principles. Instead the designer must use every means available to carefully *focus* the application of the first principles at each stage of the design process. Thus, an inventor, one who frequently constructs novel devices to meet demanding situations, is distinguished

[4]More precisely, the desired behavior of a device can be specified as any equation in the hybrid qualitative/quantitative algebra $Q1$, discussed in Chapter 5.

not by the fundamental principles he knows, but by the way he uses them and how he guides the search for a new device in an overwhelming space of possibilities.

To avoid being overwhelmed the inventor must use every element at his disposal to focus the search for innovative devices. We know from AI that three basic elements of problem solving are the *representations, basic operations, and search strategies* used. The possibility for becoming lost is so great that, to be successful, everyone of these elements must optimize the inventor's ability to focus.

AI has developed a set of well known insights that provides some guidelines for selecting appropriate representations, operations and search strategies for invention. Some examples are:

- A representation should make explicit qualitative differences between solutions.
- A representation should be as abstract as possible, while capturing the salient features of a solution.
- The basic operations performed by a problem solver should allow it to take the biggest steps possible through the search space, without sacrificing generality.
- A solution can be arrived at quickly by constructing a coarse solution in an approximate search space and then refining it.
- An incorrect solution can be refined by pinpointing the cause of failure and debugging it.

Each of these insights provides an important starting point. To date, however, little has been known, for example, about what these qualitative differences, abstractions, operations, approximate spaces or debugging techniques should be for a particular category of problem solving task.

We believe that our concept of a topology of interactions provides a key, not only for recognizing innovations – but for focusing the process of invention, by highlighting aspects of a device that are significant to how it works. The interaction topology provides a focus for identifying elements, such as abstractions, operations, approximations, and debugging techniques, that are appropriate for the process of invention. These are embodied in a set of design strategies, unified into a coherent theory of invention by the underlying theme of operations on interaction topologies. The remainder of this chapter highlights the major contributions of our theory in terms of the first principles, strategies, representations, and operations it contains.

1.5 Interaction-based Invention

We claim that the topology of interactions is a central focus of invention. Consider the justification for this claim and its impact on how we perceive invention. First, our claim is based on the observation that focusing the search for innovative devices requires a representation that highlights aspects of devices that make them innovative. That is, we know from recent work [22] that the success or failure of the discovery process at identifying "interesting" new concepts is determined in large part by the representation used. The right representation must highlight significant differences while suppressing all other detail. For invention the representation must highlight differences that make a device innovative.

We have already observed that one type of innovation is fundamental differences between devices in terms of how they work. A device "works" by constructing a set of interactions between quantities and then modulating these interactions over time. Thus to identify fundamentally different ways of producing a desired behavior, we focus on the time-varying topologies of qualitative interactions between quantities. The representation we have developed for this purpose is called an *Interaction Topology.*

Next consider the impact of the interaction topology on our perspective of invention. The approach of using interaction topologies to guide the search for novel devices we call *Interaction-based Invention.* This approach is based on the viewpoint that:

> The process of invention involves constructing a topology of interactions that both produces the desired behavior and makes evident a topology of physical devices that implements those interactions (Figure 1.3).

For more complex tasks, the design process involves transformations between different types of interaction topologies as well as transformations between interaction topologies and physical structure.

To test our theory we have implemented a program, called *IBIS* (for Interaction-Based Invention System) which has been demonstrated on the design of simple electrical and hydro-mechanical devices, such as devices similar to the ancient fluid devices shown earlier.

The major stages of interaction-based invention are highlighted in Figure 1.9. The design system IBIS starts with a description of the desired behavior, represented as a state diagram, where each arc describes a quantity shifting from one value or interval to another. The state diagram at the top of Figure 1.9 describes the behavior of Philon's lamp, that is "raise the height of fluid in the lamp to the desired level and hold it there."

First IBIS turns this diagram into a set of one or more desired interactions that produce the specified quantity shifts. These interactions are described by equations in a new hybrid qualitative/quantitative algebra, called $Q1$. For this example the desired interaction is "change the height of fluid in the lamp in the same direction as the difference between the desired level and the current height."

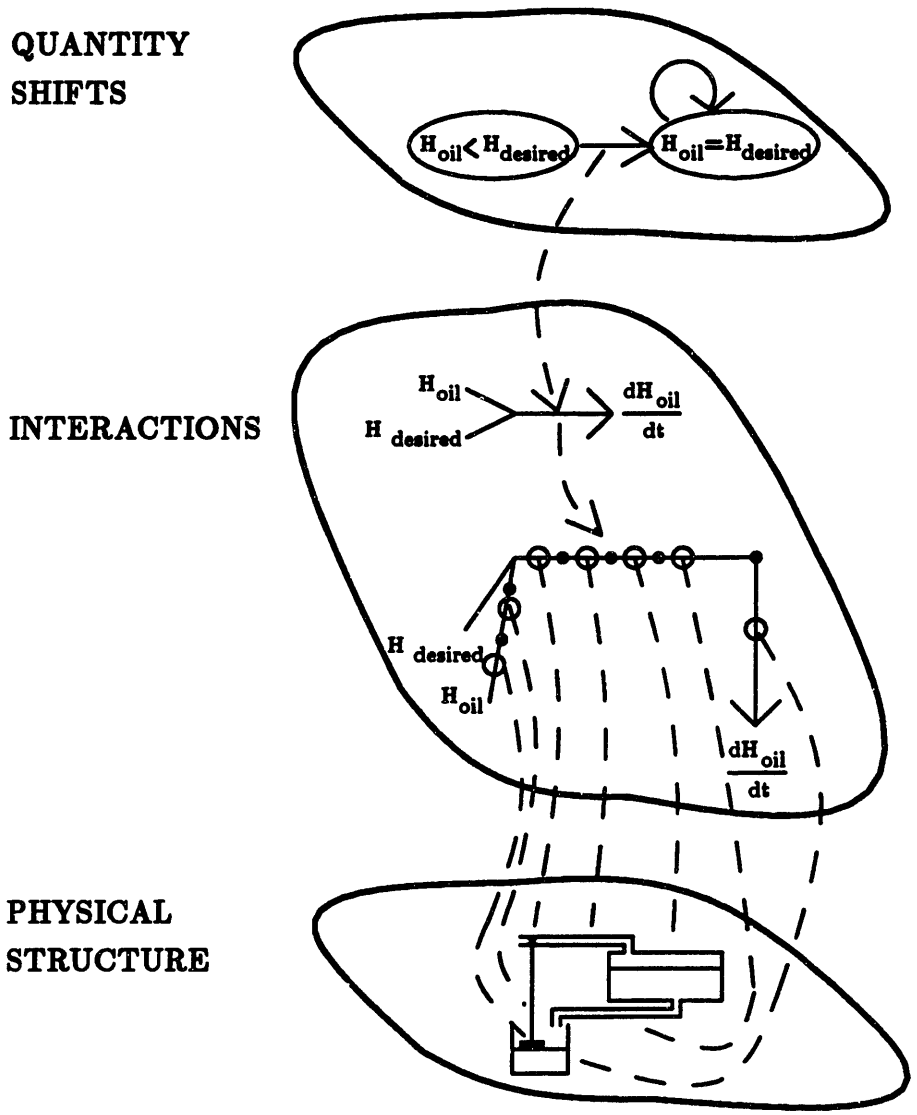


Figure 1.9: An account for the invention of a device similar to Philon's lamp according to interaction-based invention.

IBIS then constructs a detailed topology of interactions that produces the desired interaction and makes evident augmentations to device structure. By "make evident" we mean that each interaction in this topology is producible by a single connection or component available in the domain and technology being explored. The link between interactions and components or connections are provided by the basic laws of physics. The end result of the invention process is the detailed topology of interactions and the physical structure that implements it, in this case Philon's lamp.

1.6 First Principles of Interaction-based Invention

Recall that our goal is to develop a robust theory, one that can design a wide variety of innovative devices. In Section 1.4 we argued that to accomplish this our approach should be based on a core set of principles from physics and mathematics (the *first principles*). To get a feel for the scope of our approach, consider the principles used by IBIS to achieve the desired generality.

Engineers and mathematicians have developed a set of theories (e.g., calculus, algebra and circuit theory) for operating on and mapping between representations that are quantitative analogs of those used in interaction-based invention. These theories provide the starting point for developing the set of first principles. Consider what theories are relevant. The quantitative analog of quantity shifts are time-varying functions, and the analog of interaction topologies are equations. Using this analogy, the process of design goes from time-varying functions to physical structure. The relationship between time-varying functions and equations is captured by calculus, the relationship between different equations is provided by algebra, and the relationship between equations and physical structure (described as networks of components) is provided by circuit theory and physical system dynamics.

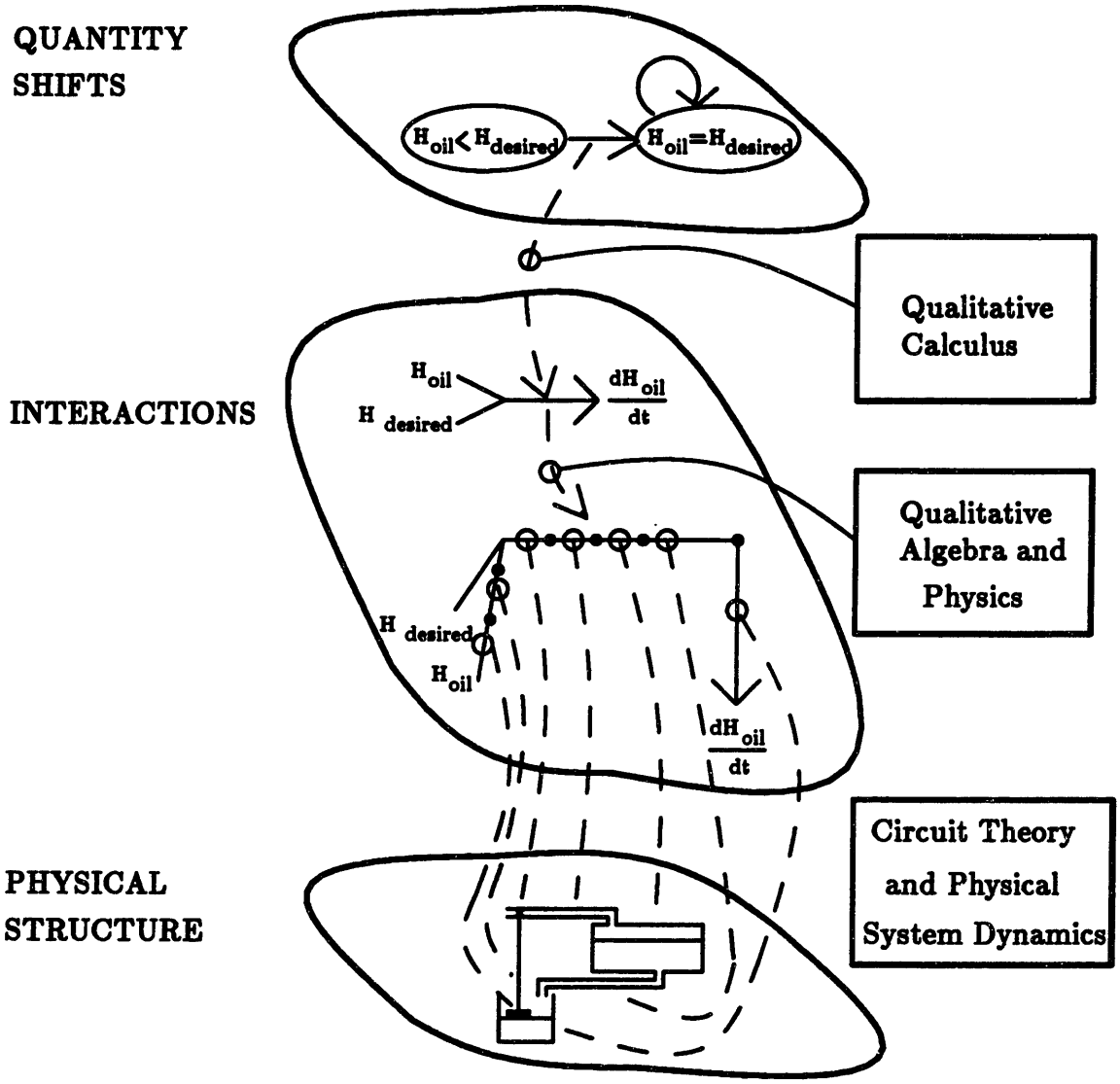


Figure 1.10: Principles required for different stages of interaction-based invention.

The exact set of principles required depends on the level of abstraction at which the behavior is examined. Many of the decisions that shape the structure of a device are made at the early stages of the design process. At this stage very little quantitative information is known and the decisions a designer makes are about classes of devices whose common behavior need only be described qualitatively. Thus the set of principles used are a qualitative abstraction of principles taken from physics (i.e., circuit theory or device physics) and mathematics (i.e., algebra and calculus).

Consider the qualitative principles required; these are summarized in Figure 1.10. Recall from the last section that the first step of interaction-based invention goes from quantity shifts, described using a state diagram, to a set of desired interactions, described by equations in $Q1$. Mappings between these representations are performed using principles for integrating rates of change, derived from our earlier work on the mathematical consequences of continuity on qualitative simulation[65]. The effect of rates of change on a quantity is determined using fundamental theorems of calculus, in particular the integration and continuity rules, which are corollaries of the mean value theorem, intermediate value theorem and the definition of continuity[65].

The second step constructs the topology of interactions that produces the desired interaction and makes evident the physical structure. Showing that the composite behavior of the interaction topology produces the desired interactions requires performing symbolic algebra on $Q1$ equations. The requisite principles are axioms for $Q1$[62] (Sections 6.2 and 6.3).

Finally, consider the principles used to construct the interaction topology and corresponding physical structure. The topology is built from interactions produced through components and connections. These interactions and the structure they impose are identified using qualitative or quantitative versions of the laws and models of circuit theory[4] and physical system dynamics[41]. The laws describe the interactions produced by connections and the models describe the interactions produced by

each type of component.

The topology is constructed by connecting together these interactions through shared variables. Taxonomic information about classes of variables (e.g., pressure, flow) and structural objects (e.g., components, terminals) are used to determine what variables can be shared, and what physical constraints are required to make them shared.

Together the principles of calculus, algebra and physics used at the different stages provide the foundation necessary to satisfy robustness. They enable IBIS to exploit new and existing technologies in novel ways, and have been used to construct both hydro-mechanical and electronic devices.

Section 1.4 argued that, while having the first principles are necessary to achieve robustness, what distinguishes an inventor is the way he uses them and how he guides the search for a new device in an overwhelming space of possibilities. This is the topic of the remaining sections.

1.7 Strategies for Invention

Our goal is a theory that embodies the broad spectrum of invention – a process that begins with a knowledge of first principles alone, and results in a set of successively more sophisticated devices. To capture the design strategies involved we have explored two case studies. First, to understand how simple devices arises from first principles, we go back to a time when little else was known but these principles. The case study is a set of ancient fluid devices that restore the level of fluid in a container (Figure 1.11). Each of these devices are considered innovative for their time, and played a significant role in developing the theory of modern feedback control.

Second, to understand how these simple devices evolve to sophisticated designs

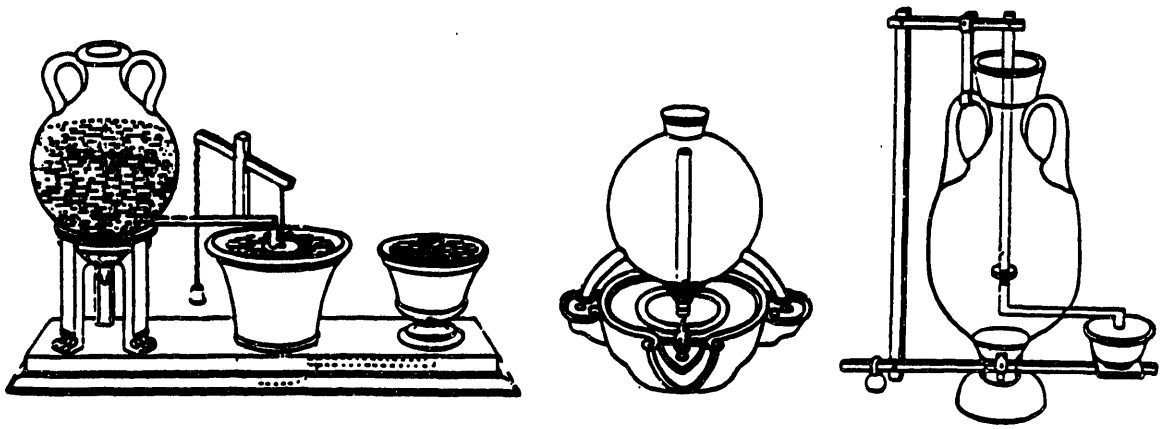


Figure 1.11: Innovative solutions to the fluid regulation problem, developed around 300 B.C. – a time when little past design experience was available.

we study a historical progression. The case study, shown in Figure 1.12, consists of a sequence of successively more complex and higher performance MOS memory buffers. These buffers were developed around 1970-80, and represent leading edge designs of their time.

Based on an informal analysis of these case studies we have identified three design strategies, which we call *tracing topologies of potential interactions*, *design from second principles*, and *evolution through focussed innovation.* These strategies provide complementary ways of using the interaction topology to guide the process of invention at different levels of granularity. The first is our theory of how innovative devices are derived directly from first principles alone. The second and third strategies are ways of selectively using the first strategy when innovation is required.

The first strategy is the only one discussed in detail in this document. The other two strategies are currently under development. We present an overview of all three, to help the reader to understand our larger perspective on invention, to see the role of design from first principles in this context, and to highlight our current progress towards this larger goal.

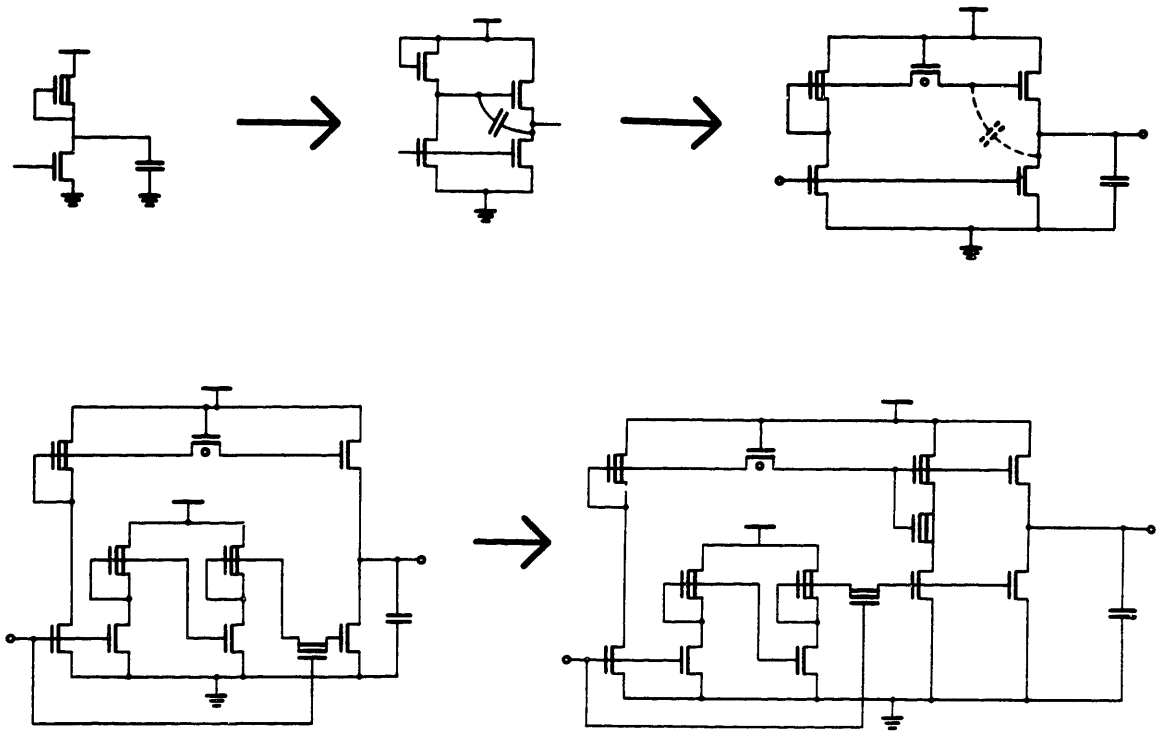


Figure 1.12: A sequence of high performance MOS memory buffer designs provides an example of evolutionary design through successive innovations.

Tracing Topologies of Potential Interactions

The first strategy proposes a way that simple devices emerge from first principles by using the interaction topology to focus on innovative alternatives. The conjecture is that ***new innovative devices are constructed in a manner similar to how existing inventions are understood.***

Section 1.1 observed that, to understand how a device works, we first visualize all the interactions produced by every component and every connection in the device, and then identify those interactions that contribute to the desired behavior (Figure 1.7). Analogously, to invent a device, we use our first principles to visualize *all* possible interactions producible by each type of component and each type of connection available in the current technology, and *all ways* that these interactions can be connected. The result is a graph, which we call a *topology of potential interactions.* Constructing a new device involves identifying those interactions in this topology that produce a desired interaction. These interactions trace a path through the topology between variables in the desired interaction. For example, Figure 1.13 shows a topology of potential interactions for hydro-mechanical components available in ancient times, together with a path relating fluid height change to height difference. The interactions along the path correspond to those produced by Philon's lamp. The fact that a topology of potential interactions can be constructed, let alone searched, is not obvious. A major objective of this thesis (Chapter 3) is to show how this is accomplished.

Alternative solutions are proposed by identifying different paths through the potential interaction topology. These paths result in devices with distinct interaction topologies, and thus correspond to innovative alternatives. For example, Figure 1.14 shows paths for three solutions that are explored in this thesis. Each corresponds roughly to a novel fluid regulation device developed in ancient times.

The fact that the topology of potential interactions embodies the first principles

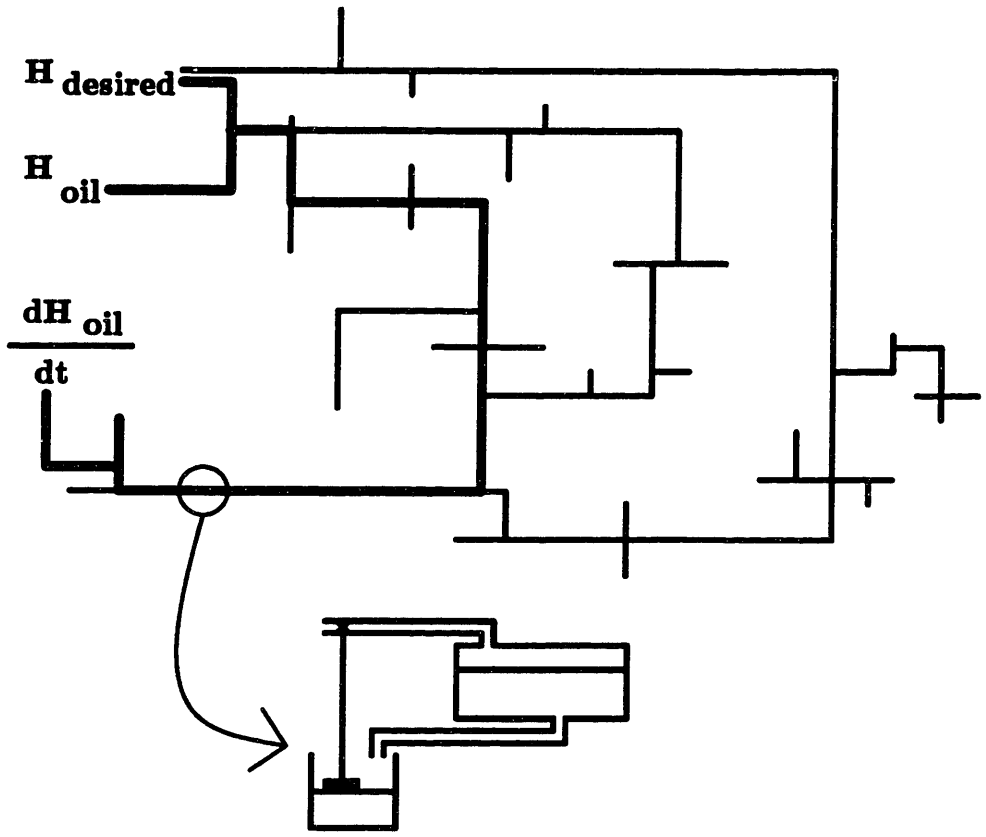


Figure 1.13: The topology of interactions for a new device is identified by tracing a path through the topology of potential interactions for the available technology.

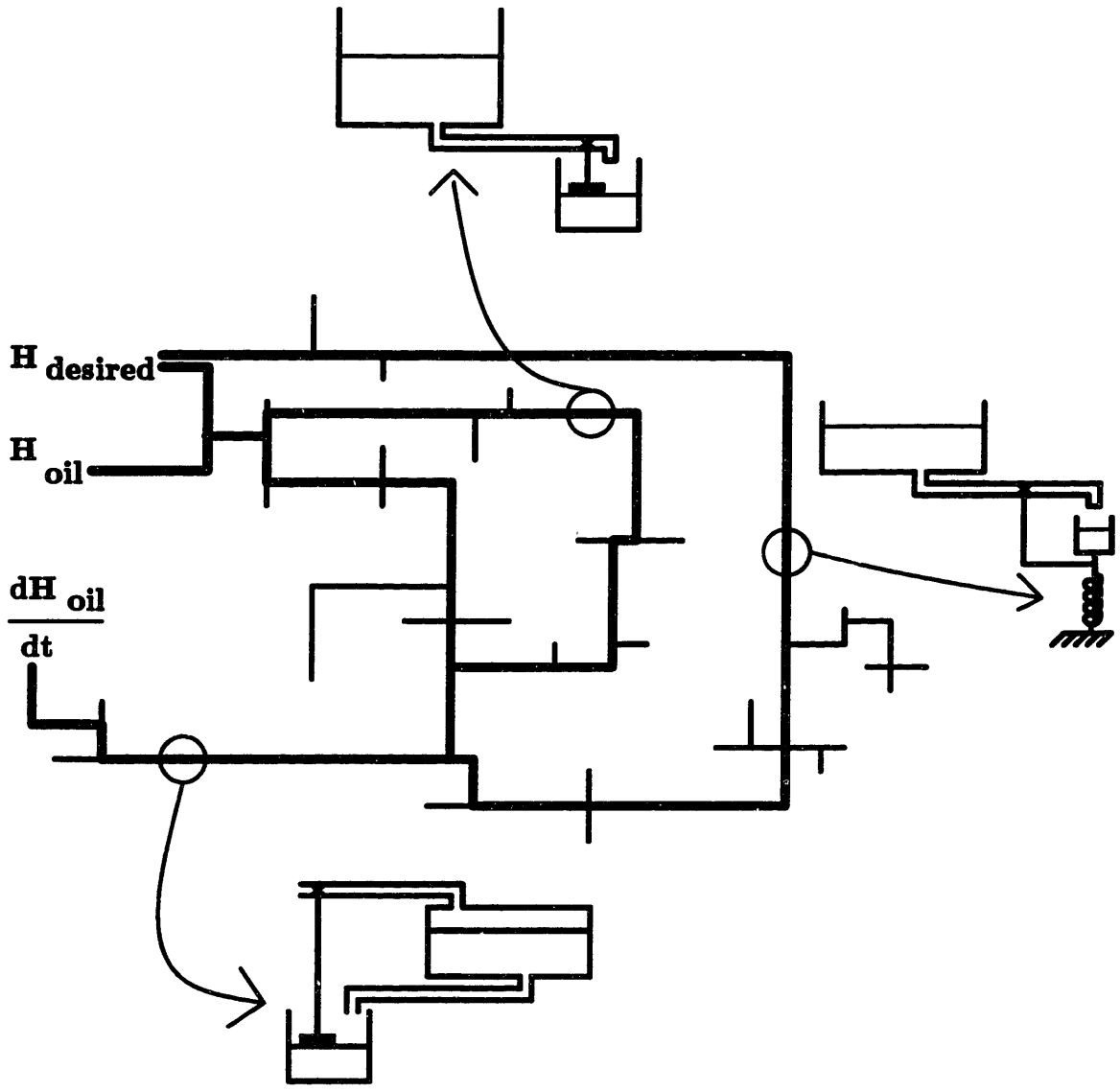


Figure 1.14: Distinct paths through the topology of potential interactions correspond to innovative alternatives.

for a technology, and highlights innovative alternatives allows us to make good on the claim that we are doing innovative design from first principles.

Design from Second Principles

Invention using first principles alone is made difficult because the interactions produced by primitive components are often complex. Thus it is not obvious how to combine them to produce simple interactions. As a design community acquires experience with a technology, they simplify the process by developing a compact vocabulary of composite structures (called *Second Principles of Design*), whose behaviors are simpler than primitive components. These principles establish a direct link between simple interactions, or quantity shifts, and augmentations to a device's physical structure. A device is constructed by decomposing the description of each desired interaction into simple interactions, then using the second principles to identify physical structures corresponding to each of these interactions. Although not universally applicable, these principles are powerful enough to provide the building blocks for efficiently constructing most designs in a particular domain.

Design Evolution through Focused Innovation

Complex devices are rarely developed from first principles in a single step. Rather, invention is an evolutionary process involving a progression of focused design changes, continuously spurred on by the need for higher performance devices. The sequence of design changes for the progression of MOS memory buffers is shown in Figure 1.15.

Each change is marked by an innovation – a novel change to how a device works through alterations to its interaction topology. Each alteration is the result of focussing innovative design strategies on critical points of a device. At each stage in this process the inventor pushes a leading edge design for performance until it breaks, then uses the insight gained about the cause of failure to find ways to improve the design. This involves identifying changes to the structure of existing interactions

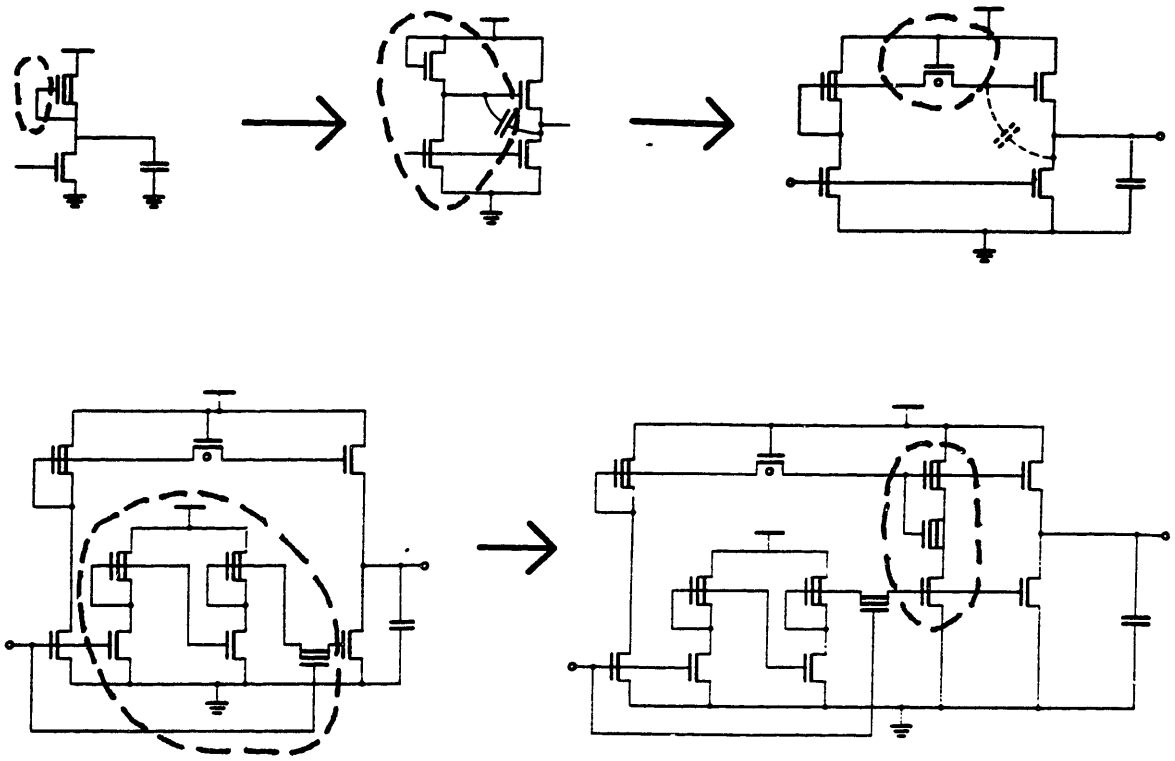


Figure 1.15: Evolutionary progression of MOS memory buffers with modifications indicated.

and points where new interactions are required. These modifications are made by strengthening, weakening, adding or eliminating interactions during windows of time.

For example, consider the first step in the evolution of the MOS memory buffer (Figure 1.16), which starts with a circuit called a ratioed inverter. The design goal is to minimize the time that the output of the buffer takes to rise and fall. The ratioed inverter is inadequate because its output rises too slowly. Examining its topology of interactions (lower left corner of Figure 1.16), the rise time of the output (V_{pd}) is limited, in part by an interaction that constrains $V_{g,s}$ to be 0 (the circled interaction in the left corner of the figure). To improve the rise time, this interaction is replaced by a set of interactions that makes $V_{g,s}$ positive over the window of time where the output is rising (lower right hand corner of Figure 1.16). These modifications are reflected by changes to the inverter's physical structure, shown in the upper right hand corner of Figure 1.16.

This strategy of focussed innovation allows the designer to hill climb towards sophisticated, high performance devices through a sequence of simpler innovations. Furthermore it allows him to exploit past experience and simple design strategies (e.g., design from second principles) to construct most of the design efficiently, while concentrating more innovative techniques only where needed. This results in a graceful integration of routine and innovative design techniques.

Together these strategies span both ends of the design spectrum from the early exploration of simple inventions in a new domain to high performance design in an established technology. The first strategy uses only knowledge of first principles and thus is appropriate when first exploring a domain. On the other hand, the third approach relies on knowledge developed from past explorations in a domain. Thus it is more appropriate for exploration of a mature field, where more is known, but the designs are also more complex due to extreme performance constraints. The second approach provides a bridge between the other two approaches. Using only first

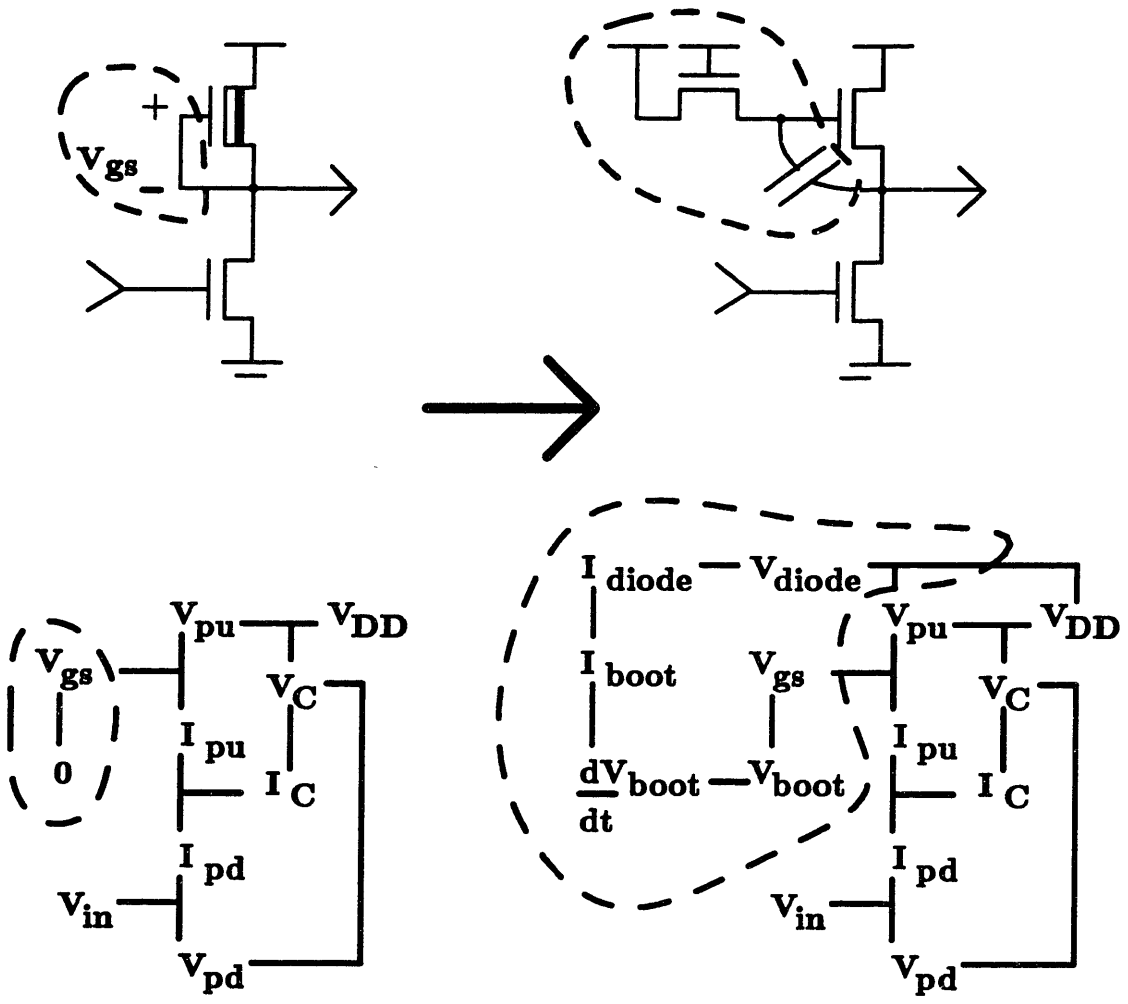


Figure 1.16: Each step in the evolution involves an innovation, consisting of a change to the interaction topology.

principles a complex device can evolve by successively pushing a simpler device for performance and enhancing it. In an established domain, past inventions and second principles can be incorporated in this second strategy to ease the computational burden.

We reiterate that the first strategy is the focus of this document; it is discussed in Chapter 3. The second two strategies are currently in the process of being developed, and will be described in future documents. The next two sections highlight the representations and procedural skills we have developed to support these three strategies. This document presents in detail only those representations and skills which support the first strategy. Those not discussed are indicated in their respective section, and references are supplied where they are discussed more extensively.

1.8 Representing Interaction Topologies

Thus far we have presented interaction topologies pictorially as graphs, in order to convey in simple images the basic intuitions behind interaction-based invention. However, the representations required to capture how a device works are far more sophisticated. A crucial element to focusing the invention process is capturing the topology of interactions at the right level of abstraction – that is, capturing only the salient features of the interaction topology with respect to how a device works. Developing a representation that captures this level of abstraction is a central concern of this thesis.

Producing a set of quantity shifts in a continuous system is a subtle process, often requiring a complex topology of interactions. Movements between intervals do not occur instantaneously. Either a quantity is nudged in a particular direction and it is carried to its destination by momentum, or it is pushed to its destination by a sustained influence. This is quite different from the sequence of discrete, instanta-

neous actions used in many AI planning approaches[16,40,1,43] or digital finite state machine design[18]. Furthermore, these influences are often not easily turned off. Instead the device must orchestrate a set of competing influences that combine to move a quantity in a particular direction. A device's behavior changes over time by strengthening, weakening, adding or eliminating interactions during windows of time.

The interaction topology representation, and the invention approach, must capture these aspects of continuous behavior. To be a good representation it must make it possible to express *all and only* the salient properties of interactions – properties necessary to account for how the desired quantity shifts are produced. This is absolutely crucial – given the difficulty of inventing novel devices from first principles, if the representation forces any superfluous details to be introduced then Ibis will quickly become lost.

Current representations for capturing how devices work are inadequate. Overcoming these inadequacies requires a new perspective with respect to qualitative, temporal and causal representations, and their integration. Our research makes three notable contributions along this front:

- *Q1* – Existing qualitative representations are too weak to capture the relevant constraints between quantities imposed by interactions – they over-abstract. Avoiding over abstraction requires a representation that allows the abstraction of each interaction to be selected along a qualitative/quantitative spectrum. We propose such a representation based on a hybrid qualitative/quantitative algebra, called Q1.

- *Concise Histories* – Existing temporal representations of behavior which impose a total ordering on events (e.g., the situational calculus) are too restrictive to capture only the salient properties of dynamic interactions – they introduce too many irrelevant temporal orderings and events. Our solution, called *concise histories*, is to use a compact description of dynamically changing interactions

to identify only those events and temporal orderings that impact how a device produces its desired behavior.

- *Causal Dominance* – Due to the pervasiveness of feedback, the set of interactions that contribute to device behavior is often exceedingly complex. Inventors cope with this by constructing causal accounts for how a device behaves in a particular situation, that substantially reduces this complexity without significantly altering the behavior accounted for. Existing theories of causality neither explicate or correctly predict these accounts. We have developed a theory, called *causal dominance*, which proposes that causal accounts are constructed by tracing paths through an interaction topology along the paths of dominant signal flow. The net effect is to break each feedback loop at its weakest point. These accounts are used when altering an invention, to identify quantities and interactions that have the greatest impact on device behavior.

The qualitative algebra $Q1$ – is used by the strategy of tracing interaction topologies, and is discussed in Chapter 5. The second two representations – concise histories and causal dominance – are used by the the strategy of design evolution through focussed innovation. These representations are not discussed further here; however, early versions of these representations and their use are presented in more detail in [59] and [63], respectively. A more detailed exposition of how these two representations fit into interaction-based invention will be included in [61].

1.9 An Inventor's Procedural Skills

The phrase "design from first principles" indicates the importance of first principles as a foundation for achieving a robust and domain independent theory of invention. However, this does not imply that all first principles are explicitly manipulated by the

design strategies. Typically the first principles focus one level of granularity below basic operations required for invention.

To make IBIS efficient, many of these principles are embodied in a set of *procedural skills.* These skills are an efficient set of procedures, such as algebraic manipulation, that raise the granularity of first principles to operations of interest for invention – without sacrificing robustness. For interaction-based invention these include operations for constructing interaction topologies and their corresponding physical structure, determining the composite behavior of a topology, and comparing it to the interactions desired. These skills facilitate IBIS' efficiency by placing it in the role of coordinator – using the strategies to decide which interactions and structures to add – while using the procedural skills to handle the details of performing and evaluating the additions. The skills necessary to build interaction topologies use a sophisticated combination of qualitative, constructive, causal and temporal reasoning, drawing substantially upon and requiring significant advances in the corresponding areas of AI. The following are three of the most notable skills developed for our theory of invention:

- *Minima* – Determining whether an interaction topology produces the desired interactions requires operations for combining and comparing interactions. Given that interactions are described as qualitative equations, this require skills for qualitative algebraic manipulation. These skills are embodied in a powerful symbolic algebra system, called *Minima*, for the Q1 algebra. The existence of this system refutes the growing belief in the qualitative reasoning community that qualitative algebras can never be made powerful enough to do interesting tasks[51], such as verification or design.

- *Iota* – To construct an interaction topology, interactions are "joined" together by sharing a variable between them. Sharing a variable may impose additional constraints on physical structure; for example, two pipes can share pressure

by connecting them together. We refer to this process of joining together interactions as *constructive reasoning.* Determining the constraints on structure involve reasoning about the relationships between classes of variables, and parts of physical structure the variables depend on. These are the types of relationships manipulated by traditional knowledge representation systems. All of Ibis' constructive reasoning skills are embodied in a simple, complete and efficient knowledge representation system, called *Iota.*

Over the last decade researchers in the knowledge representation community have found that their is a computational cliff that most knowledge representation systems fall off (e.g., [9,34,55]) – computing subsumption and classifying concepts is NP-hard for even extremely impoverished knowledge representation languages [8]. Thus Iota's completeness, efficiency and adequacy for interaction-based invention provides an important counter example.

- *Episode-based Prediction* – A domain independent analysis technique has been developed that constructs concise histories – a topology of dynamically changing interactions unraveled over time – to describe how a device produces its time-varying behavior. This technique captures important components of an inventor's skill at explaining how a device works in response to changing inputs This plays a crucial role during evolutionary design at identifying points in an interaction topology where innovative changes are required.

Minima and Iota are discussed respectively in Chapters 6 and 8. Episode-based prediction is not used in the design strategy discussed in this document – tracing interaction topologies – and thus is not presented further in this document. A more detailed exposition of it is presented in [59].

The procedural skills, coupled with the design strategies, interaction topology representation and first principles, discussed in the preceding sections, are the basic elements that enable IBIS to focus the process of invention, without sacrificing

robustness.

1.10 Road Map

The major components and subcomponents of Ibis are shown in Figure 1.17, where the first three levels in this diagram correspond to the strategies, procedural skills and interaction topology representations. The bottom level is a problem solving architecture (CTMS and Builder) that has been developed to support our approach (described in part in [60]). A circle is drawn around the components of Ibis that are discussed in detail in this document.

The remainder of this thesis is organized as follows: Chapter 2 describes the three major transformation stages of interaction-based invention, highlighting the representations and principles involved. The remainder of the document focuses on the last stage, where the interaction topology is constructed. Chapter 3 describes the design strategy – tracing topologies of interaction – where simple, innovative devices are constructed from first principles alone. The major claims of the thesis with respect to our theory of invention are then summarized in Chapter 4.

The second half of the thesis discusses the representations, principles and procedural skills the support this strategy. Chapter 5 discusses the hybrid algebra Q1, used to describe how interactions behave, and then Chapter 6 presents the qualitative symbolic algebra system, Minima, used to reason about the composition of these interactions. Chapter 7 summarizes the first principles used to determine what interactions are producible, and Chapter 8 describes the constructive reasoning component, Iota, that is used to connect together interactions into topologies, through augmentations to physical structure.

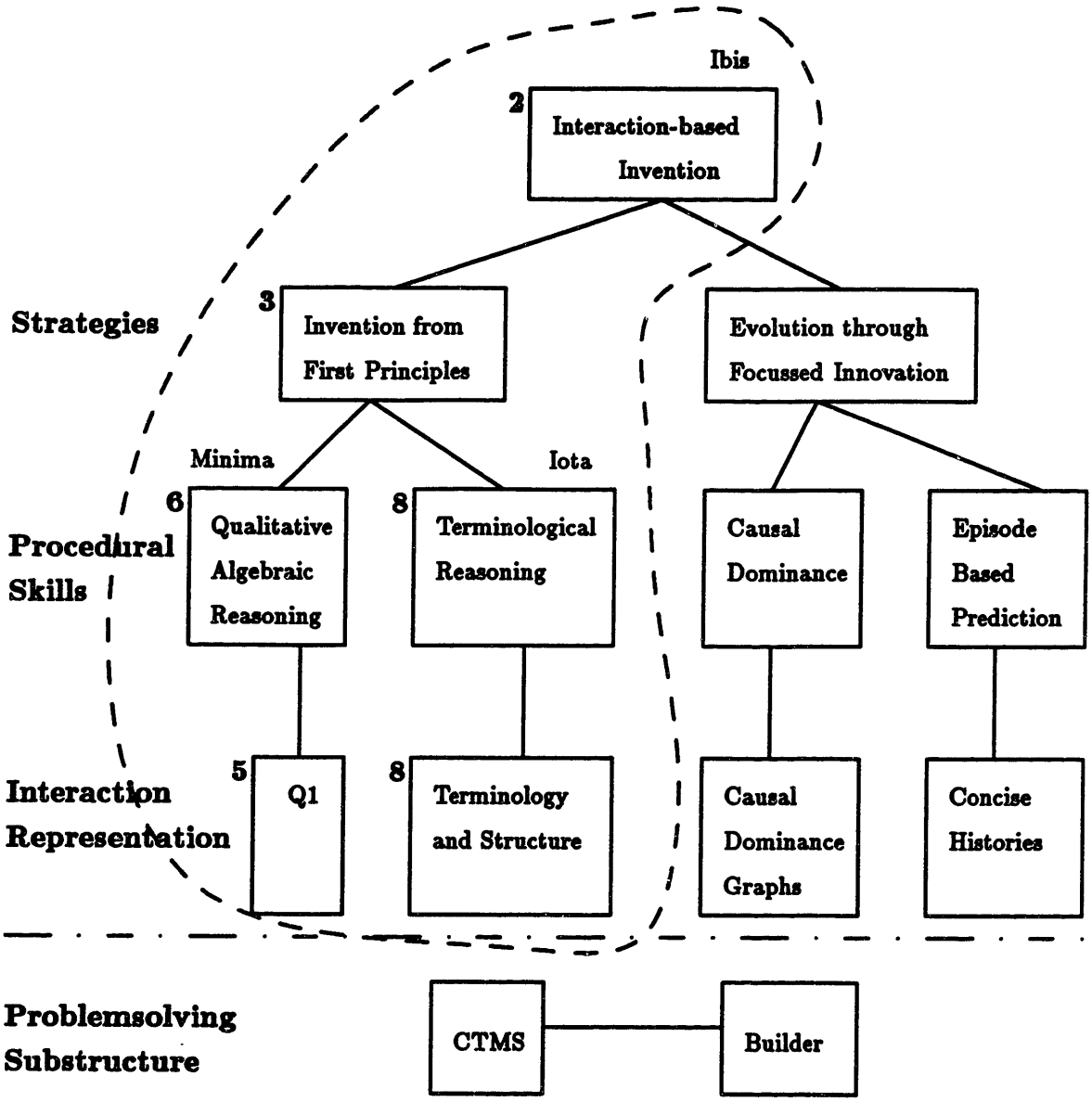


Figure 1.17: Major components of Interaction-based Invention.

Major components of Interaction-based Invention. The circle denotes components discussed in this document. Numbers indicate the chapters where each component is discussed.

Part I

Strategies for Invention

Chapter 2

Interaction-based Invention

This chapter develops the major components of interaction-based invention and its implementation *Ibis* (for Interaction-Based Invention System). The development is motivated by a simple fluid example that highlights the complexity of interactions produced by apparently simple devices (Section 2.1). This is used to provide a concrete description of what interactions are (Section 2.2), and the basic stages of the interaction-based invention process (Section 2.3). Section 2.4 describes the first principles, operations and strategies used by Ibis at each stage, focusing primarily on the initial stages of the process. The final stage, where a detailed interaction topology is constructed, is discussed in the remaining chapters.

2.1 Detailed Account of the Punch Bowl Design

Suppose you are throwing a major party that includes beverages. Having waiters manually refill the punch bowl from a large vat would intrude on the ambiance of the event. Thus you would like the level of the punch bowl to be restored to the level

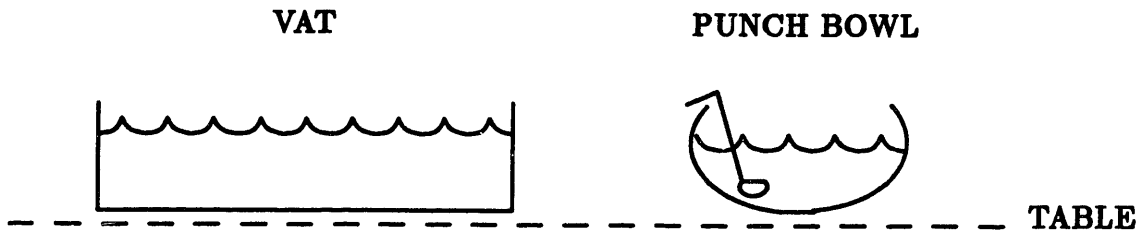


Figure 2.1: Initial setup for the punch bowl problem.

of the vat automatically. That is, whenever there is a height difference between the bowl and vat, a device should automatically change the bowl height to meet that of the vat.

There are some additional constraints on the problem. The vat and bowl are sitting on a table, open to the air. The vat is quite large, so you don't have to worry about refilling it, and its level will only drop negligibly throughout the evening. Connections, such as pipes, are only allowed to be made to the bottoms of the vat and bowl; connections to the tops would be unsightly. At your disposal are some standard kitchen items and pipe fittings left behind by the plumber: pipes, valves, containers, a faucet and even the kitchen sink. Assuming for the moment that you have no previous design experience, you must rely only on a basic knowledge of mathematics and the physics of fluids.

You begin by considering the possible cases: The punch bowl is either below or at the same height as the vat, the height you desire. If the punch bowl's height is lower than desired, it must be restored to the desired level. The height is restored by increasing it relative to vat height until the heights are the same. If the punch bowl is at the desired height then it should remain so. This is accomplished by holding the height constant relative to vat height. Thus in general, whenever the vat/bowl height difference is positive, the change in this difference should be negative, and whenever the difference is zero, the change should also be zero. These two cases are accomplished by changing the height difference in the direction opposite to the

difference. Finally, the vat is so large that its height remains essentially constant throughout the party; only the bowl height can be changed. Thus, bowl height is to be changed in the same direction as height difference.

Given this refined goal you begin searching for ways of relating height change to height difference. None of the components you have available to add to the design will directly affect height or height change. However, fluid height is already related to quantities like volume, flow and pressure; thus, you search for ways of relating height and height difference indirectly by relating these other variables.

First, considering height change you recall from physics that the height of fluid in a container times the cross sectional area is equal to the volume of fluid. Thus, height change is in the same direction as change in volume. Using this fact you replace height change with volume change in the desired interaction. The resulting goal is: produce a change in punch bowl fluid volume in the same direction as change in vat/bowl height difference. In addition, you recall that volume is increased or decreased by having fluid flow in or out of it; thus, volume change is controlled by fluid flow. Using this fact you replace fluid flow for volume change in the current goal, resulting in the new goal: produce a fluid flow into the punch bowl in the same direction as change in vat/bowl height difference.

Next you consider what quantities affect height difference. You know that height is related to pressure; more specifically, that the pressure at the bottom of a container is equal to the product of the height and density of the liquid in the container times gravitational acceleration. Also, the liquid density for the vat and punch bowl are the same, since both contain punch. Combining these two facts you realize that the height difference between vat and bowl is in the same direction as change to pressure difference between the bottoms of the vat and bowl. Using this correspondence you replace pressure difference for height difference in the goal, resulting in: produce a fluid flow into the punch bowl is in the same direction as vat/bowl pressure difference.

This new goal is much more workable – you know of components that relate pressure and fluid flow. Specifically, you know for a pipe that fluid flows to the end that has the lower pressure. Thus our task is completed simply by attaching a pipe between the bottoms of the vat and bowl (and, for aesthetics, hiding the vat behind a tasteful and rare tapestry) (Figure 2.2).

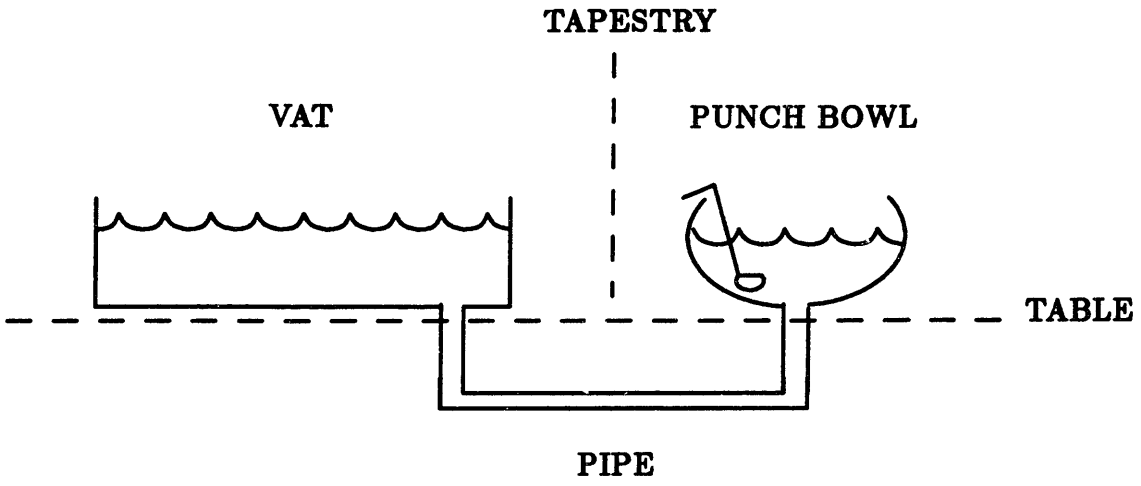


Figure 2.2: Using a pipe to restore the punch bowl level.

The reason we chose this example is its similarity to other quantity shifting devices explored in this thesis. The punch bowl problem (raising punch level) is basically the same as that solved by ancient fluid devices, such as Philon's lamp (raise the oil level), and modern logic devices (raise the voltage level). Except in these other cases the solutions are more elaborate.

In this example both the desired behavior (raise the fluid level) and the resulting device structure (attach a pipe) are quite simple. Thus it is striking, even at this informal level, that the reasoning process is so complex. This highlights the difficulty faced by the inventor when reasoning from basic principles of physics.

As we explore this example in more depth throughout this thesis, we observe that constructing this type of device draws upon an extensive set of principles from physics

and mathematics (e.g., algebra and calculus), as well as a variety of reasoning skills (e.g., algebraic, causal, temporal, and terminological reasoning) used by the inventor to reason from these principles efficiently. Capturing these skills requires significant advances in AI within each of these areas of reasoning.

The remainder of this chapter uses the punch bowl example to explore two questions: First, how is the interaction topology represented for level shifting devices? Second, how is this interaction topology and its corresponding device structure implemented?

2.2 Interactions

Section 1.1 of the introductory chapter argued the central importance of interactions based on the following points:

- an innovative device is one that works in a way qualitatively different from its predecessors,
- "how a device works" is captured by the interactions between quantities and their effects, and
- "qualitative differences" are captured by the topological structure of these interactions.

Thus a representation for interactions is a central component of our theory of invention. To be a good representation it must allow us to capture all and only the salient features of interactions with respect to how a device produces the desired behavior. If we don't have this Ibis will not be able to sort out qualitative differences in how a device works from superfluous details, and thus won't be able to focus on what makes a device innovative. To develop this representation we use the punch

bowl example to identify the properties of interactions that must be captured. The following observations and claims are made in this section:

- Each statement in the account of the punch bowl example describes an interaction. This supports our claim about the central importance of interactions during invention.
- The invention of quantity shifting devices involves using many different types of interaction at different stages of the invention process.
- These types of interactions are characterized along several dimensions – qualitative or quantitative, extensional or intensional, static or dynamic, and causal or constraining. Each stage of invention alters one dimension of an interaction.
- Capturing salient features of interactions with respect to how a device works requires major advances in qualitative, causal and temporal representations.

The primary focus, in Section 2.1, throughout the discussion of the punch bowl problem is on interactions. The discussion begins by considering the relationship between height change and height difference, moves to interactions that already exist between height and other quantities (e.g., height change is in the same direction as flow), and then considers new ones that could be added (e.g., pressure difference is related to flow by adding a pipe). This helps make clear why we believe that interactions are not only crucial to understanding existing innovative devices, they are the central focus of the invention process itself.

The punch bowl example uses a variety of interactions including ones we would like to achieve ("[change] the height difference in the direction opposite to the difference"), interactions imposed by existing physical structure ("volume [of fluid in the bowl] is increased or decreased by having fluid flow in or out of it"), and interactions that can be produced by augmenting the physical structure ("you know for a pipe that fluid flows to the end that has the lower pressure").

The interactions used vary along several dimensions. First, they combine qualitative and quantitative information. At the qualitative level, many of the interactions relate the signs of quantities, differences between quantities or changes in quantities. For example the interaction "[change] the height difference in the direction opposite to the difference" relates the sign of a change to the sign of a difference. This is captured by the equation:

$$[d(Hd_{v,b})/dt] = [-Hd_{v,b}]$$

where $[x]$ denotes the sign of x, $Hd_{v,b}$ the height difference from vat to bowl, and $d(Hd_{v,b})/dt$ the change in height difference. Equations on signs are the types of relations typically studied in qualitative reasoning research[3].

At other times the designer must reason about the precise interaction between quantities at a quantitative level, rather than simply relating the signs of quantities. For example, the exact relationship between fluid density, height and pressure must be known to determine that "height difference between vat and bowl is in the same direction as change to pressure difference." The answer is easily stated in qualitative terms, but deriving it requires quantitative reasoning.

To see why this is the case we examine the argument necessary to determine the correspondence. Let H_v denote fluid height in the vat, d_v denote fluid density in the vat, and P_v denote pressure at the bottom of the vat. Let H_b, d_b and P_b denote the analogous variables for the bowl. Finally let g denote the gravitational constant. We know from physics that for any container c:

$$H_c = P_c/(d_c \times g)$$

thus, the vat/bowl height difference, $Hd_{v,b}$, is:

$$Hd_{v,b} = H_v - H_b = P_v/(d_v \times g) - P_b/(d_b \times g)$$

Using the fact that the vat and bowl densities are the same (both contain punch):

$$d_v = d_b = d$$

the height equation is rewritten as:

$$Hd_{v,b} = (P_v - P_b)/(d \times g) = (Pd_{v,b})/(d \times g)$$

Since gravity (g) and density (d) are always positive, we conclude that height and pressure difference are in the same direction,

$$[Hd_{v,b}] = [Pd_{v,b}]$$

This is the desired conclusion. We could not have inferred this interaction without knowing the exact relationship between height and pressure, and the fact that the two fluid densities are equal. Thus qualitative and quantitative information must be combined.

In general the inventor must reason at several points along a spectrum of qualitative/quantitative information, depending on the properties of the design problem and components being used. The language used to describe interactions must capture this spectrum. In Chapter 5 we show that existing qualitative algebras do not provide an adequate language for invention. We then propose an algebra, called $Q1$, that is sufficiently expressive to capture this spectrum.

Along a second dimension, interactions may be described *extensionally* or *intensionally.* Some interactions are extensional; i.e., they relate quantities by enumerating specific values, as in "whenever the vat/bowl height difference is positive, the change in this difference should be negative." This is captured by the expression:

$$[Hd_{v,b}] = [+] \text{ implies } [d(Hd_{v,b})/dt] = [-]$$

where [+] denotes positive and [−] denotes negative.[1]

Other interactions are intensional, describing interactions by using equations, as in "produce a fluid flow into the punch bowl in the same direction as the vat/bowl height difference" ($[Q_b] = [Hd_{v,b}]$). These relations are typically on signs or real values.

Interactions also have a temporal component, they may be *static* or *dynamic*. The interactions in the previous paragraph are static in the sense that they hold for all time. On the other hand, the expression " ... the height [of the punch in the bowl] is restored by increasing it relative to vat height until the heights are the same" is dynamic. The interaction to restore the bowl height ($[dH_{v,b}/dt] = [-]$) comes into existence when the height is too low, is sustained for an interval of time, and then vanishes when the height is restored. This interaction is represented by a state transition shown in Figure 2.3.

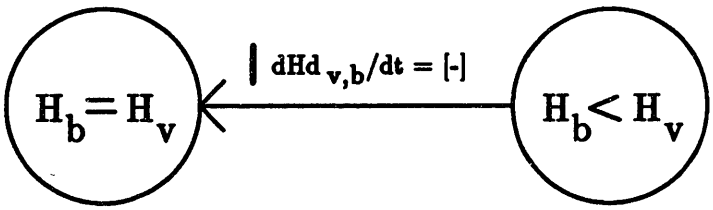


Figure 2.3: A dynamic interaction representing a quantity shift can be represented by an arc in a state diagram. The arc shown in this figure says the increase in bowl fluid height relative to vat height is sustained until the vat height is reached. An arc represents a state transition that occurs over *an interval of time*. The label on an arc is of the form "x | y," where x lists conditions that must be sustained until the transition occurs, and y lists conditions that hold while x is sustained. In this diagram no conditions (x) are necessary for the transition to occur, and $dH_{v,b}/dt$ is negative until the transition completes.

[1] [ ] is placed around + and − to distinguish positive and negative from addition and subtraction.

Interactions may also be *causal* or *constraining*, depending on whether the interconnection between variables is directional or not. If its directional, then the interaction is a causal relationship. If it is non-directional, that is the effect can flow in any direction, then it is a constraining relationship.

Interactions specified as goals of the design process are often causal – the interaction specifies that one set of quantities controls another quantity. For example, in the interaction "[change] the height difference in the direction opposite to the difference," change in bowl height is controlled by height difference. This is denoted:

$$[Hd_{v,b}] \Rightarrow [-d(Hd_{v,b})/dt]$$

where direction of control is denoted by the arrow head on the equality relation.

Interactions describing the behavior of primitive components are often constraining – the interaction does not restrict which quantity is dependent on the others. For example, the behavior of a pipe is described by the interaction "flow through a pipe is in the same direction as the pressure difference across the pipe." This is denoted $[Q_{t1}] = [Pd_{t1,t2}]$, where $t1$ and $t2$ are the two ends of the pipe. This interaction allows pressure difference to control flow or vice versa. To see this consider the two directions. Suppose a pipe is connected between two containers with different pressures – the height of fluid in one container is greater than another. Then the pressure drop across the pipe will cause fluid to flow into the container with the lower pressure (and lower height), that is $[Pd_{t1,t2}] \Rightarrow [Q_{t1}]$. On the other hand, if a pipe is connected to a steady flow source, such as a faucet, then the flow through the pipe causes a pressure build up at the faucet relative to the other end of the pipe ($[Q_{t1}] \Rightarrow [Pd_{t1,t2}]$).

These four aspects – qualitative versus quantitative, extensional versus intensional, static versus dynamic and causal versus constraining – must be captured by a language of interactions. Furthermore, to be a good representation this language must capture the salient properties of how level shifting devices work. That is *all and*

only those features of interactions that affect how the device performs the desired quantity shifts. This is absolutely crucial – *given the difficulty of inventing novel devices from basic principles, if the representation introduces any superfluous details then the inventor quickly becomes lost.*

In Section 1.8 of the introduction we claimed that existing qualitative, temporal and causal representations cannot capture these four dimensions of interactions at the right level of abstraction. Qualitative representations developed thus far, which are based on sign algebras, are too weak to capture the more quantitative components of interactions. Earlier temporal representations, which are based on global states, are too constraining, requiring irrelevant events and temporal orderings to be introduced into descriptions of interactions. Earlier causal representations do not provide an accurate theory of what these causal relationships reflect about a device's operation. Our contribution with respect to solving these three problems were summarized in Section 1.8 of the introduction. Each of these different components of our interaction representation are discussed as they become relevant to particular aspects of interaction-based invention. Next we explore how these basic properties of interactions shape the process of invention.

2.3 Invention as Transforming Interactions

In this section we observe that the design of quantity shifting devices involves a series of transformations between different types of interactions, ending in a physical structure.

- The initial transformation stages are relatively simple, and involve a process analogous to digital finite state machine design[18].

- The difficulty is in the last stage where the complexity of physical components must be confronted directly. This stage is where a detailed interaction topology must be constructed that makes evident the topology of physical components.

This section describes what operation is performed by each transformation – Section 2.4 discusses how they are performed.

2.3.1 The Transformations

Our task involves a succession of three transformations: It begins with a set of dynamic, extensional interactions which are transformed to static extensional interactions. Next these are transformed to intensional interactions. Finally they are transformed to a physical structure. These three transformations are summarized in Figure 2.4.

The three transformations are apparent in the account of the punch bowl example (Section 2.1). The first two transformations are described in the first part of the account, repeated below:

> You begin by considering the possible cases: The punch bowl is either below or at the same height as the vat, the height you desire. If the punch bowl's height is lower than desired, it must be restored to the desired level. The height is restored by increasing it relative to vat height until the heights are the same. If the punch bowl is at the desired height then it should remain so. This is accomplished by holding the height constant relative to vat height. Thus in general, whenever the vat/bowl height difference is positive, the change in this difference should be negative, and whenever the difference is zero, the change should also be zero. These two cases are accomplished by changing the height difference in the direction

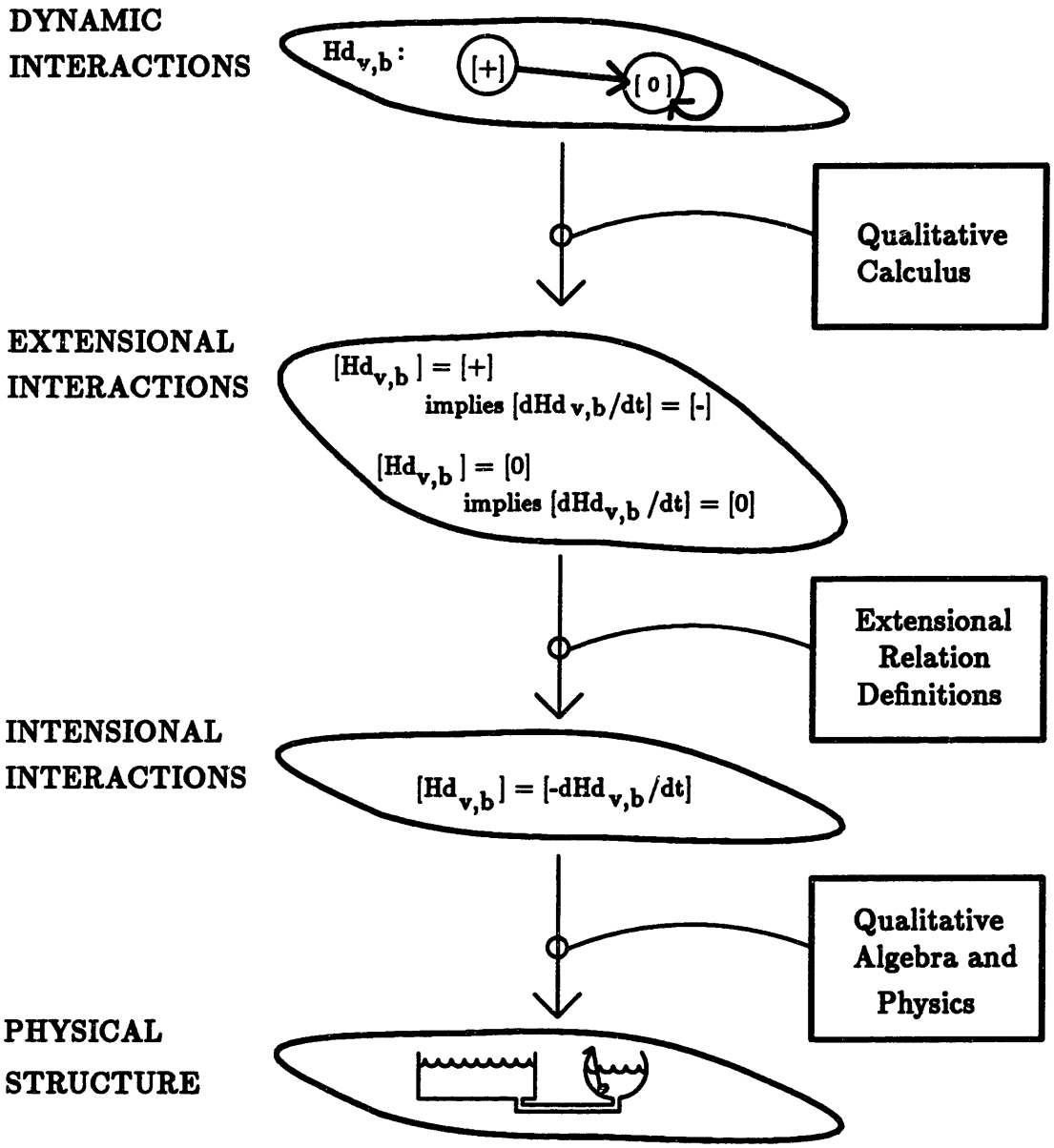


Figure 2.4: Transformations, between interactions and to physical structure, performed during the invention of quantity shifting devices. The class of first principles required for each transformation is indicated to the right of each arrow.

opposite to the difference. Finally, the vat is so large that its height remains essentially constant throughout the party; only the bowl height can be changed. Thus, bowl height is to be changed in the same direction as height difference.

The desired behavior is initially described using extensional dynamic interactions. For example, the third sentence of the description, "If the punch bowl's height is lower than desired, it must be restored to the desired level" is a dynamic interaction, since it describes how height changes over time (from too low to the desired height). This is represented as an arc between two states in the state diagram at the top of Figure 2.4. The description includes a second dynamic interaction, "If the punch bowl is at the desired height then it should remain so." This is described by the second arc in the state diagram of Figure 2.4.

The first transformation stage maps the dynamic interactions to ones that are static. For example, the first interaction is transformed in the seventh sentence to "whenever the vat/bowl height difference is positive, the change in this difference should be negative." This is a static interaction since the relation it specifies between height and height change holds at each point in time. It is represented as:

$$[Hd_{v,b}] = [+] \text{ implies } [d(Hd_{v,b})/dt] = [-]$$

The second interaction is transformed to "whenever the difference is zero, the change should also be zero":

$$[Hd_{v,b}] = [0] \text{ implies } [d(Hd_{v,b})/dt] = [0]$$

These two transformations are shown in the upper half of Figure 2.4.

In this transformation stage the dynamic interactions describe desired shifts of a quantity, in this example height difference $Hd_{v,b}$. The interactions resulting from

the transformation describe constraints on the sign of the quantity's derivative that accomplish each desired shift.

The second transformation in the design account combines the static extensional interactions to produce a single intensional interaction. For example, the two extensional interactions listed above become "[change] the height difference in the direction opposite to the difference":

$$[Hd_{v,b}] = [-d(Hd_{v,b})/dt]$$

The effect of the transformation is to accumulate individual constraints on the signs of quantities to a single equation on signs.

The remainder of the account (Section 2.1) describes the third transformation. It begins with the intensional interaction as a goal, and ends with the decision to attach a pipe between the bottom of the vat and bowl. That is, the intensional interaction has been transformed to augmentations to physical structure. This is the last transformation shown in Figure 2.4.

Note that, in the design account, the part describing the first two transformations is relatively brief (one paragraph), while the part describing the third transformation is quite extensive. This is a reflection of the relative complexity of the different transformation stages. The nature of this complexity will become clear as we discuss the transformation stages in more detail (Section 2.4).

2.3.2 Rationale for the Transformations

Consider a possible rationale for why these transformations are performed. For quantity shifting devices the initial behavior can be described most simply and compactly as the sum of the individual quantity shifts. This suggests the use of a state dia-

gram for the initial specification. On the other hand, the interactions of components and connections are most compactly described intensionally as some form of equation (e.g., as opposed to extensionally) – consider why. Recall from Section 2.2 that, when characterizing interactions, we concluded that in some cases these interactions must be described quantitatively. Because the reals are uncountable, these interactions can only be described intensionally – their extension would be infinite. Thus, before constructing a detailed topology of interactions from components and connections, we must transform the interactions, initially described by a state diagram, into an intensional description (e.g., equations).

Next consider the rationale for the two steps that are taken to perform this transformation. The state diagrams are dynamic extensional interactions, while the equations are static intensional interactions – the interactions differ along two dimensions. To perform the transformation we can change the dimensions of the interactions one at a time – first we change the dynamic interactions into ones that are static, and then change the resulting interactions from extensional to intensional. These are exactly the first two transformation stages we discussed above and highlighted in Figure 2.4.

2.3.3 Analogy to FSM Design

The decomposition into three transformations is roughly analogous to the three stages of digital, finite state machine (FSM) design[18] (Figure 2.5). This analogy will help us to understand the transformation process for our task, and to highlight critical issues and differences.

FSM design begins with a state diagram, transforms each arc to an entry in a truth table, converts the truth table to a set of boolean equations, and then maps each boolean operation to a logic gate. The state diagrams used for FSM design and our approach are roughly analogous. The primary difference is that FSM diagrams

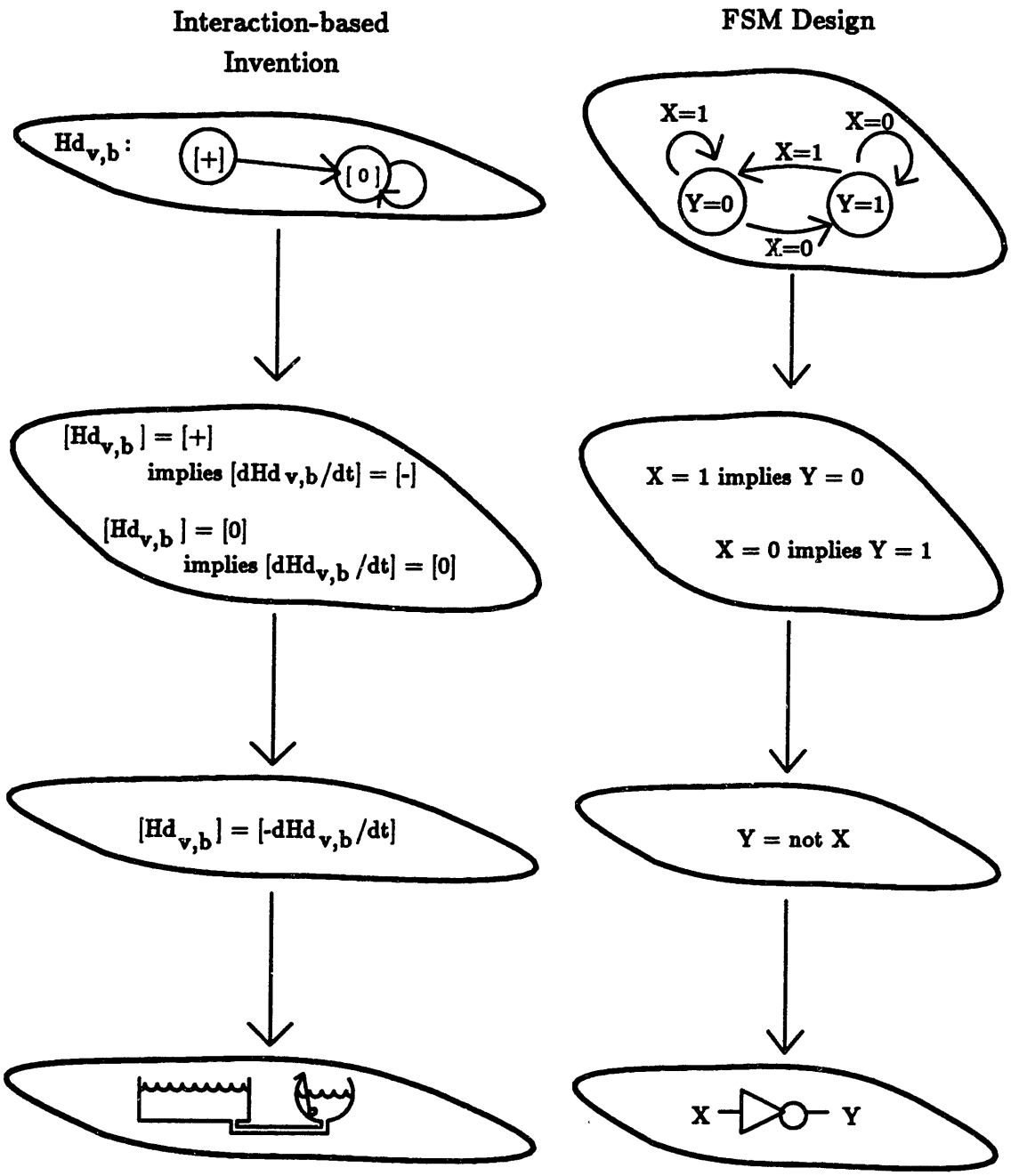


Figure 2.5: The three steps of finite state machine design (to the right) are roughly analogous to the three transformation steps required to construct quantity shifting devices (to the left).

describe discrete changes that occur instantaneously. In contrast our diagrams describe continuous changes that take time. The extensional interactions are analogous to individual entries in a truth table. Using a notation similar to that for extensional interactions, a truth table entry looks like:

$$A = 1 \text{ implies } B = 0$$

In contrast to:

$$[Hd_{v,b}] = [+] \text{ implies } [dH_b/dt] = [+]$$

The primary difference is in the values. Truth table entries specify boolean values (1, 0), while our extensional interactions specify signs $(+, 0, -)$. The boolean equations are analogous to the intensional interactions. The difference again is that boolean equations are on truth values, while the intensional interactions are on signs of quantities. Finally, both design processes end with a description of physical structure – for FSM design these are logic gates, in our case they are lumped elements of circuit theory and physical system dynamics. The next section uses this analogy to explain what is involved at each stage of interaction-based invention.

2.4 What the Transformation Process Entails

Next consider the process underlying each of the transformations stages. Using the FSM analogy, we show that the first two stages of each approach are similar, and can be accomplished simply. However, the last stage – transforming intensional interactions to physical structure – is strikingly different. While the correspondence between boolean operators and logic gates is 1-1 for FSM design, the correspondence between intensional interactions and continuous devices is not at all obvious. This last stage is where a detailed topology of interactions is constructed – the central focus of this

thesis.

In the introductory chapter we stated that Ibis' problem solving abilities are broken into three basic components: The first is a set of first principles about interactions. The generality of these principles ensures Ibis' robustness. However, because of the fine granularity of these principles, many of them are not manipulated directly. Instead they are embodied into a set of tools for efficiently manipulating interactions (e.g., procedural skills), and a set of strategies that use these tools to construct interaction topologies. The next two subsections discuss the first principles, tools and strategies used for each of the first two transformation stages. As mentioned above, the third stage is much more complex, and requires a much more extensive set of principles, tools and strategies. The third subsection discusses the nature of the difficulty. Using the insights gained, we develop the principles, tools and strategies in the remaining chapters.

2.4.1 First Transformation

Recall from Section 2.3 the first transformation takes a set of dynamic interactions, describing quantity shifts, and produces static interactions, describing conditions on the values of quantities that accomplish the shifts. This is analogous to the first step of FSM design, where a state diagram is transformed to a set of constraints on truth values – a truth table. Furthermore, the process by which the transformation is performed in the two cases is similar. The key difference is that, for our task transitions represent qualitative changes in continuous quantities, as opposed to instantaneous changes of discrete quantities.

For FSM design the strategy is extremely simple, a state diagram is transformed one arc at a time. An arc describes an instantaneous transition from one state (S1) to another (S2) that occurs when condition I holds, where S1, S2 and I are described

as sets of truth assignments (e.g., x = 1, y = 0). The result of transforming an arc is:

$$S1 \text{ and } I \text{ implies } S2'$$

That is, the transition is made to happen simply by forcing the device to jump immediately into the new state.

The process is similar for quantity shifting devices. Again the strategy is to transform a diagram by transforming each arc separately. The difference is that FSM design involves discrete quantities, while the quantities in our task are continuous. As a result transitions take time – a quantity can not jump discontinuously to the specified level or region. Instead the shift is accomplished by *pushing the quantity in the right direction until it reaches the desired level or region.* This "push" in turn is performed by constraining the quantity's derivative.

The first principles relevant to performing these transformations are those about how the value of a quantity's derivative affects its time-varying behavior – these are qualitative principles of calculus[65]. The two principles that specify how to perform this transformation are the Intermediate Value Theorem and the Mean Value Theorem. The Intermediate Value Theorem specifies what direction to push a quantity, while the Mean Value Theorem specifies constraints on the sign of a quantity's derivative, necessary to accomplish the push.

More specifically, the Intermediate Value Theorem says that a continuous quantity is brought to a higher level by increasing it until it reaches that level, brought to a lower level by decreasing it, and remains at a level by holding it constant. Next the Mean Value Theorem is used to determine how to make a quantity increase, decrease or remain constant: the simple answer is to make the quantity's derivative positive, negative or zero, respectively.

These two theorems are not used directly to perform the transformation. Instead

we use them to derive a lemma that is applied directly, which we call an *operational principles.* The two theorems are combined into an operational principle, called the *Integration Rule,* that specifies conditions on the sign of a quantity's derivative necessary to accomplish a particular type of shift. This rule is summarized by the state diagram shown in Figure 2.6.

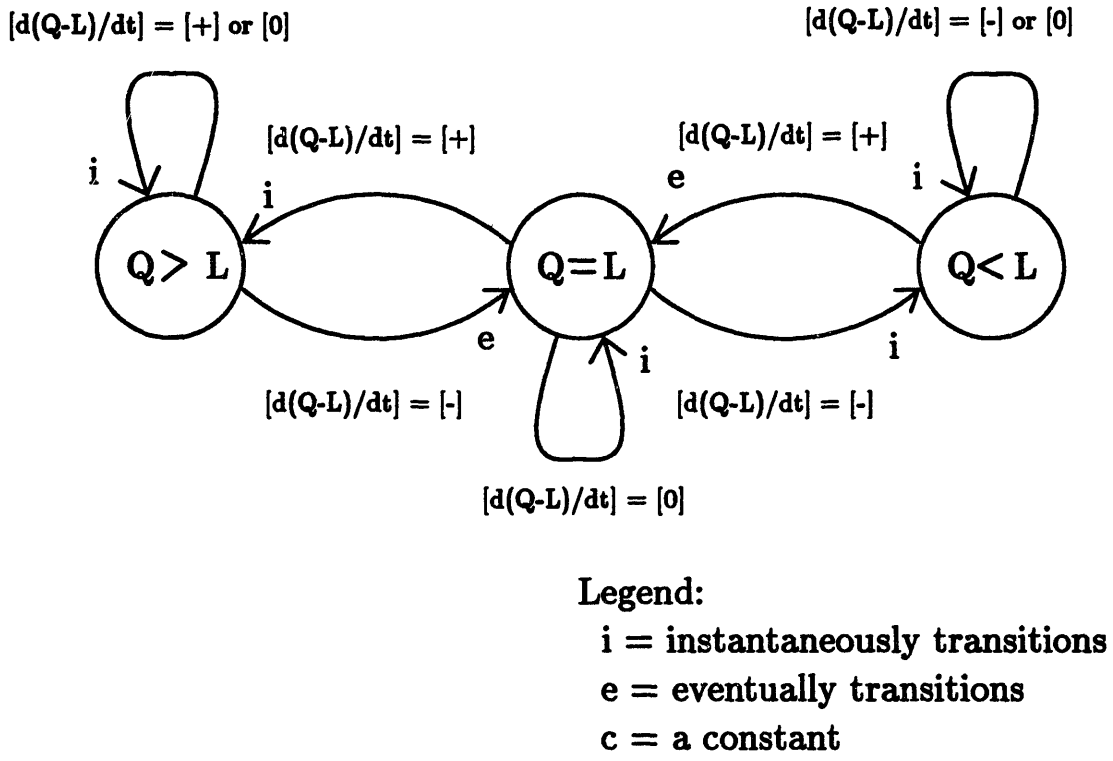
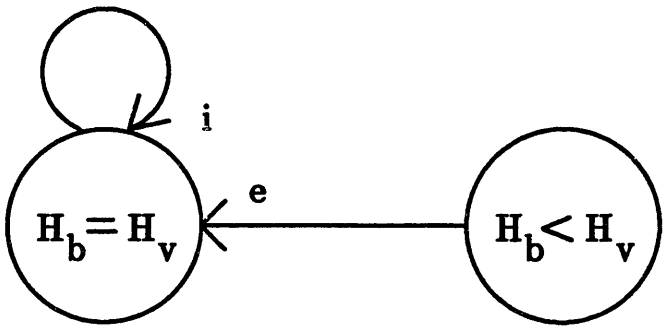


Figure 2.6: Integration Rule: uses constraints on rate of change to force quantities to shift levels over time.

In this figure Q is a quantity being shifted relative to level L. Both Q and L may be time-varying quantities. Each arc describes a movement of Q relative to L, resulting from a positive, negative or zero derivative. For example, the arc from "$Q = L$" to "$Q < L$" tells us that, if Q equals L and $d(Q - L)/dt$ becomes negative, then Q will immediately become less than L. The arc from "$Q < L$" to "$Q = L$" tells us, that if Q is less than L and $d(Q - L)/dt$ becomes positive and is sustained,

then Q will be pushed upwards until it eventually reaches L.[2]



Legend:
i = instantaneously transitions
e = eventually transitions

Figure 2.7: Desired dynamic interactions for punch bowl example describing rise of fluid in bowl to desired level.

Next consider how the Integration Rule is used to transform an arc. Suppose we have an arc specifying that, when a quantity $Q1$ is less than $L1$ and condition I occurs and is sustained, then $Q1$ should become equal to $L1$. Consulting Figure 2.6, the arc from $Q < L$ to $Q = L$ tells us that $Q1$ will eventually reach $L1$ by making $d(Q1 - L1)/dt$ positive. Equivalently, the result of transforming the arc is:

$$[Q1 - L1] = [-] \text{ and } I \text{ implies } [d(Q1 - L1)/dt] = [+]$$

where $[Q1 - L1] = [-]$ is how "$Q1 < L1$" is written in our qualitative algebra.

Returning to the punch bowl problem, the two dynamic interactions (Figure 2.7) are transformed in a similar manner. H_b is changing relative to H_v thus we take Q to be H_b and L to be H_v.

[2]We have ignored some additional constraints on the magnitude of the derivative necessary to guarantee that the transition actually occurs (but see [59]).

One of the dynamic interactions specifies that H_b should rise to H_v. This is transformed into:

$$[H_b - H_v] = [-] \text{ implies } [d(H_b - H_v)/dt] = [+]$$

The second interaction specifies that whenever H_b is at H_v it should remain so. Consulting Figure 2.6 we see that Q remains at L if $[d(Q - L)/dt] = [0]$, thus the second interaction is transformed to:

$$[H_b - H_v] = [0] \text{ implies } [d(H_b - H_v)/dt] = [0]$$

Reviewing the overall process, the first transformation is similar to that for FSM design. The difference lies in the fact that we are changing continuous quantities rather than discrete. In this section we identified the necessary first principles, tools and strategies. The requisite first principles are the Intermediate and Mean Value Theorems of calculus. To raise their granularity they are embodied into an operational principle – the Integration Rule. This is the tool used to transform a single interaction. The strategy for transforming a state diagram is straightforward – apply the integration rule to every arc.

2.4.2 Second Transformation

Recall from Section 2.3 that the second transformation takes the extensional interactions produced by the first transformation and combines them into one or more intensional interactions. The effect of the transformation is to accumulate individual constraints on the signs of quantities into sign equations. Also recall that this is similar to the second step of FSM design where a set of truth table entries are transformed into boolean equations. How the transformation process is performed is

also similar. Put simply, for FSM design this involves recognizing that a set of entries in a truth table is a partial extension of a boolean operator. For example, suppose the arcs of a state diagram are transformed into:

$$A = 0, B = 0 \quad \text{implies} \quad C = 0$$
$$A = 1, B = 0 \quad \text{implies} \quad C = 0$$
$$A = 1, B = 1 \quad \text{implies} \quad C = 1$$

These constraints define a partial function whose truth table is:

A	B	C
0	0	0
1	0	0
1	1	1

It is partial since values for C are left unspecified for A = 0, B = 1. The entries in this table are a subset of those in the truth table defining logical AND:

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

Thus the three constraints are satisfied by constructing the boolean equation:

$$A \text{ AND } B = C$$

Note that, by transforming the constraints to this equation we have made an additional commitment:

$$A = 0, B = 1 \text{ implies } C = 0$$

The process is analogous for our invention task, except that the constraints, operator tables, and equations are on signs $(+, 0, -)$, instead of truth values (1,0). To demonstrate this we return to the punch bowl example. In the last section we saw that the first transformation produced:

$$[H_b - H_v] = [-] \text{ implies } [d(H_b - H_v)/dt] = [+]$$

$$[H_b - H_v] = [0] \text{ implies } [d(H_b - H_v)/dt] = [0]$$

The table defining the equivalent partial function is:

$H_b - H_v$	$d(H_b - H_v)/dt$
$-$	$+$
0	0

This is a subset of the entries defining the minus operator ($\ominus$) of the sign algebra (see Chapter 5):

x	y
$-$	$+$
0	0
$+$	$-$

Thus the two constraints are satisfied by:

$$[H_v - H_b] = \ominus[d(H_v - H_b)/dt]$$

This result makes one additional commitment:

$$[H_b - H_v] = [+] \text{ implies } [d(H_b - H_v)/dt] = [-]$$

That is, if the level of punch in the bowl goes above the vat, then it is lowered until the levels are equal.

Note in the punch bowl problem (Section 2.1) it was assumed that the vat is sufficiently large that its fluid level remains essentially constant throughout the party (i.e., $[dH_v/dt] = [0]$). Thus the equation simplifies to:

$$[H_v - H_b] = [dH_b/dt]$$

Returning to the analogy for a moment, for FSM design this transformation is a bit more complex. Often the state diagrams have many arcs and states, resulting in a very large truth tables. Capturing these tables requires boolean equations involving complicated expressions. Techniques such as boolean minimization have been developed to identify these expressions.

In contrast the state diagrams for quantity shifting devices tend to be small. The technique described here has been adequate for the simple examples used in this thesis. We are only beginning to explore analogs to Boolean minimization.

Reviewing the overall process, the second transformation is similar to that for FSM design. The difference lies in the fact that we are dealing with the signs of quantities, rather than truth values. For this stage the first principles are the tables defining the sign operators. These tables are matched directly with the constraints being transformed, thus they are also the operational principles (i.e., principles that are applied directly). Finally, for the examples in this thesis, the strategy of matching constraints to single operators is sufficient. More extensive techniques are a topic of

future work.

2.4.3 Third Transformation

The remaining transformation stage takes the intensional interactions, produced by the second stage, and constructs a physical structure – a network of devices, such as pipes and containers. This stage is at the heart of interaction-based invention. Here Ibis must finally confront the complexity of continuous devices directly. Producing the desired interaction involves constructing a complex topology of intensional interactions – each interaction produced by a component or connection. When constructing this topology, to avoid being lost in the search space, Ibis requires a sophisticated set of representations, principles, design strategies and tools for building the interaction topology. These are the focus of the remaining chapters of this thesis. This section discusses how this process differs from FSM design, what makes it so difficult, and some basic insights into how Ibis deals with this complexity.

How the Analogy Fails

The analogy to FSM design has been close thus far; however, it breaks down for the last transformation stage. Consider first how the approaches differ. For FSM design this stage takes a boolean equation and produces a logic circuit. The circuit is constructed simply by replacing each boolean operator with a corresponding logic gate and passing the result up from variables or subexpressions by connecting wires. For example, given the expression "(A AND B) OR C", the circuit shown in Figure 2.8 is constructed.

For interaction-based invention the last stage transforms an intensional interaction

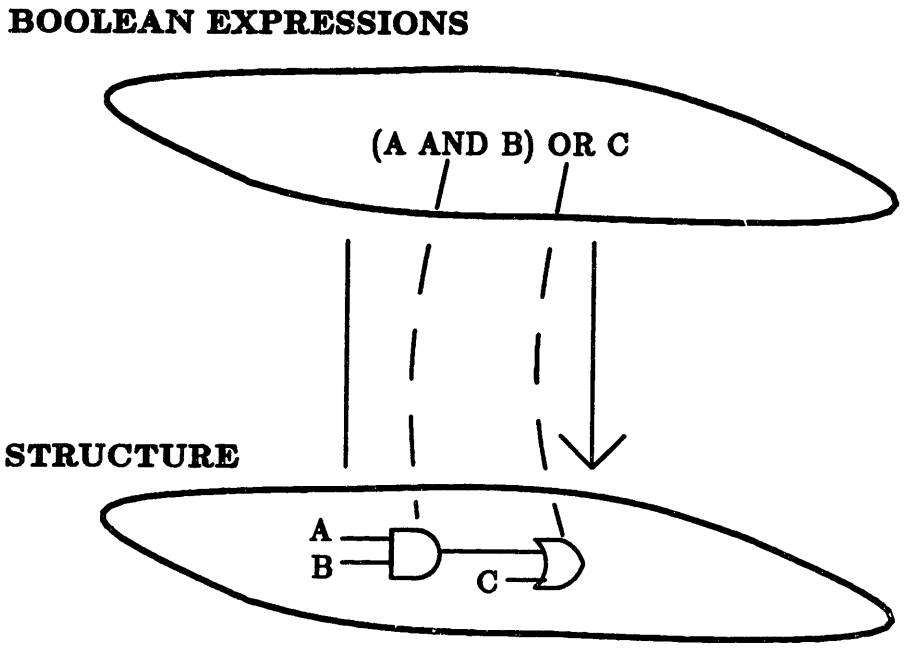


Figure 2.8: Mapping a logical expression to its corresponding digital circuit.

to physical structure. The interaction is a qualitative equation, such as:

$$[H_v - H_b] = [dH_b/dt]$$

taken from the punch bowl example. If the analogy held, then all we would have to do to build the corresponding structure is to map each qualitative operator in the equation (i.e., $-$, [ ] and $=$) to its corresponding "gate" – a component whose behavior is exactly that of the qualitative operator.

However, this is not what happens. Instead the physical structure designed will typically produce a complicated topology of interactions, whose composite behavior is the desired interaction. For example, consider the pipe solution to the punch bowl problem. Figure 2.9 depicts the interaction topology produced by this device as a graph. At this point we are primarily interested in the graph as a measure of complexity; its details are described in the next chapter. Each edge in the graph

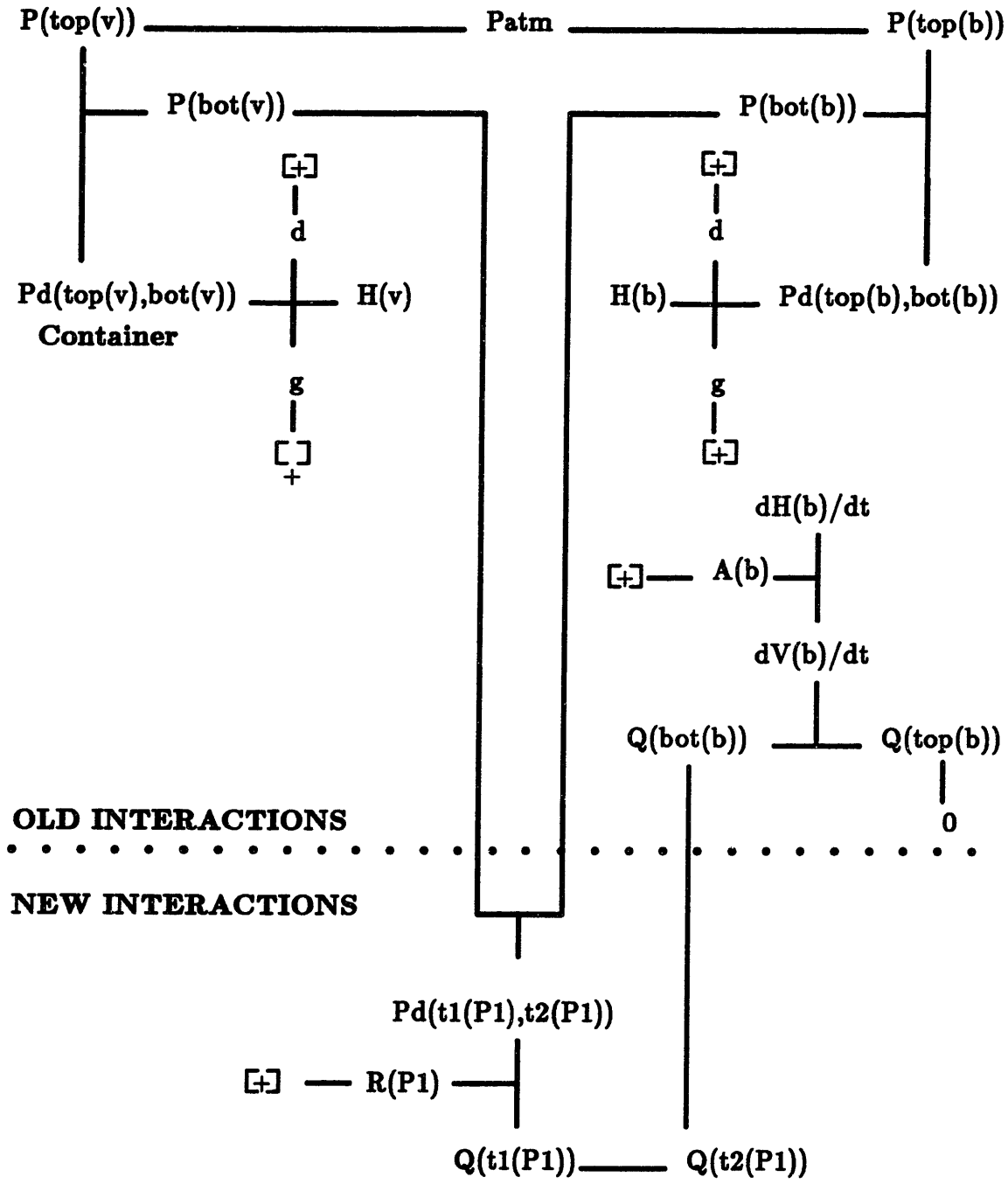


Figure 2.9: A graph of interactions for the pipe solution to the punch bowl problem provides an intuition for the level of complexity of interaction topologies. Each link denotes an interaction. Above the dotted line are existing interactions, produced by the bowl and vat. Below the line are new interactions, produced by attaching the pipe.

represents an interaction, between a set of variables, that is produced by one of the components (vat, bowl or pipe) or connections. Each interaction shown in this graph is necessary to produce the desired interaction – Section 3.5.5 of the next chapter uses every one of these interactions to derive the desired interaction algebraically.

In this example, even though a simple structure is proposed, the topology involves 19 interactions that are directly relevant. For other solutions to the same problem, such as Heron's weight regulator and Philon's lamp, the interaction topologies are much more complex. In contrast the solution to the FSM problem has a topology containing only six interactions, with a one to one correspondence to components and connections. The topology is significantly smaller, even though, in terms of expression size, the problem is of greater complexity.

Why the Interaction Topology is Complex

Next consider what it is about continuous systems that make their interaction topology so complicated, and how this makes the design process particularly difficult. Three factors contribute to the complexity of the interaction topology that are not apparent in FSM design: mismatch between desired and producible interactions, inaccessibility of variables, and parasitic effects.

Mismatch Between Interactions First, often there is a mismatch between the interaction specified as the desired behavior, and the interactions producible by individual components and connections in the current technology. For example, the desired interaction for the punch bowl is to equate the sign of height difference to the sign of height change. None of the component types specified in the problem description produce this interaction alone. In fact none of these components relate height difference to height change in any way. For example, a pipe relates pressure difference to fluid flow and resistance, but not heights.

This highlights a major source of complexity that we don't see in digital design – the types of the variables being related matter (e.g., flow Q vs pressure P). The problem is that often no available component will produce interactions that relate variables of each type specified in the desired interaction. For example, in the punch bowl problem, none of the available components relates height difference to height change.

More generally, a domain will often involve a number of distinct variable types. For example, in the fluid domain we have pressure, pressure difference, flow, height, height difference and volume – 6 variable types, not including constants. For hydro-mechanical devices, such as Heron's weight regulator, we have about twice as many types (e.g., position, velocity, force, mass, and so forth). Suppose an interaction involves no more than three variables. Then, taking into consideration only variable types, the number of possible types of interactions is the cube of this number – 216 for fluids, and 1728 for hydro-mechanical systems. In contrast the number of different component types provided in a technology is relatively small – at most a few dozen. Thus, in terms of types of variables related, only a small percentage of the possible interactions will be produced by available components individually.

As a consequence, to produce a desired interaction, getting the variables to relate usually requires using, not one, but a set of interactions that take us between each of the variables. For example, in the interaction topology of Figure 2.9, 10 of the 19 interactions are on the direct path between the three variables being related (the remainder being on side branches).

Inaccessibility of Variables Consider the second factor contributing to the complexity of interaction topologies – accessibility of variables. In FSM design, if we want a variable to be involved in an interaction, then we access that variable simply by attaching a wire. However, for continuous physical systems, often the variables appearing in a desired interaction are not directly accessible to components being added.

For example, in the punch bowl problem the variables in the interaction are H_v, H_b, and dH_b/dt. None of these are directly accessible to the available components – pipes, valves, faucets and containers. That is, none of the interactions produced by these components involve these heights; they interact with variables of other components only through pressure (P) and flow (Q).

Thus, to gain access to those variables he wants to affect, an inventor must exploit existing interactions. Then he must determine what interactions should be added such that they, together with these existing interactions, produce the desired interaction. This process is demonstrated by the second half of the design account for the punch bowl problem (Section 2.1). This part of the account begins with the statement:

> Given this refined goal you begin searching for ways of relating height change to height difference. None of the components you have available to add to the design will directly affect height or height change. ...

It ends with the conclusion that, using interactions produced by the bowl and vat, the desired interaction:

$$[H_v - H_b] = [dH_b/dt]$$

can be produced by adding:

$$[P_v - P_b] = [dQ_b/dt]$$

where P and Q are directly accessible to other components.

Relating this account to the topology in Figure 2.9, the three variables identified in the account – P_v, P_b and dQ_b/dt – are connected to new interactions in the figure. These are the three solid, vertical lines that cross the dotted line between new and old interactions. The punch bowl account verbally traces a path through interactions from each of H_v, H_b and dH_b/dt to P_v, P_b and dQ_b/dt, respectively. These interactions

correspond to edges in the upper half of the graph ("old interactions") that lie along a path between these variables.

We see from this graph that the need to use existing interactions to access variables further complicates the interaction topology. In the punch bowl example, 14 of the 19 interactions in the topology are existing interactions, and six of these interactions lie directly along the path between variables in the desired interaction. These 14 interactions must be incorporated into the design, since they are the only means, using the available components, to get at H_v, H_b and dH_b/dt.

Parasitic Effects The last factor is the parasitic effect of extraneous interactions produced by components. For FSM design the logic gates used produce exactly the interactions we want, and nothing more. In contrast, for continuous physical systems, often the interactions produced by primitive components are much more complex than the interactions we are trying to produce. And they have an influence on variables we do not want to affect.

The complexity of primitive component interactions results from a combination of several factors: the behavior model of a primitive component type usually consists of several interactions; each interaction normally involves several variables; and the interactions are undirected (i.e., information can flow between an interaction's variables in any direction). For example, consider the complex behavior of a fluid container. Equations describing the interactions produced by the components of a container are shown in Figure 2.10. The fluid container involves eleven interactions, each of which is undirected and involves between one and three variables. Taken together these interactions relate nine variables and three constants.

While not all components are this complex, most of them produce extraneous interactions, and in some domains all the primitive components used are quite complex. For example, most digital circuits today are built in a CMOS or nMOS technology. For these designs the only components available are different types of MOS transis-

Model for a fluid container C:

$$
\begin{aligned}
V_c(t) &= Hd_{surface(c),bottom(c)}(t) \times Area_c \\
Pd_{bottom(c),top(c)}(t) &= fluid\_density_c \times gravity \times Hd_{surface(c),bottom(c)}(t) \\
F_{bottom(c)}(t) &= M\_fluid_c(t) - M\_empty_c \\
F_{bottom(c)}(t) &= M\_fluid_c(t) \times gravity \\
deriv(V_c(t),t) &= Q_{top(c)}(t) + Q_{bottom(c)}(t) \\
0 &= Q_{top(c)}(t) + Q_{bottom(c)}(t) + Q_{air(c)}(t) \\
deriv(V_c(t),t) &= Q_{top(c)}(t) + Q_{bottom(c)}(t) \\
M\_fluid_c(t) &= fluid\_density_c \times V_c(t) \\
[Area_c] &= [+] \\
[M\_empty_c] &= [+] \\
[fluid\_density_c &= [+]
\end{aligned}
$$

Figure 2.10: Fluid containers produce a complex set of interactions. The equations shown for the container are required for the examples explored in this thesis, such as Philon's lamp.

tors. The interactions produced by these components are even more complex than those for the container (Figure 2.10).

The major consequence of these extraneous interactions is their parasitic effect on the device's overall behavior, by weakening or undoing the effect that other interactions were selected to achieve. This parasitic affect can be widespread. Unlike digital systems, the interactions in a continuous system are undirected – effects propagate in either direction along an interaction. In addition, most continuous systems tend to include many positive and negative feedback loops. The behaviors of variables involved in a feedback loop are mutually dependent. The result of these two factors – undirected interactions and loops – is that each interaction has an indirect influence on the behavior of many variables in a device.

What makes constructing a topology of interactions difficult is that, to ensure correct behavior, the broad influence of extraneous interactions must be counteracted. This is accomplished by canceling out or at least tempering the effect of the extraneous interactions so that they don't overwhelm the desired interactions.

Often the designer must be satisfied with simply tempering the effect of the extraneous interactions, in which case the extraneous interactions degrade the desired performance. To produce the desired behavior, the designer is then faced with the problem of orchestrating a set of competing interactions.

An example of a quantity shifting device that must orchestrate competing interactions is the MOS inverter. The inverter raises and lowers its output by "filling up" or "emptying out" charge from a capacitor. One interaction – the pullup – adds charge to the capacitor, while a second interaction – the pulldown – removes charge. Unless "turned off," the pulldown interaction is parasitic when trying to raise the output, since it is trying to drop the output; conversely, the pullup interaction is parasitic when trying to lower the output.

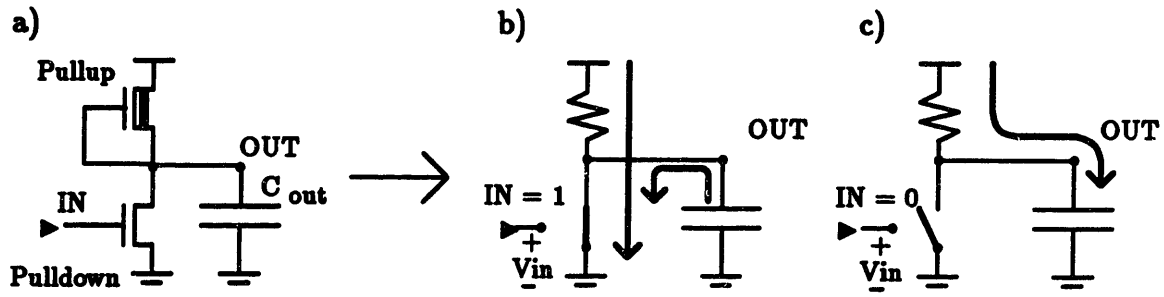


Figure 2.11: a) MOS Inverter consisting of a pullup and a pulldown. b) and c) are simplified versions where the pullup is viewed as a resistor and the pulldown is viewed as a voltage controlled switch. b) shows discharging and c) shows charging.

For many inverter designs only the pulldown interaction is turned on and off; the pullup interaction is left unchanged. When the pullup interaction is desired, the pulldown interaction is turned off. But, when the pulldown interaction is desired, the pullup is left on, and the pulldown is strengthened so that it overwhelms the pullup interaction. The result is a design that is simpler structurally – no direct control of the pullup interaction is necessary. However, the parasitic effect of the pullup degrades the inverter's performance when lowering the output.

Coupling Between Factors The three factors just discussed – mismatch of interactions, inaccessibility of variables, and parasitic interactions – tend to feed upon each other, making the design task all that much harder. For example, the initial sequence of interactions required is lengthy, because a set of interactions is needed to gain access to the desired variables, and then a second set is required to deal with the mismatch in variable types. In the punch bowl example, the first set is six interactions, and the second set is four interactions. Having this many interactions to start with exacerbates the problem by introducing a number of extraneous effects that must be eliminated. In the punch bowl example, roughly half of the interactions (9 of 19) are cancelling out extraneous variables. Finding additional interactions that eliminate these effects is made difficult by inaccessibility, mismatch, and the fact that they in

turn may introduce extraneous, and possibly parasitic effects.

Computational Complexity of the Task

Given the factors listed in the last section, constructing an interaction topology is not a clear cut task. We can further reinforce this point based on an informal complexity analysis. The task of constructing an interaction topology can be stated as the following question:

> Does there exist a set of interactions producible by physical structure that result in the desired interaction?

We take the "set of interactions producible" to be a subset of those interactions described by the first principles of physics (e.g., component models). Interactions are described by equations in our hybrid qualitative/quantitative algebra Q1 (Chapter 5). Q1 is composed of two algebras: the qualitative algebra is an algebra on the sign of quantities (+,0, −), and the quantitative algebra is the standard algebra on the reals.

Suppose we simplify the task substantially by assuming that the interactions are described purely by the sign algebra, and the set of producible interactions is finite. The design task is then equivalent to propositional satisfiability, and thus is NP-complete. [3] Thus we already have a task that is computationally expensive.

[3]More precisely, A set of interactions must be consistent to be a design solution, and determining that a set of qualitative equations is consistent (QSAT) is equivalent to propositional satisfiability (PSAT). The more subtle part of this proof is mapping PSAT to QSAT. To do this, given a set of propositional expressions, we make the following transformations to a set of sign equations: For each expression E we introduce an equation E = True. We treat each positive literal as a variable, and for each negative literal "not(P)" we replace it with the variable NOT-P. Then for each proposition P and its negation NOT-P, we introduce two equations: "P OR NOT-P = True," and "P AND NOT-P = False". Finally, we replace True with positive, False with zero, OR with sign addition, and AND with sign multiplication. The resulting sign equations are consistent exactly when the propositional expressions are satisfiable.

To be more realistic we need to relax the constraint that the producible interactions are finite. For example, the component models describe the interactions produced by any of an infinite sized class of components. To describe these interactions we must introduce quantifiers. with this augmentation the problem becomes equivalent to satisfiability for the first order predicate calculus – a semi-decidable problem.

Finally, taking into consideration that some interactions are described quantitatively, then we need to introduce the standard algebra on the reals. As a result we are faced with the consequences of Godel's incompleteness theorem.

Intuitions for Dealing with the Complexity

Given the analysis of the previous section, the design task does not lend itself to complete and efficient algorithms. Instead we consider some basic insights from AI into how to simplify the design task. Three approaches are briefly discussed: simplifying the search space, restricting the problem size, and changing the set of primitive devices. These approaches correspond to the three design strategies discussed in the introduction (Section 1.7). The objective of this section is to place these three strategies within the context of how they deal with the complexity of interaction topologies.

Two ways of simplifying the search space are through abstraction and approximation. Abstraction removes irrelevant details, taking a least commitment approach. While approximation removes details that are relevant but secondary, highlighting only those properties that most constrain the design choice.

We have already employed search space abstraction by using a hybrid qualitative algebra to describe and manipulate interactions. Expressions describing interactions at a more qualitative level tend to be much simpler than their quantitative equivalent.

This is primarily due to the elimination of variables that are considered constant from a qualitative viewpoint. For example, in the punch bowl example, if we are only interested in the sign of pressure difference and fluid flow then the equation:

$$(P_v - P_b)/(d \times g) = Q_b$$

simplifies to:

$$[P_v - P_b] = [Q_b]$$

since fluid density, d, and gravitational acceleration, g, are always positive.[4]

Search space approximation (removing relevant but secondary properties) is the basis of the strategy – tracing topologies of interaction – described in Chapter 3. A set of simplified interactions are explored first in order to construct a coarse solution. This solution is then refined, taking into consideration those details previously ignored. This strategy is the focus of this document.

A second strategy – called design evolution through focused innovation – takes the approach of restricting the size of the design problem. This strategy focuses invention from first principles on those subsets of a problem where innovation is crucial – subsets that are critical to the design's overall performance. These subproblems are identified by performing a form of critical path analysis, to determine parts of the interaction topology that are preventing the design from achieving higher performance. Ibis then reasons from first principles in order to make novel changes to this part of the interaction topology. Sophisticated designs evolve from simpler ones – by successively applying this strategy, the solution hill climbs towards ones with higher performance. This strategy was highlighted in Section 1.7 of the introduction, and will be discussed in more detail in [61].

[4]In addition, in Section 6.3.3 when analyzing the properties of $Q1$, we show that any sign expression is reducible to a quadratic formula. This is a significant compaction compared to the equivalent real expressions, which are reducible to rational form – a polynomial over a polynomial.

We are currently in the process of developing a third strategy – called *design from second principles* (Section 1.7) – that confronts the complexity of the task directly. Recall from above that, for continuous systems, interaction topologies are difficult to construct because the interaction topologies of primitive components are complex, and thus difficult to compose. As we observed at the beginning of Section 2.4.3, this is not the case for FSM design. The interaction topologies of primitive components for FSM design are simple and produce exactly the behaviors of primitive interactions – the boolean operations. The strategy then is to make constructing the interaction topology equally simple for continuous systems, by augmenting the set of primitive components with ones that produce primitive interactions. These include both the basic operators of the qualitative algebra Q1, and primitive quantity shifts.

These new components, called "second principles," are really composite structures, that are designed from first principles when a new technology is initially being explored. Eventually it is our intent that these will be developed by Ibis using the first strategy. The second principles are then used during invention whenever possible, falling back on first principles only when the second principles fail (e.g., when there is no second principle for the desired behavior, or none of the second principles satisfy the performance constraints).

This concludes our discussion of the difficulties that arise during the third transformation stage, when constructing the interaction topology. The remainder of this document focuses on the basic insights and technical details of the first strategy for constructing innovative devices from first principles – tracing interaction topologies.

2.5 Summary

- The invention of quantity shifting devices involves using many different types of interaction at different stages of the invention process.

- These types of interactions are characterized along several dimensions – qualitative or quantitative, extensional or intensional, static or dynamic, and causal or constraining. Each stage of invention alters one dimension of an interaction.
- The design of quantity shifting devices involves a series of transformations between different types of interactions, ending in a physical structure.
- The initial transformation stages are relatively simple, and involve a process analogous to digital finite state machine design.
- The difficulty is in the last stage where the complexity of physical components must be confronted directly. This stage is where a detailed interaction topology must be constructed that makes evident the topology of physical components.

Chapter 3

Tracing Paths of Interaction

This chapter presents the centerpiece of this thesis – a strategy for design from first principles alone that focuses the search on innovative alternatives.

As we discussed in the thesis introduction (Section 1.7), the strategy – called *tracing topologies of interaction* – is based on the conjecture that an inventor constructs a new innovative device in a manner similar to how he understands existing inventions. To understand a device the inventor visualizes all the interactions produced by each component in the device, and identifies those interactions contributing to the desired behavior. Analogously, we conjecture that he might invent a device by visualizing all possible interactions producible in the current technology, and identifying those interactions contributing to the desired behavior.

At the center of this approach is a representation, called a *topology of potential interactions*, that allows Ibis to "visualize" the space of producible interactions. This representation focuses the search on innovative alternatives by:

- using the first principles to compactly describing all interactions producible by all components and all connections in the available technology.

- using a topology to compactly describe all possible ways these producible interactions can inter-relate.

- highlighting the connection between variables participating in each interaction, but ignoring the behavior the interactions produce. This emphasizes topological differences between interactions, allowing the design strategy to focus initially on innovative differences between potential solutions.

This space is far simpler to search than the real space of interactions, yet preserves the most constraining design properties.

To construct a device, Ibis first constructs a coarse solution by identifying paths along the topology of potential interactions between variables we want to interact. A coarse solution is an interaction topology consisting of the set of interactions along this path. The solution is then refined so that the combined behaviors of interactions along the path produce the behavior of the desired interaction. To construct the device, the models and laws corresponding to each interaction are used to identify the corresponding augmentations to physical structure.

The first part of this chapter develops the search space used, and the motivation for why it is a good abstraction. Next, the search strategy is presented, and the basic steps are demonstrated by solving the punch bowl example. Finally we demonstrate the approach on the design of the ancient hydro-mechanical devices explored in the thesis introduction, using a more extensive model of physics. Many of the technical details of the approach are contained in the second part of this thesis, which discusses the representation for how variables interact (Chapter 5), the first principles of physics used (Chapter 7), and the procedural skills used to construct the interaction topology (Chapter 8) and to show that it produces the desired behavior (Chapter 6).

3.1 Motivation for the Approximate Search Space

The approach, taken here, of first constructing a coarse solution in an approximate search space, and then refining the solution, is beneficial if the approximate space:

1. preserves design properties that substantially constrain the solution, and
2. is much easier to explore than the original space.

The former is necessary in order for the approximate search space to have discriminatory power, and the latter is necessary in order for the space to provide a computational saving.

In this section, to identify how the search space should be approximated, we first characterize the properties of the devices being invented. Next we identify the most constraining properties of this invention task. Finally we show that, if the search focuses only on these properties, then a substantial cost is eliminated.

3.1.1 Properties of the Devices Being Invented

Recall that our goal is to be able to invent, from first principles of physics, devices at a level of complexity similar to the ancient fluid devices described at the beginning of this thesis. The behavior that these devices are designed to achieve is relatively simple. They are similar to the desired behavior for the punch bowl problem, and can be described by one or two interactions. In addition, each device typically involves a small number of components – often three or four and rarely more than half a dozen. Most devices containing a larger number of components appear to have evolved from devices developed by other inventors [25].

Given these observations, we target our strategy towards inventing devices that

achieve one or two desired interactions with no more than about a half a dozen components. Tasks of greater complexity are handled using the evolutionary design strategy, discussed in the introduction (Section 1.7), to focus the application of first principles on subproblems of this size.

3.1.2 The Most Constraining Properties

Although the behavior these devices are designed to achieve is simple, and they involve a small number of components, this is not to say that how the devices work is simple. As we saw in Section 2.4.3, the topologies of interactions produced by devices tend to be quite complex. Furthermore, constructing this topology is a subtle and difficult process.

Our representation should highlight what makes this process difficult, while suppressing other details. Recall from Section 2.4.3 that three factors contribute to this difficulty. The first is that there is a mismatch between the desired interaction and the interactions produced by available components. More specifically, there is a mismatch in types of the variables being related (e.g., height versus pressure). Part of the inventor's task is to identify a set of interactions that relate the right set of variables.

The second factor is inaccessibility – often the variables we want to relate cannot be connected to directly by new interactions, but instead must be accessed through existing interactions. Part of the inventor's task is to identify what existing interactions and variables provide access to the variables he wants to influence.

The third factor is that devices produce a much larger set of interactions than we want. As a result these additional interactions can have a parasitic effect on the remainder of the device. Part of the inventor's task is to cancel out these parasitic effects, and to avoid having them influence other variables whose behavior we are trying to control.

For each of these factors, the difficulty is explicit in what variables interact, independent of how they interact. For example, for the first factor we said the mismatch is in the variable types. Thus it is difficult to find a set of interactions that relate a particular set of variables at all, independent of what relationship is produced. For the second factor – inaccessibility – we are interested in any sets of interactions that will allow us to get at the variables we want to influence, independent of the behavior these interactions produce. Finally, for the third factor, we want to prevent the extraneous interactions from influencing other important variables, independent of what their behavior is. Thus, *the types of the variables related by each interaction substantially constrain whether that interaction is part of a solution to a design problem.*

This observation is central to developing our invention strategy. The invention task is to construct an interaction topology that produces a desired behavior. This observation suggests that we construct a coarse solution by looking for a set of interactions that relate the variables specified in the desired interaction, and ignore the specific behavior these interactions produce. To construct these solutions we use an approximate search space that focuses on the variables related by each interaction, ignoring the interaction's behavior. For example, given the interactions for a pipe:

$$\begin{aligned} Pd_{t1,t2} &= R \times Q_{t1} \\ 0 &= Q_{t1} + Q_{t2} \\ R &= [+] \end{aligned}$$

We preserve only the facts that the first interaction relates $Pd_{t1,t2}, R$ and Q_{t1}, the second relates Q_{t1} and Q_{t2}, and the third relates R to the constant [+].

Section 3.2 shows how these approximated interactions can be represented as a graph, where the vertices are variables, and each edge connects the set of variables

related in a particular interaction. Given this graph we identify a coarse solution by tracing a path between the variables participating in the desired interaction. The coarse solution is simply the set of interactions along this path.

3.1.3 The Eliminated Cost

To show that this approximate search space is useful, the remaining step is to argue that the approximation eliminates a substantial cost during search. To construct a completed solution, we must identify a set of interactions whose aggregate behavior produces the desired interaction. Determining that this is produced requires an extensive sequence of qualitative algebraic manipulations. These manipulations are demonstrated later in this chapter, in Section 3.5.4. Each of these manipulations is potentially quite costly – in Section 6.4, we show that in the worst case it is exponential in the size of the equations being combined. In sharp contrast, searching the approximate space is linear in the size of the graph. The saving is accumulated in any place where, during the exploration of the complete search space, algebraic manipulations would be performed on a set of interactions that don't relate the variables in the desired interaction. Even though the worst case complexity for algebraic manipulation is rarely achieved, the minimal cost of exploring the approximate space results in a substantial saving.

To summarize, the basic intuition behind our design strategy and the approximate search space is:

Focus on what can interact before worrying about how they interact.

Given this intuition we must elaborate on three points: First, how do we represent *what* interacts, while suppressing *how* they interact. Second, using this simplification, how do we construct a graph that captures the space of all producible interactions.

Third, how is this graph used to explore innovative alternatives. These points are the focus of the next three sections.

3.2 Representing What Interacts

We begin by using the punch bowl example to make more concrete the representation of *what* interacts. To make the presentation simpler, we begin with a technology consisting only of components used in the problem solution (containers and pipes), and simple models for each component. Section 3.7 introduces a more robust set of components and models, using them to construct a variety of alternative solutions to the punch bowl example.

3.2.1 Background: Describing Structure

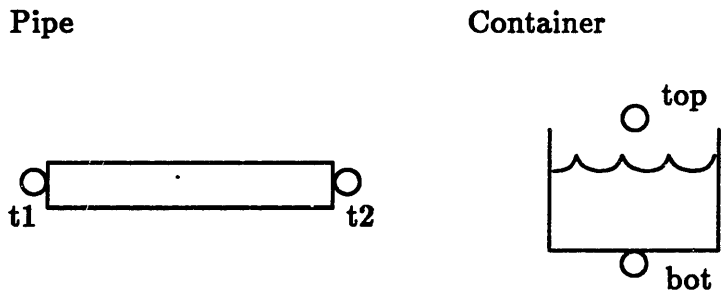


Figure 3.1: Examples of components and terminals. To the left is a pipe with two terminals, $t1$ and $t2$. Terminals are denoted by open circles. To the right is a container with two terminals, *top* and *bot*.

First we must introduce some vocabulary for our representation of structure. Recall that structure is represented as components and connections, where components communicate between each other through connections. Each component has a set of *terminals*, providing points for connection. For example, a pipe is a component with

two terminals, called $t1$ and $t2$, and a container is a component with two terminals, called *top* and *bot* (for bottom) (Figure 3.1). Components are connected together by connecting their terminals to a common point, called a *node* (Figure 3.2). We refer to the node that a terminal t is connected to as node_of(t).

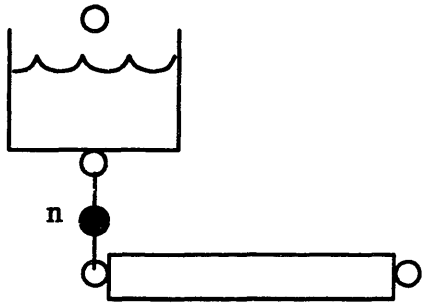


Figure 3.2: Example of connection between two components through a shared node, n. Nodes are denoted by closed circles. A connection is denoted by a line from a terminal to a node.

In the fluid domain each node has an associated pressure P, and each terminal has an associated fluid flow Q. That is, a terminal is a point where fluid can flow into a component, and a node is a point where the fluid flows of several components are combined. For example, in the punch bowl problem, fluid can flow into the vat (v) through its top or bottom. To capture this we model the vat as a component with two terminals, top(v) and bot(v).

3.2.2 Pictorial Descriptions of Interactions

Consider how we represent what interacts for the punch bowl example. The punch bowl problem involves a vat and bowl. The set of interactions produced by a vat are:

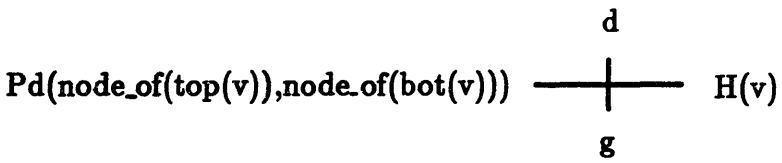
$$
\begin{aligned}
Pd_{node\_of(top(v)),node\_of(bot(v))} &= d \times g \times H_v \\
dV_v/dt &= dH_v/dt \times A_v \\
dV_v/dt &= Q_{bot(v)} + Q_{top(v)} \\
A_v &= [+] \\
d &= [+] \\
g &= [+]
\end{aligned}
$$

Here H denotes height of fluid, Pd pressure difference, Q fluid flow, A cross-sectional area and V volume of fluid contained. d and g are constants denoting density of punch and gravity.

We represent graphically the fact that a set of variables participate in an interaction by a single line connecting each of the variables. For example the interaction,

$$Pd_{node\_of(top(v)),node\_of(bot(v))} = d \times g \times H_v$$

is represented by the connection shown below:



A connection preserves the information about which variables interact directly, but suppresses information about how they interact. For example, the connection shown above says that the two variables, $Pd_{node\_of(top(v)),node\_of(bot(v))}$ and H_v, and the two constants, d and g, are related directly through an interaction.

The complete set of interactions for a component are represented by combining

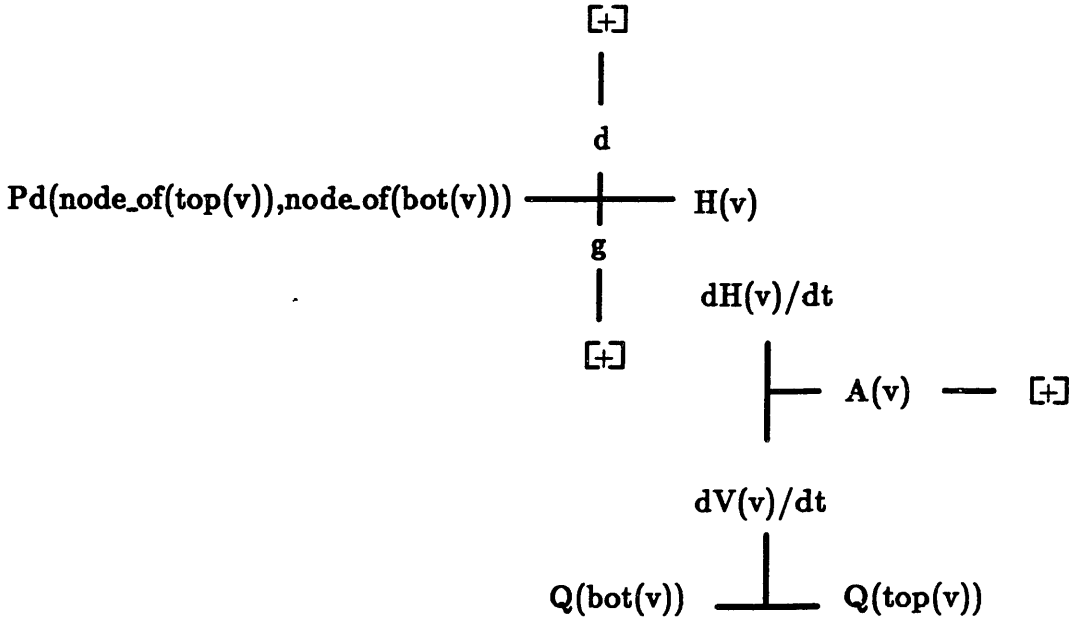


Figure 3.3: Topology of interactions for the vat highlighting what interacts.

the connections for all its individual interactions into a graph, where the vertices of the graph are the variables and the edges are the connections. For example, the graph for a vat is shown in Figure 3.3. Consider some of the information contained in this graph. The graph contains six connections corresponding to the six vat interactions listed at the beginning of this section. The graph tells us, for example, that height difference, dH_v/dt, interacts indirectly with flow, $Q_{bot(v)}$, by way of volume change, dV_v/dt. This is expressed by two connections: one involving dH_v/dt and dV_v/dt, and the second involving dV_v/dt and $Q_{bot(v)}$. It is not necessary for the interactions of a component to relate all variables. In this example, the interactions produce two disjoint graphs, one involving the variables $dH_v/dt, A_v, dV_v/dt, Q_{bot(v)}$, and $Q_{top(v)}$, and the second involving the variables $Pd_{node\_of(top(v)),node\_of(bot(v))}, d, g$ and H_v.[1]

[1]In this example there is an additional relationship not expressed by the interactions. V_v and dV_v/dt are related since the second is the derivative of the first. This information has already been exploited earlier in the invention process during the first transformation stage (Section 2.4.1). Recall that in this stage we used the relationship between a quantity and its derivative to identify constraints that produce a desired quantity shift; this was accomplished by the Integration Rule.

3.2.3 Primitive Topologies for Models and Laws

The interactions produced by each type of primitive component and connection are provided by the *component models* and *connection laws* of physical system dynamics and circuit theory. We construct a similar graph, called a *primitive topology*, for each of the models and laws of the domain and technology being explored, and then use these graphs to build up graphs for a particular design problem. For the punch bowl problem we are interested in component models and connection laws for fluids. For the simple solution to this problem – connecting a pipe – we need models for pipes and containers (to model the vat and bowl). These are shown in Figure 3.4. Each model in this figure describes the names and types of the variables and constants associated with a particular component type, as well as a primitive topology describing what variables and constants interrelate. For example, the pipe model, on the right side of the figure, specifies that a pipe, pi, has three variables: $Q(t1(pi))$ and $Q(t2(pi))$ which are of type "fluid flow", and $Pd(npt1, npt2)$ which is of type "pressure difference." It also has one constant, $R(pi)$, which is of type "fluid resistance."[2]

In some cases the variable names get unwieldy, such as $Pd_{node\_of(t1(pi)),node\_of(t2(pi))}$. To simplify them, the "let" statements in the models assign names to subterms of the variables; for example, "let npt1(pi) be node_of(t1(pi)), npt2(pi) be node_of(t2(pi))". This allows the pressure difference to be abbreviated to $Pd_{npt1(pi),npt2(pi)}$.

A model is sometimes broken into several parts, denoting specializations. For example, there is a basic model for a container and three specializations: normal size container, very large container and open container. For a normal size container, change in fluid volume is equal to the net flow into the container through the top

This relationship provides no additional information that is useful for constructing an interaction topology.

[2]As is discussed later, in Section 3.3.2, information about types is used to determine which interactions can be connected together, that is, what variables can be constrained to be the same.

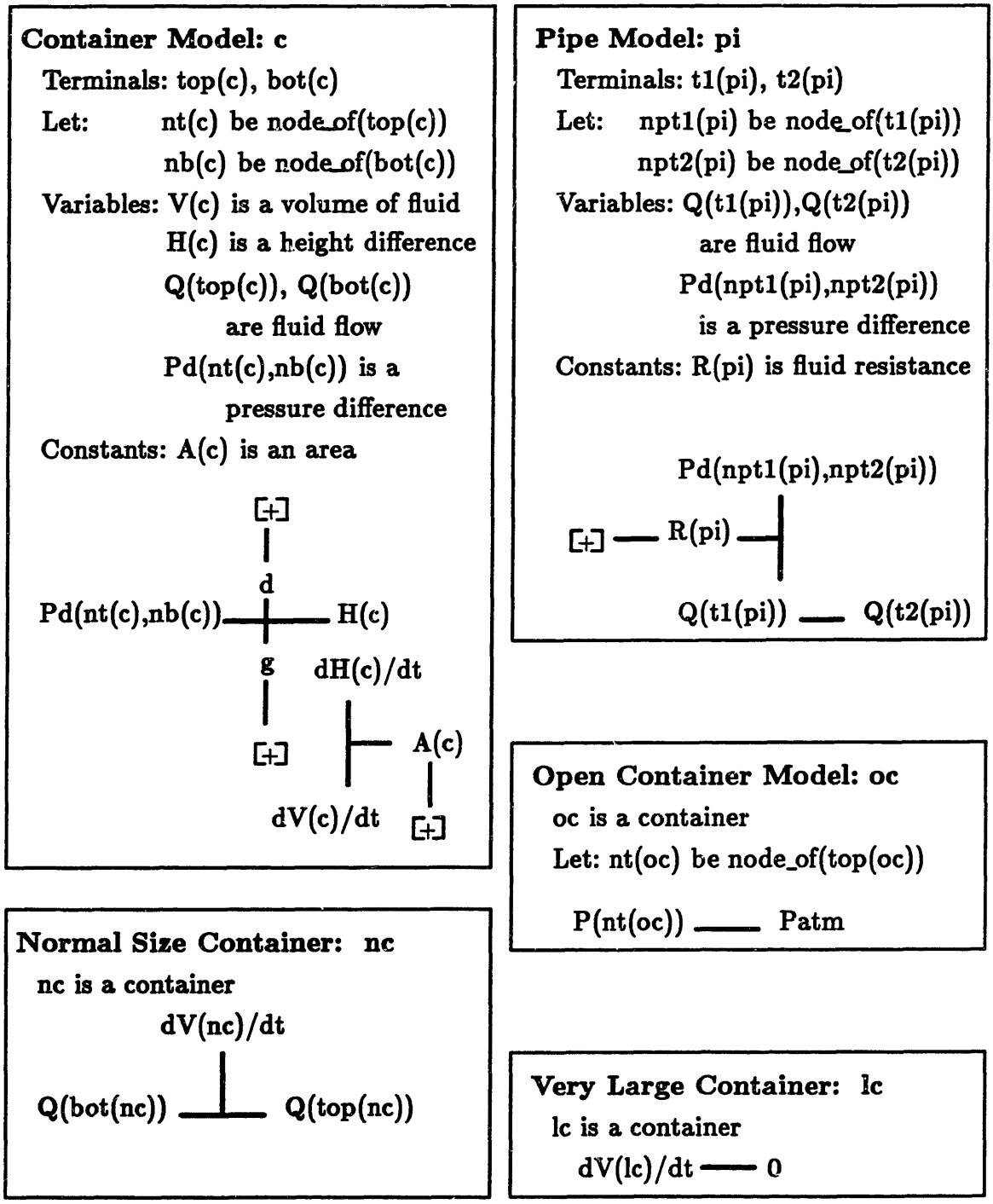


Figure 3.4: Primitive topologies for simplified models of fluids.

and bottom. A very large container is one whose volume of fluid changes negligibly over time. We use the idealization that the change in fluid volume is always 0. The vat in the punch bowl problem is an example of a very large container. An open container is one whose top is open to the air. In this case the pressure at the top of the container is at one atmosphere (P_{atm}). The vat and punch bowl are both examples of open containers. Primitive topologies for these three specializations are shown at the bottom of Figure 3.4.

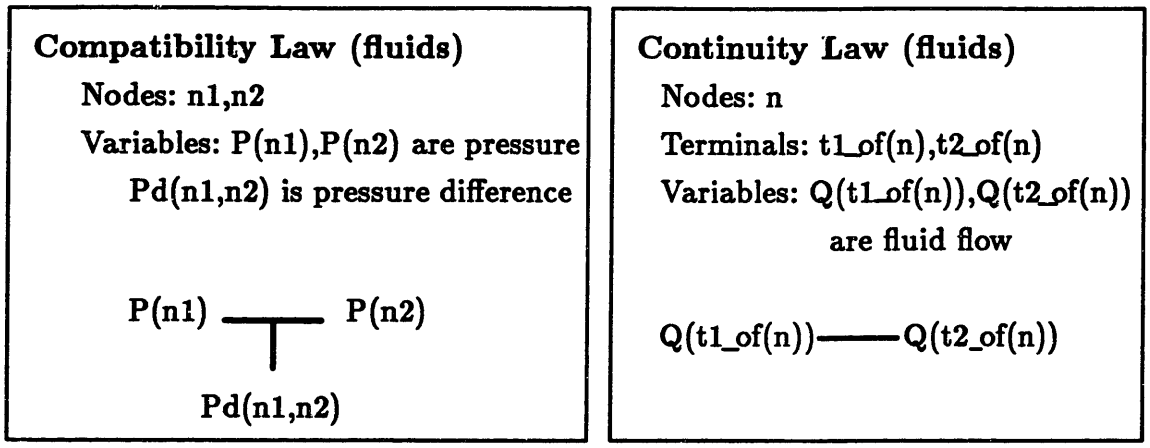


Figure 3.5: Topology of interactions for simplified laws of fluids.

Graphs for the two connection laws of fluids are shown in Figure 3.5. These laws characterize the interaction between components through the nodes and connections. For example, the graph for the compatibility law of fluids tells us that the pressures at two nodes, P_{n1} and P_{n2}, are related to pressure difference $Pd_{n1,n2}$ (the precise law is $Pd_{n1,n2} = P_{n1} - P_{n2}$). The graph for the continuity law of fluids tells us that the fluid flows, Q, of terminals connected to a common node are related (the precise law states that the flows sum to zero). In general any number of terminals can be connected to a node, but to simplify the example, we assume that a node, n, has two terminals connected to it. We refer to them as $t1\_of(n)$ and $t2\_of(n)$.

3.2.4 Using Primitive Topologies to Build Descriptions

To describe the interactions corresponding to a particular physical situation, we piece together primitive topologies corresponding to the device models and laws that apply to the components and connections of that situation. For example, consider the initial physical situation in the punch bowl problem, described at the beginning of Section 2.1. We are given that there is a vat of punch and a punch bowl, both of which are open to the air (i.e., the open container model applies). The punch bowl is a normal sized container, while the vat is a very large container – "its volume changes negligibly throughout the party". Finally, no unsightly connections are permitted to the tops of the containers; that is, $Q_{top(b)} = 0$ and $Q_{top(v)} = 0$.

The graph for this initial situation is represented by Figure 3.6. The interactions for the vat are on the left and those for the bowl are on the right. Moving from top to bottom on the left hand side, the subgraphs for the vat are composed of primitive topologies for the open container model, compatibility law, container model, and very large container model (Figure 3.4). The subgraph for the bowl is similar, the difference is that the primitive topology for the normal size container model is used, instead of that for a very large container.

Note that, to simplify this graph, we have used the abbreviations specified by the "let" statements in the models (Figure 3.4). For example, for container c (i.e., the vat and bowl) we abbreviated $node\_of(bot(c))$ by $nb(c)$ and $node\_of(top(c))$ by $nt(c)$.

A graph of interactions is used to determine if and how two variables interact; that is, to show that there exists a constraining relation between the variables. Specifically, two variables interact when there exists a path along the graph between the variables. For example, in Figure 3.6 fluid height in the bowl (H_b) interacts with flow out the bottom of the bowl $Q_{bot(b)}$; however, H_b does not interact with the height of fluid in the tank H_v. In the next chapter we use this representation to describe the space

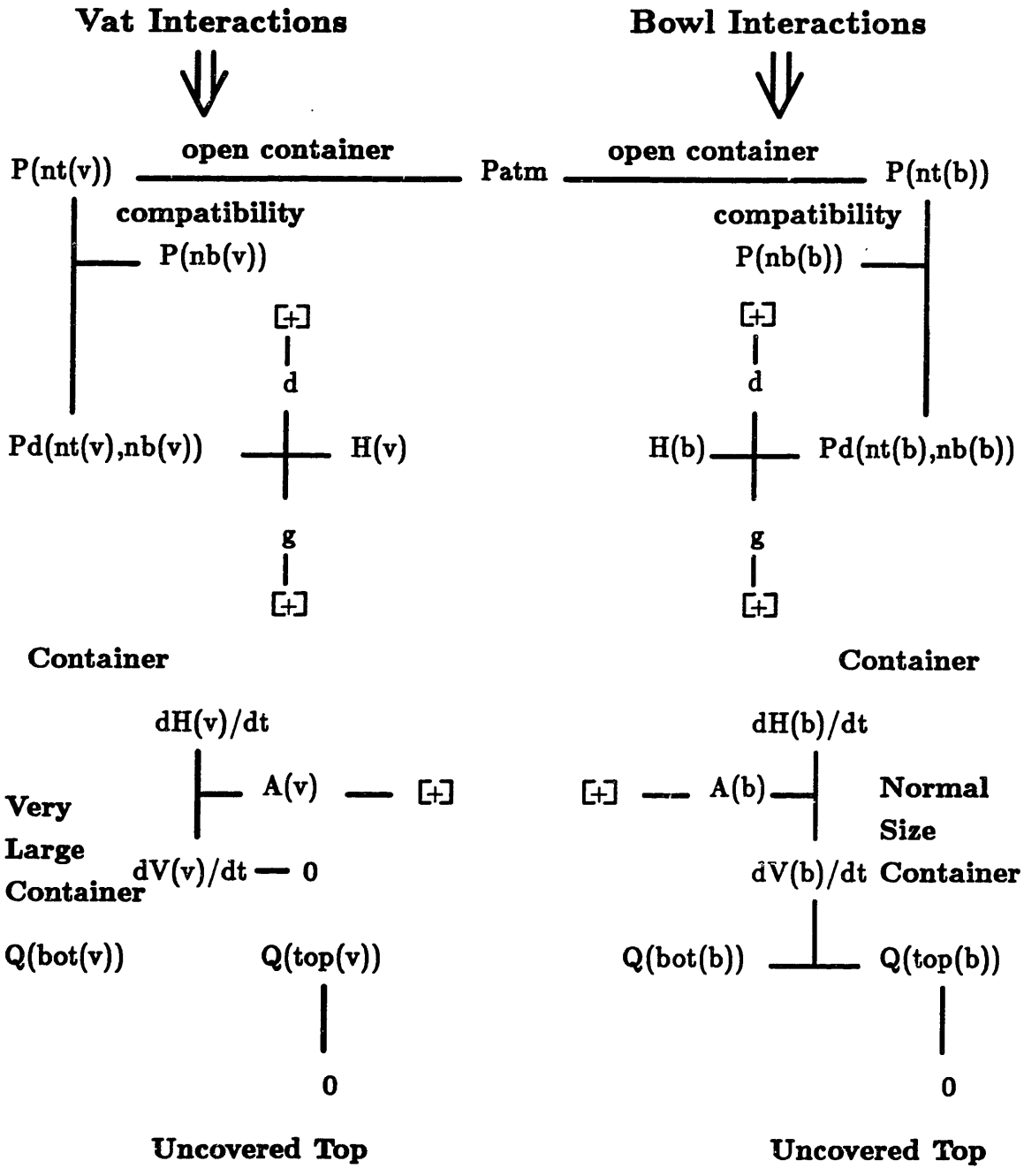


Figure 3.6: Topology of existing interactions for initial specification of punch bowl problem.

of all interactions producible, which is used to identify ways of making two variables interact.

3.3 Using a Topology to Represent the Space of All Interactions

Thus far we have made concrete the notion of a graph that captures which variables interact, while suppressing how they interact, and have argued that this type of graph is a useful approximation for invention. Next we develop a graph of this type that approximates the space of all interactions producible, and all ways these interactions interconnect for the current technology.

Recall that this graph is used to identify a first cut interaction topology for a design problem – a topology that relates the variables specified in a desired interaction, but may not produce the right behavior. This topology is identified by constructing a path through this graph between the variables being related, where the interactions in the topology are those that lie along the path.

How is this graph constructed? Recall that for a design problem, the desired interaction is constructed from a combination of existing interactions (e.g., interactions produced by the vat and bowl), and interactions producible through augmentations to physical structure (e.g., by adding a pipe). Thus the graph being searched must capture both types of interactions. To accomplish this we first construct separate graphs for existing interactions (called a *topology of existing interactions*) and interactions producible through augmentations (called a *topology of potential interactions*). We then compose these two into a graph representing the complete search space. The two graphs and their interconnection are discussed in the next three subsections.

3.3.1 The Topology of Existing Interactions

The topology of existing interactions is a graph of interactions produced by the components and connections of a completed or partially specified device. An example of this type of topology was shown for the punch bowl problem in Figure 3.6 of the last section. In the last section we also saw that this type of graph is constructed simply by instantiating for each component the primitive topologies of the appropriate component models and connection laws. The fact that the instantiation of these primitive topologies shares variables results in a larger, composite graph. This is all there is to the construction process.

An important remaining issue is to understand why the topology of existing interactions is central to the invention process. More specifically, why is it important to use existing interactions to build a desired interaction, rather than simply building it completely out of new interactions?

There are several reasons for this: First, an inventor typically minimizes the cost of a design by keeping down the number of components added. Thus, we want to exploit existing structure whenever possible. This may involve function sharing – using the same structure or interactions to achieve several different desired interactions.

Second, the existing structure places constraints on the design that the inventor should try to work around. As we argued at the end of the last chapter (Section 2.4.3), unless absolutely necessary we do not want newly added interactions to have a parasitic effect on those that already exist (e.g., pushing a signal towards the opposite direction from that desired). The topology of existing interactions helps to identify when a new interaction improperly influences, directly or indirectly, other quantities whose behavior we are already controlling.

The most important reason for using existing interactions is that, because of inaccessibility, a device often cannot be made to work without them. In Section 2.11

we discussed that often the variables we want to relate cannot be connected to directly by new interactions, but instead must be accessed through existing interactions. Part of Ibis' task is to identify what existing interactions and variables provide access to the variables it wants to influence.

For example, in Section 2.11 we showed that the three variables, H_v, H_b and dH_b/dt are not directly accessible, in the desired interaction for the punch bowl problem. However, the variables $P_{bot(v)}$, $P_{bot(b)}$ and $Q_{bot(b)}$ are directly accessible, and there are paths through the existing interactions between them and the three variables in the desired interaction. These paths are shown in Figure 3.7. In this figure the path at the upper left tells us that height of fluid in the vat H_v can be accessed through pressure at the bottom of the vat $P_{bot(v)}$. Analogously, the path at the upper right tells us that H_b can be accessed through $P_{bot(b)}$. Finally, the path at the bottom right tells us that change of fluid height for the bowl dH_b/dt is accessible through fluid flow into the bottom of the bowl $Q_{bot(b)}$.

Because the concept of *access* is important to our task, we introduce some useful terminology to refer to it. Given a desired interaction, we define an *access variable* to be a variable that is related to a variable in the desired interaction by existing interactions and can appear explicitly in a new interaction. For example, above we demonstrated that $P_{bot(v)}$ is an access variable of H_v, while $Q_{bot(b)}$ is an access variable for dH_b/dt. An *access path* is a minimal set of existing interactions that relates the access variable to one of the variables in a desired interaction, where each of the intervening quantities along the path is a variable. That is, the intervening quantities are not constants, since otherwise the access variable would have no affect on the variable we are trying to access. For example, the three paths of Figure 3.7, discussed above, are all the access paths for the punch bowl problem.

Consider the role of access in our invention strategy. Recall that to produce a desired behavior we construct a topology of interactions using both new and existing

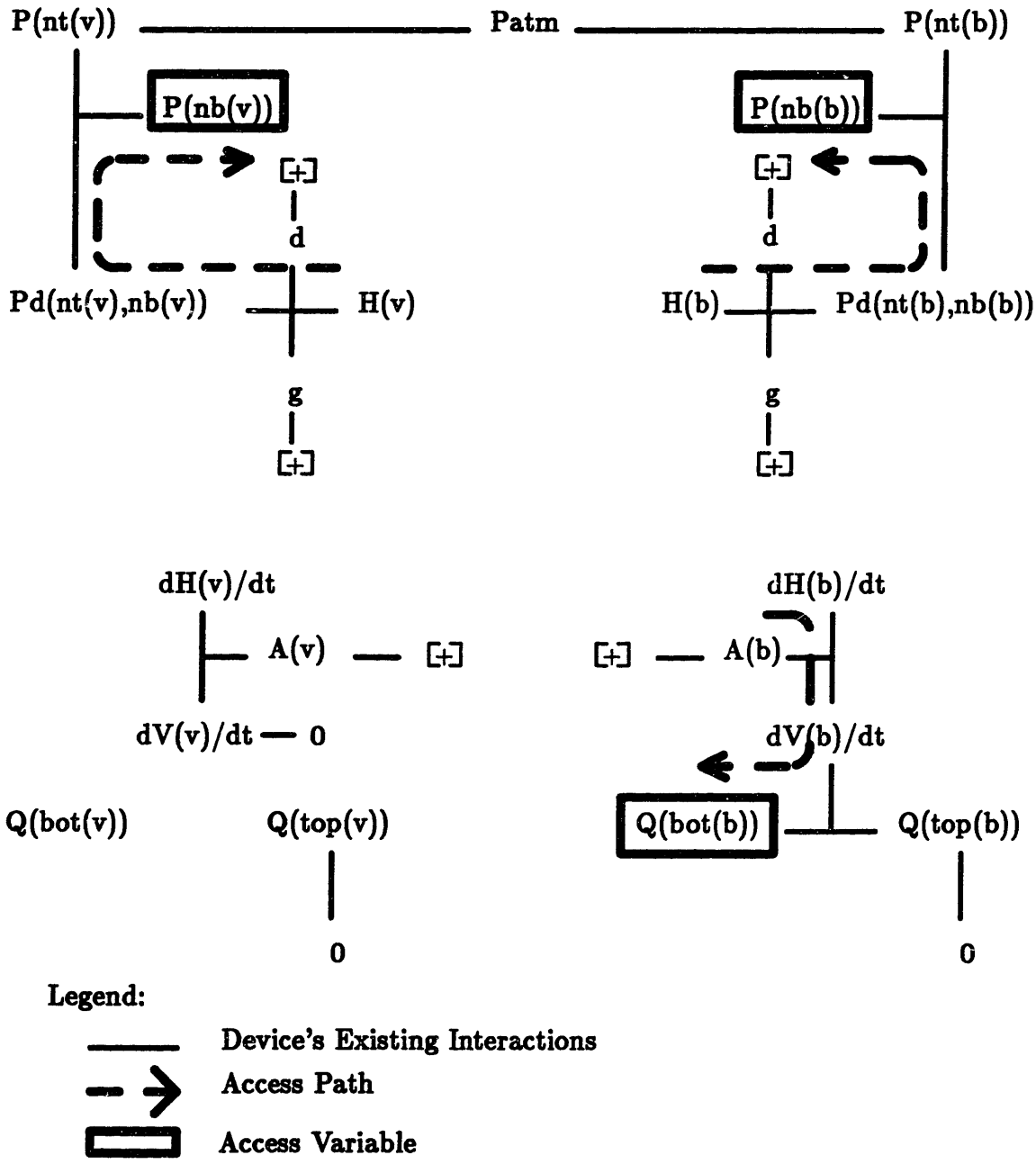


Figure 3.7: Paths through the topology of existing interactions for the punch bowl problem, used to access the variables in the desired interaction (H_v, H_b, and dH_b/dt).

interactions. Using the idea of access we split the construction process into two parts: In the first part we use existing interactions to relate each variable in the desired interaction to variables that can appear explicitly in new interactions (Figure 3.8). These latter variables are access variables, and the existing interactions used are the corresponding access paths. The access variables and paths are identified by tracing paths outward from the variables in the desired interaction to access variables.

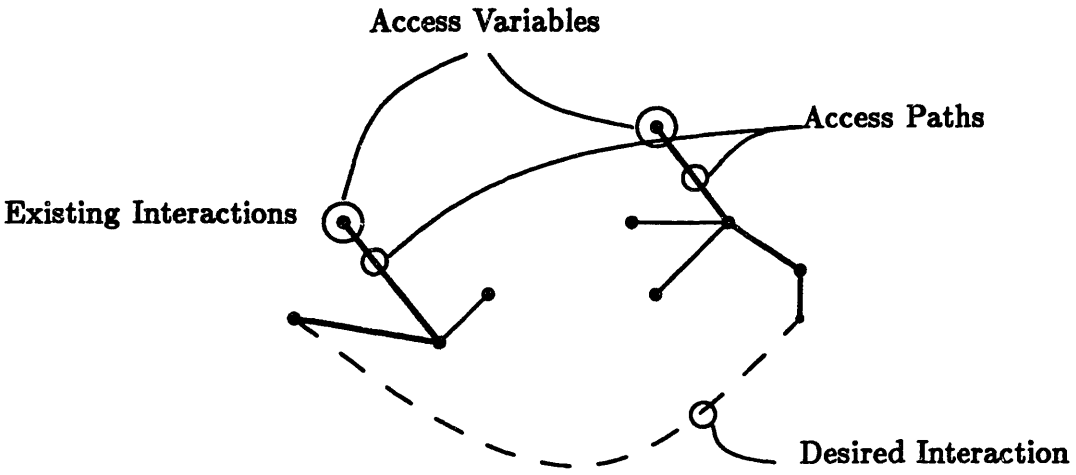


Figure 3.8: Identify access paths and access variables for desired interaction.

In the second part, we construct a set of new interactions between the access variables (Figure 3.9), such that the composite behavior of the new interactions, and those along the access paths, produce the desired interaction. How these new interactions are constructed is central to our invention strategy and is developed in the next few sections.

An example will make this more concrete. For the punch bowl problem, we have already performed the first step. Given the desired interaction between dH_b/dt, H_v and H_b, we have identified the corresponding access variables, $Q_{bot(b)}$, $P_{bot(v)}$, and $P_{bot(b)}$, and the corresponding access paths, shown in Figure 3.7. In the second step we construct new interactions that relate the three access variables ($Q_{bot(b)}$, $P_{bot(v)}$,

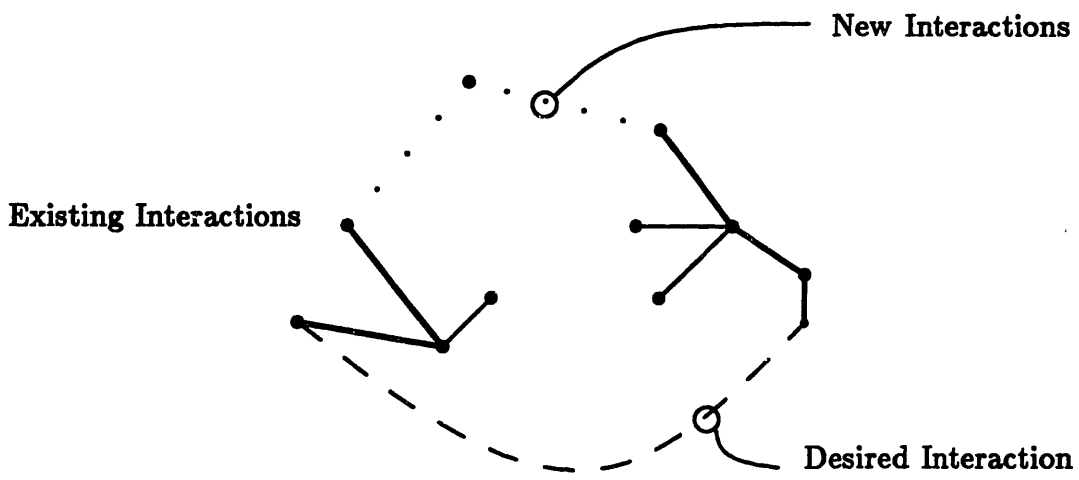


Figure 3.9: Build new interactions between access variables.

and $P_{bot(b)}$), for example by connecting a pipe. Figure 3.10 shows the new interactions produced by connecting the pipe, together with the existing interactions along the three access paths.

The topology of existing interactions describes part of the search space. The missing component is a description of all new interactions that can be added. This is the topic of the next section.

3.3.2 The Topology of Potential Interactions

As we observed in the introduction, invention would be easy if we had some way of looking at all the possible ways to add new interactions through augmentations to physical structure. Suppose we organized all these possible new interactions into a graph, which we call a *topology of potential interactions.* Then to identify new interactions used to relate a set of access variables, we would simply trace paths through this graph, much in the way we traced paths through the topology of existing interactions in the preceding section. This approach has the feature of focusing the

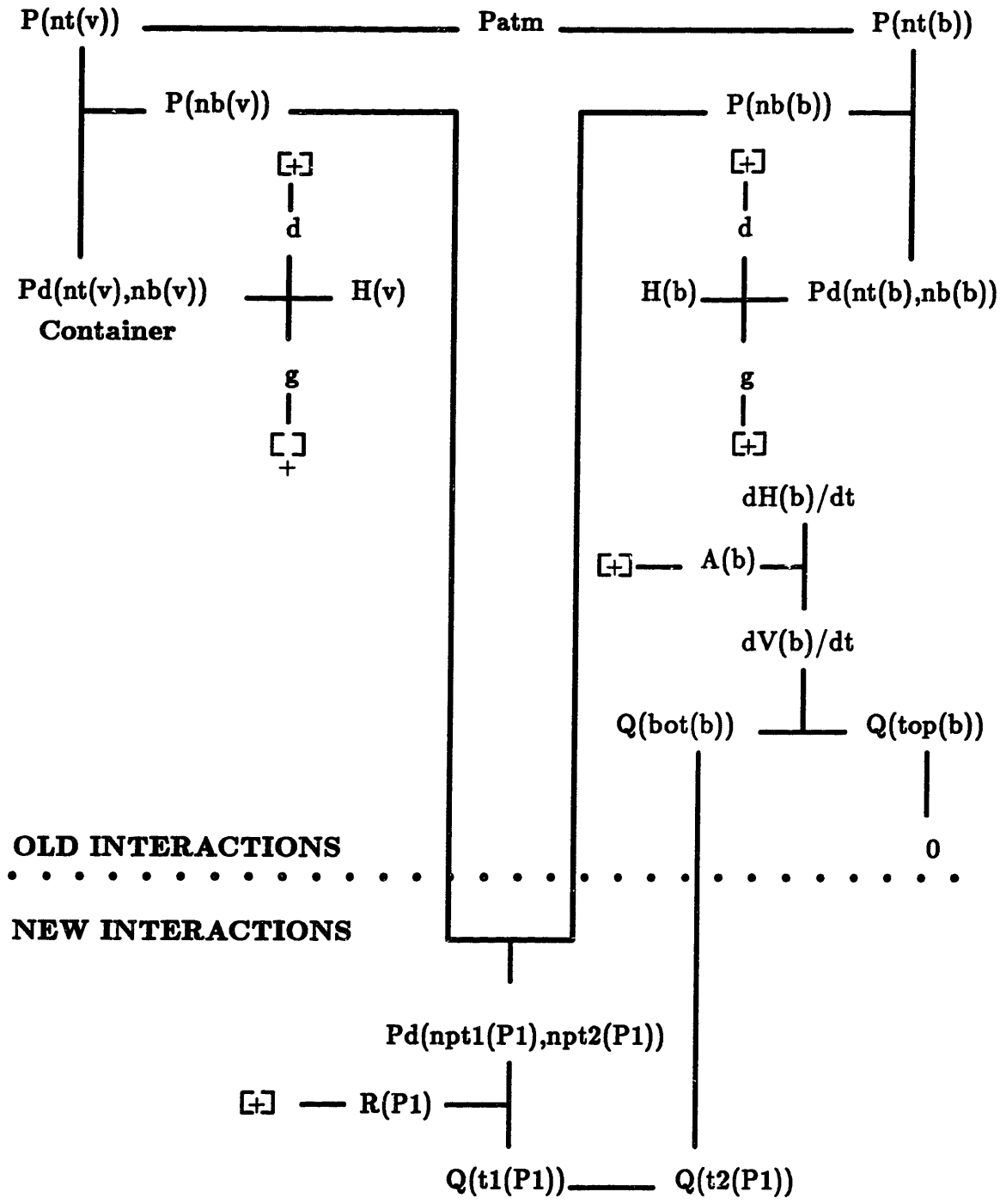


Figure 3.10: The desired behavior for the punch bowl problem is produced by new interactions produced by connecting a pipe, together with existing interactions along the three access paths.

design process on innovative alternatives, as distinct paths correspond to distinct interaction topologies, and thus to fundamental differences in how the device works.

In order to facilitate invention, the potential interaction topology must meet several requirements:

- it should reflect what is producible in the current domain (e.g., hydraulics) and technology (e.g., pipes and containers);
- it must be complete – that is, it must represent *all* interactions producible by *any* component or connection available in the current domain and technology, and *all* possible ways these interactions can interconnect;
- it must be relatively compact so it can be searched quickly;
- it must tell us what augmentations to make to physical structure in order to produce the interactions needed (e.g., get a pipe and attach it between the vat and bowl), and
- it must be easy to construct from first principles of physics.

The primary difficulty in constructing the potential interaction topology is that any graph that *explicitly* contains every interaction, and all ways they interconnect, is enormously too big. A device can contain any number of components or connections, and thus any number of interactions. To reduce this size, we need to create the right abstraction such that, although the graph doesn't explicitly contain every interaction, it does so implicitly.

The key to satisfying these needs, while overcoming this difficulty, is to construct the graph of potential interactions from the primitive topologies for the laws and models of the current domain and technology (e.g., Figures 3.4 and 3.5).

Design systems typically reason about component models and laws individually[29, 39,36,26,33,31], viewing them as a set of separate tools for augmenting a device's structure and behavior. *Instead we want to consider the models and laws together – in terms of the types of interactions they produce as an aggregate.*

To do this we construct the potential interaction topology by connecting together the individual model and law graphs *where it is at all possible for them to interact* – where two interactions can be made to share a variable. We refer to a connection between two variables that can possibly be shared (i.e., be made the same) as a *link*. Figure 3.11 shows the potential interaction topology for the punch bowl example, constructed from the primitive topologies of the fluid models and laws of Figures 3.4 and 3.5. We see from this graph that, in the fluid domain, the models and laws interact by sharing variables for pressure and fluid flow.

Consider some examples of what variables are or aren't linked, and the reasons why. At the bottom of Figure 3.11, $Q_{t1(pi)}$ in the pipe model is linked to $Q_{t1\_of(n)}$, since it is possible for $t1(pi)$ and $t1\_of(n)$ to refer to the same terminal, and thus for $Q_{t1(pi)}$ and $Q_{t1\_of(n)}$ to refer to the same flow. More specifically, the expressions $Q_{t1(pi)}$ and $Q_{t1\_of(n)}$ denote the same variable when $t1(pi)$ and $t1\_of(n)$ denote the same terminal (by definition each terminal has a unique flow Q), and the latter is the case when there is a physical connection from terminal $t1(pi)$ to node n.

In contrast, no flow variable (Q), is linked to height (H), pressure (P), volume (V), area (A) or resistance (R), since by definition each of these denote distinct classes of variables. Also notice that sometimes two occurrences of the same type of variable are not linked together. For example, $Q_{t1(pi)}$ of the pipe model is not linked to $Q_{bot(nc)}$ of the normal size container model. The reason is as follows: The two flows can only be shared if the two terminals, $t1(pi)$ and $bot(nc)$, are the same, since by definition the flows of distinct terminals are distinct. Likewise, these terminals are the same only if the components, nc and pi, are the same, since by definition the

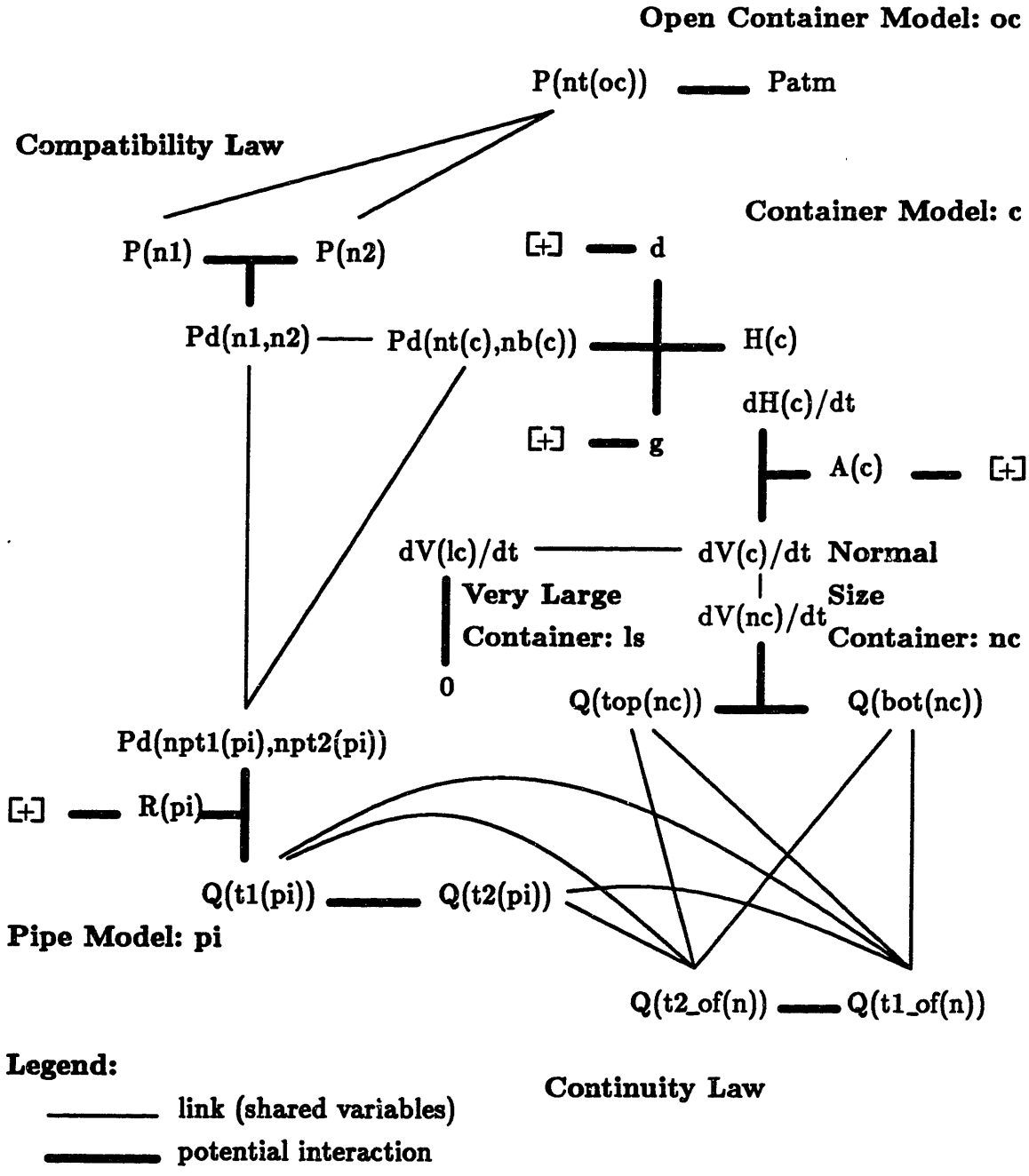


Figure 3.11: Topology of potential interactions for the simplified fluid models and laws.

terminals of distinct components are distinct. Finally, *nc* and *pi* denote a container and a pipe, and by definition containers and pipes are distinct classes of components – *nc* and *pi* are distinct. Therefore, $Q_{t1(pi)}$ and $Q_{bot(nc)}$ must be distinct, and thus are not linked.

In general, to determine which variables can be linked, our invention system Ibis must reason about the relationship between the types of objects contained in the interactions specified by the laws and models (e.g., variables, components, nodes and terminals). These relationships are imposed by the definitions of the technical terminology used in our physics. These relations include:

- the decomposition of objects representing structure into distinct classes (e.g., nodes, terminals and components of different types).
- The relationships between these structural objects (e.g., the terminals of one component are distinct from those of another).
- The distinct kinds of variables used (e.g., H, V, P, Q, A and R).
- And their relationship to the structural objects (e.g., associated with each terminal is a single unique flow variable (Q), and with each node a single unique pressure variable (P)).

In the AI community, reasoning about these types of relationships – which hold by definition of the class of objects involved – is called *terminological reasoning.* In our case we are using these definitions to build interaction topologies by connecting individual interactions. In addition, in Section 3.5.3 we use a similar type or reasoning to identify connections between physical parts, that produce connections between interactions. We refer to the use of definitions to perform this type of reasoning as *constructive reasoning.* Ibis uses a novel constructive reasoning component, called *Iota*, to link variables, both for the topologies of potential and existing interactions.

The novelty of Iota has to do with the task it performs (construction), the expressivity of its definitional language, its completeness, and its guaranteed efficiency. These contributions were summarized in Section 1.9 of the introduction, and the technical details of Iota are discussed in Chapter 8.

Reviewing what we have accomplished, the potential interaction topology satisfies each of the items on our wish list, specified at the beginning of this section – it reflects what is producible in the current domain and technology, it is complete, compact, makes physical augmentations evident, and is constructed from first principles.

First, the potential interaction topology reflects the set of interactions producible in the current domain and technology. For each type of structural augmentation – component or connection – that can be made in a particular technology, models and laws are provided specifying all the interactions they produce. Each interaction in the potential interaction topology is part of a primitive topology for one of these models and laws, and thus is producible in the current domain and technology.

Second, the potential interaction topology is complete. This means that it describes all interactions producible, and all connections between these interactions. To the extent that the models and laws supplied to Ibis are accurate, the first is satisfied. Connections between interactions are identified by Iota. Given that Iota is logically complete, then it will identify all such connections.

Third, the potential interaction topology is compact. We argue this in three steps: First, the topology is constructed from the primitive topologies of the models and laws. Each model and law describes the interactions produced by all components or connections of a particular type. Thus, the size of the topology corresponds to the number of types, not the number of components and connections.

Second, the number of component and connection types in a domain is typically small. For example, in the integrated circuit domain the number of component types

is kept under three, so that the fabrication process can be optimized to those types, and only one type of connection is allowed – electrical connections. For discrete components the number of primitive component types is kept down in order to reduce manufacturing costs and excess stock.

Third, for any one component type the primitive interaction topology is relatively small – typically no more than a dozen interactions. Again the models are for primitive components, which tend to have the simplest behavior. Although we argued at the end of the last chapter that they are more complex than desired, there is an effort, by engineers designing the primitive components, to make them as simple as possible, so that they are easier for other designers to use.

Summarizing the three step argument for compactness – we have moved from instances to types, the number of types is small, and for any one type the interaction topology is small. As a consequence of these three factors, the topology of potential interactions will be small. The needed compaction of the search space is achieved by using the models and laws to represent classes of interactions, rather than individual interactions.

The topology also satisfies the last item on our list – it makes evident the physical augmentations required. The models and laws provide the link between each interaction and a component or connection that produces it. For example, the fluid compatibility law tells us that, if the terminals of two components are connected to the same point, then the pressure at the two terminals will be equal.

We use this link to determine exactly what augmentations must be made to a device's physical structure. Recall that, given a desired interaction, Ibis constructs a candidate solution, consisting of interactions along a path through the potential interaction topology. Roughly speaking, to identify the necessary augmentations to physical structure, Ibis examines the models and laws that each of these interactions participates in – if the interaction is part of a model, then we add a component of the

type that the model describes; if the interaction is part of a law then a connection is added. We elaborate on this point in Section 3.5, once we have a concrete example to work with.

Finally, the graph is constructed only from first principles of physics. The only knowledge of physics used are the component models and connection laws, used to identify producible interactions. These models and laws constitute the first principles of our physics – physical system dynamics and circuit theory. Thus the last item on our list is satisfied.

3.3.3 Constructing the Complete Space of Interactions

Recall that the purpose of the topologies, developed in the last two sections, is to construct a graph that shows the combined effect of both existing interactions and *potential interactions producible by any possible augmentation to the physical structure.*

This graph is built by connecting together the topologies of existing and potential interactions discussed in the last two sections. The topology of existing interactions describes all interactions produced by existing physical structure, and how they interconnect (e.g., Figure 3.6). The topology of potential interactions is an abstraction that compactly describes the complete set of interactions producible through physical augmentations, and the ways they interconnect (e.g., Figure 3.11). So we have descriptions containing all of the two types of interactions we are interested in, and all interconnections between interactions of the same type.

What remains is to describe the interconnection between existing and potential interactions. To accomplish this we link the two topologies where there is a variable shared by a potential and an existing interaction – these are the access variables. Shared variables are identified using Iota (Chapter 8), just as they were in the last

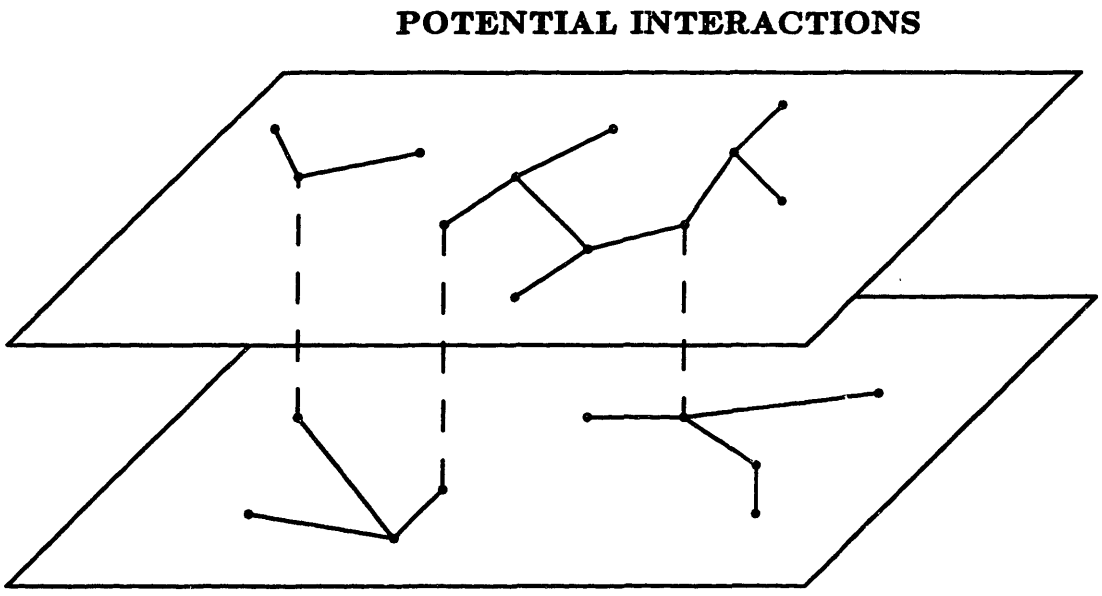


Figure 3.12: Interrelating existing and potential interactions.

section when constructing the topology of potential interactions. The combined graph is shown in Figure 3.12.

To summarize, the initial goal was to represent a compact description of the search space that highlights its salient features – a graph that allows us to visualize all the possible ways to add new interactions, through augmentations to physical structure. We have constructed such a graph by combining topologies containing all existing and potential interactions. This graph satisfies the five desiderata: reflecting what is producible, completeness, compactness, making physical augmentations evident, and being constructed from first principles alone.

Finally, by organizing the search space as a topology, it highlights innovations. And by constructing this topology from first principles, it provides the key to making good on the claim that we are doing innovative design from first principles.

3.4 Invention as Augmenting Interaction Topologies

At this point we have all the elements necessary to describe our invention strategy. Recall that the task is, using only a knowledge of first principles of physics, to construct a topology of interactions that produces a desired behavior, and to construct a physical structure that produces it.

Also recall that the approach being explored is to simplify the search, by initially focusing on making a set of variables interact without worrying about how they interact. Finally, recall from the introduction (Section 1.7), that the intuition behind the design strategy is: First, we visualize a topology that describes all interactions – both existing interactions and any interactions producible by any component or connection in the available technology – and all ways they interconnect. The graph

developed in the last section is such a topology. Then to propose a solution, we identify those interactions that contribute to the desired behavior.

Given these intuitions, our strategy consists of two parts: First we construct a coarse solution – a set of augmentations to physical structure, whose resulting interactions establish a path between the variables in the desired interaction. That is, initially we have a partial path in the existing interactions, but there is a gap. Our search is for augmentations to physical structure that produce interactions that bridge the gap, thus causing these variables to interact. During this construction the behavior produced by the interactions is ignored. In the second part we check whether the coarse solution produces the behavior of the desired interaction, refining the solution when necessary.

Given the graph of potential and existing interactions, developed in the last section, the strategy for constructing a device involves four steps: First, identifying new interactions to be added, by tracing a path; second, instantiating the solution, by augmenting the device's interaction topology and physical structure, and, third, testing that this is a valid solution, and refining it when necessary.

First consider how Ibis identifies the new interactions to be added. Given a desired interaction Ibis identifies the new interactions required by drawing a path through the combined graph between the variables in the desired interaction (Figure 3.13). Ibis traces a path outward from each variable in the desired interaction, along the existing interaction topology to the access variables. It then crosses over at the access variables to the potential interaction topology, and traces a path in this second topology that completes the connection between the variables in the desired interaction.[3] Each path provides a candidate for a set of interactions to be added, and each path corresponds to a distinct interaction topology. The interactions to be added are those that lie in the potential interaction topology.

[3]More generally, a path may cross between the two topologies many times.

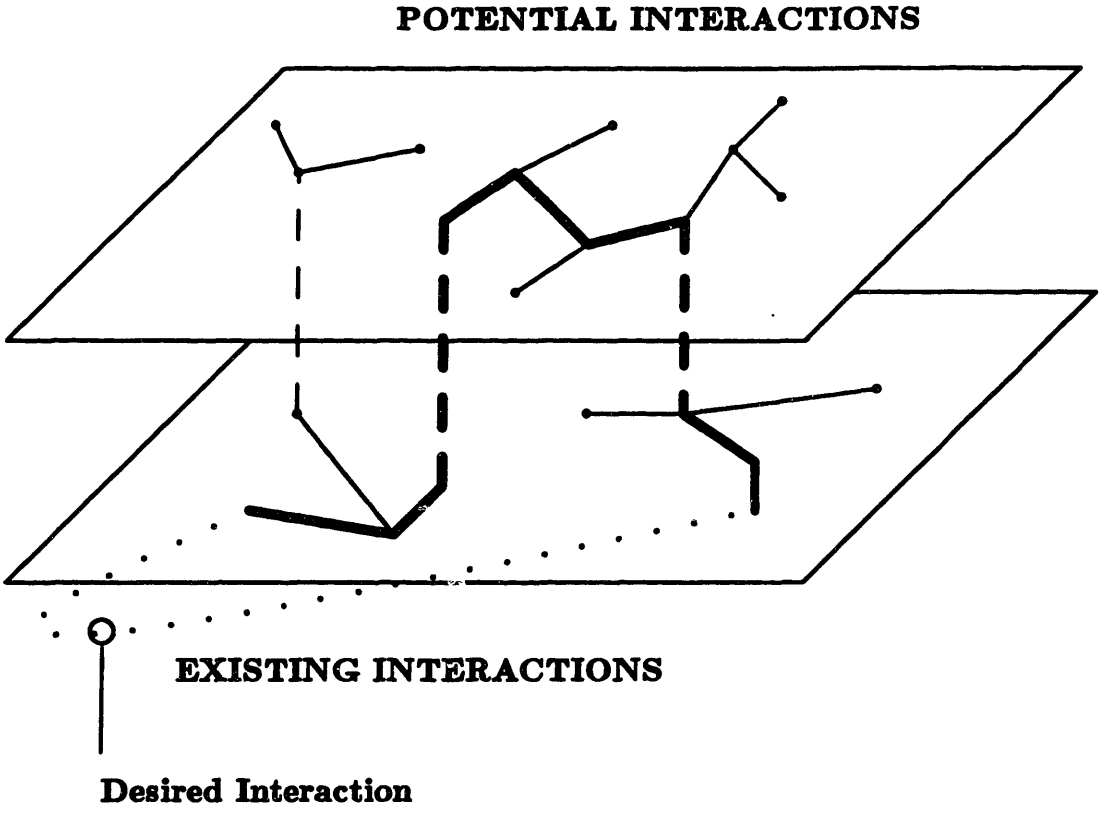


Figure 3.13: Identifying needed interactions by tracing paths along existing and potential interactions.

Next Ibis instantiates this solution. The topology of existing interactions is augmented by instantiating the potential interactions lying along the path just traced, and connecting them together wherever a link was crossed. This is accomplished by augmenting the device's physical structure with components and connections that produce these interactions (Figure 3.14).

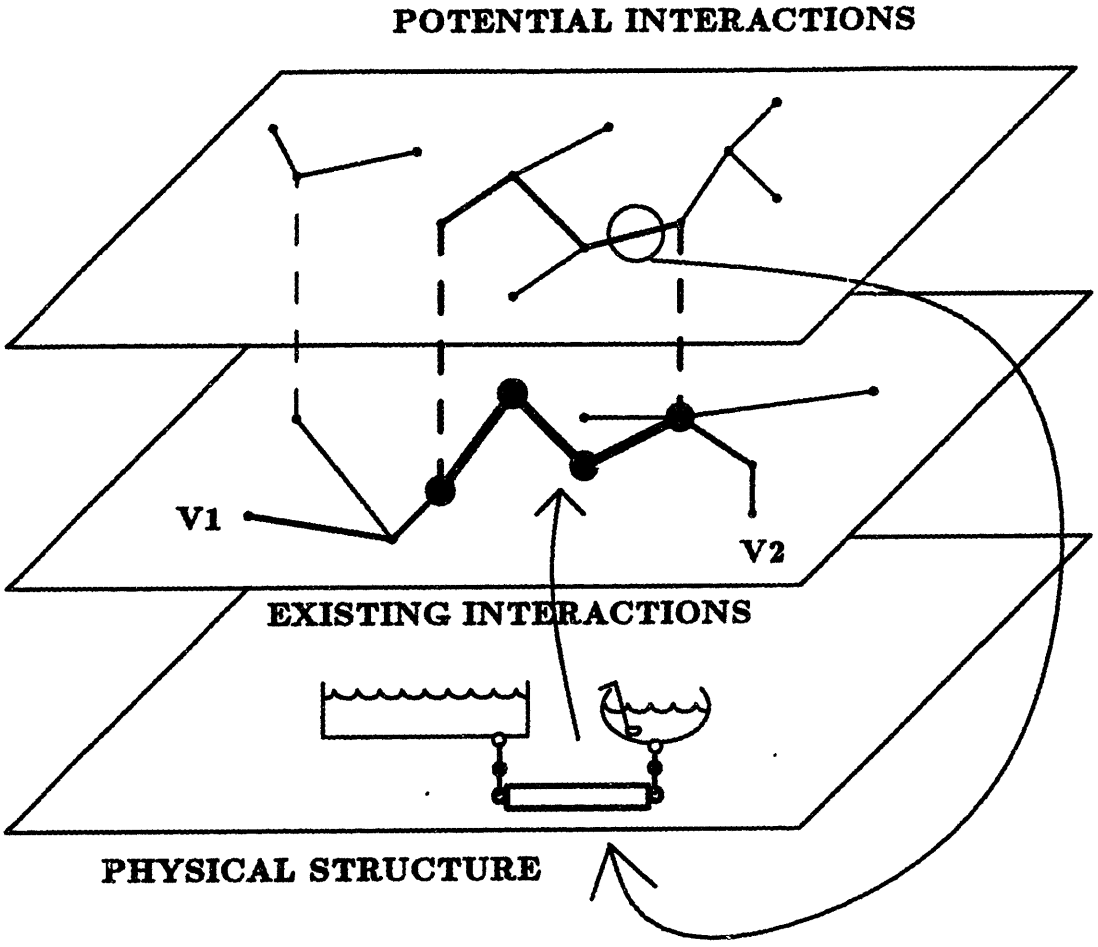


Figure 3.14: Making augmentations to the device's topology of existing interactions, by augmenting its physical structure.

Third, Ibis tests whether this candidate is a valid solution. For it to be a valid solution, it must satisfy two conditions: First, the physical structure must be something that can be built. Second, the interaction topology must produce the desired

interaction. If the candidate fails either of these two conditions then it is refined. This refinement process is the topic of Section 3.6.

Consider the first condition – recognizing an inconsistent physical structure. The possibility for an inconsistency arises from the fact that connections in the topology of potential interactions are locally consistent, but not necessarily globally consistent. Recall that the topology of potential interactions, and the connection between the two topologies were constructed by linking pairs of variables that could possibly be shared – that is, variables that can consistently be made the same. Making two variables be the same places constraints on physical structure. For example, suppose variables $Q_{bot(nc)}$ and $Q_{t1\_of(N)}$ are to be made the same, where nc is any normal size container and N is any node. Then this requires bot(nc) to be the same as $t1\_of(N)$ (i.e., the terminal – bottom of nc – is connected to node N). For a link to be consistent, these constraints on structure must be physically realizable.

When linking variables in the potential interaction topology, Ibis only tests for pairwise consistency, between the two variables linked together. This does not guarantee that a set of these links is globally consistent. Thus, a set of links along a path may result in a physically unrealizable structure.

Consider an example, consisting of an inconsistent pair of links. For the first we use the above link between $Q_{bot(nc)}$ and $Q_{t1\_of(N)}$. The second link is between $Q_{t2(pi)}$ and $Q_{t1\_of(N)}$, where pi is any pipe. Separately each of these links is consistent; however, together they are inconsistent. If both interactions are instantiated we get:

$$Q_{bot(nc)} \equiv Q_{t1\_of(N)} \equiv Q_{t2(pi)}$$

Q is a one to one function, thus:

$$bot(nc) \equiv t1\_of(N) \equiv t2(pi)$$

However, the bottom of the normal container, *bot*(*nc*), can't be the same as the end of the pipe, *t2*(*pi*) – *nc* and *pi* are distinct components (a component can't be both a container and a pipe), and by definition distinct components have distinct terminals. This example demonstrates the point that instantiating sets of links can lead to physically unrealizable structures. These consistency tests are performed by Iota (Chapter 8), using knowledge about how classes of physical parts and variables can be related, and existing constraints on the design's interaction topology and physical structure.

The second condition to be tested for is that the interactions in the candidate produce the behavior of the desired interaction. To check this we first combine each of the interactions in the candidate into a single interaction between the variables in the desired interaction. This combined interaction is then compared to the desired interaction. The composition and comparison operations are performed by the symbolic algebra system for Q1, called Minima (Chapter 6).

Given this graph, the strategy for constructing a device involves four steps: first, identifying new interactions to be added, by tracing a path; second, instantiating the solution, by augmenting the device's interaction topology and physical structure, and, third, testing that this is a valid solution, and refining it when necessary.

To summarize, the basic steps of Ibis' strategy for design from first principles is:

1. Construct the topology of existing and potential interactions.
2. Identify a coarse solution – new interactions to be added – by tracing a path.
3. Augment the device's interaction topology, by making the requisite augmentations to physical structure.
4. Test the correctness of the solution by:
   (a) checking if the structure is physically realizable, and

(b) determining whether the desired behavior is produced.

5. If incorrect, refine the solution wherever possible.

3.5 Applying the Strategy to the Punch Bowl Problem

We use the punch bowl problem to make concrete the basic steps of the invention strategy outlined above. Keep in mind that the only purpose of the example is to make the basic concepts clear, not to demonstrate the robustness of the approach. As we pointed out earlier, we are using simple component models and have supplied only those models that are actually used in the solution. Examples that demonstrate the robustness of the approach are explored in Section 3.7.

Recall from Section 2.1 that, in the punch bowl problem, we are given a vat of punch and a punch bowl sitting on a table. Both are open to the air (uncovered). Connections can be made to the bottom of either container, but no connections are allowed to their tops (i.e., "no unsightly connections").

The design problem is to maintain the level of punch in the bowl at the same level as the punch in the vat. In Chapter 2 we saw that, during the first two stages of interaction-based invention, Ibis transforms this goal to the following desired interaction:

$$[H_v - H_b] = [dH_b/dt]$$

To solve this problem we consider each of the steps, enumerated in the last section.

3.5.1 Constructing the Search Space

Ibis begins by constructing the topology of potential interactions for the fluid domain (Figure 3.11) – if one doesn't already exist. Next it constructs the topology of existing interactions for the initial problem statement (Figure 3.6), and then links the two topologies together. Recall that the interactions are supplied by the component models and connection laws, and that for each link the pair of shared variables is identified by Iota (described in Chapter 8).

3.5.2 Constructing the Coarse Solution

Next, Ibis starts looking for a coarse solution by tracing paths through the two topologies between variables in the desired interaction: H_v, H_b and dH_b/dt. Recall we begin by tracing paths outward from each of these variables along the topology of existing interactions, cross over to the potential interaction topology where access variables are reached, and continue until all the paths meet (possibly crossing between the topologies several times).

We need to be a bit more precise about what the tracing process entails. Thus far we have said, first, that the paths must go between each of the variables in the desired interaction – if these variables aren't all interconnected, then the desired interaction can't be produced. Second, the path should not contain intervening constants – their values cannot be changed, and thus would isolate the two variables we are trying to relate. What we haven't specified is whether or not any branches of the path can terminate on variables (or constants), other than those in the desired interaction. The approach we take is:

> When constructing a path, every branch must terminate either on a variable in the desired interaction or a physical constant.

This approach is based on the following intuitions. If a path terminates on a variable, then that is the only interaction the variable appears in. If this variable is not mentioned in a desired interaction, then it will somehow have to be eliminated when all the interactions are composed together; otherwise, the composite interaction won't match the desired interaction. However, if the variable only appears in one interaction then it is highly unlikely that it will be eliminated. To avoid this complication, the path is not allowed to terminate on variables not in the desired interaction.

Rather than elaborating on this argument at this point, it is more important to make the approach concrete through our example. In Section 3.8 we examine the argument more carefully, demonstrate cases where the approach breaks down, and suggest more conservative approaches that avoid this problem. Based on our limited experience, cases where the approach breaks down are uncommon, and are therefore not a great concern.

To construct the coarse solution for the punch bowl problem, Ibis first traces a path through the topology of existing interactions, starting at the variables in the desired interaction – H_v, H_b and dH_b/dt – and searching for reachable access variables (Figure 3.15). For example, moving outward from H_v (upper left of the figure), the first interaction traversed is one relating H_v to d, g and $Pd_{nt(v),nb(v)}$, which we denote:

$$\langle H_v, d, g, Pd_{nt(v),nb(v)} \rangle$$

The path branches, moving to d, g, and $Pd_{nt(v),nb(v)}$. d and g are constants, so their paths terminate (denoted as black dots on the figure). The path at $Pd_{nt(v),nb(v)}$ continues, traversing the interaction:

$$\langle Pd_{nt(v),nb(v)}, P_{nt(v)}, P_{nb(v)} \rangle$$

Again the path branches, one going to $P_{nt(v)}$ and the other to $P_{nb(v)}$. The path at

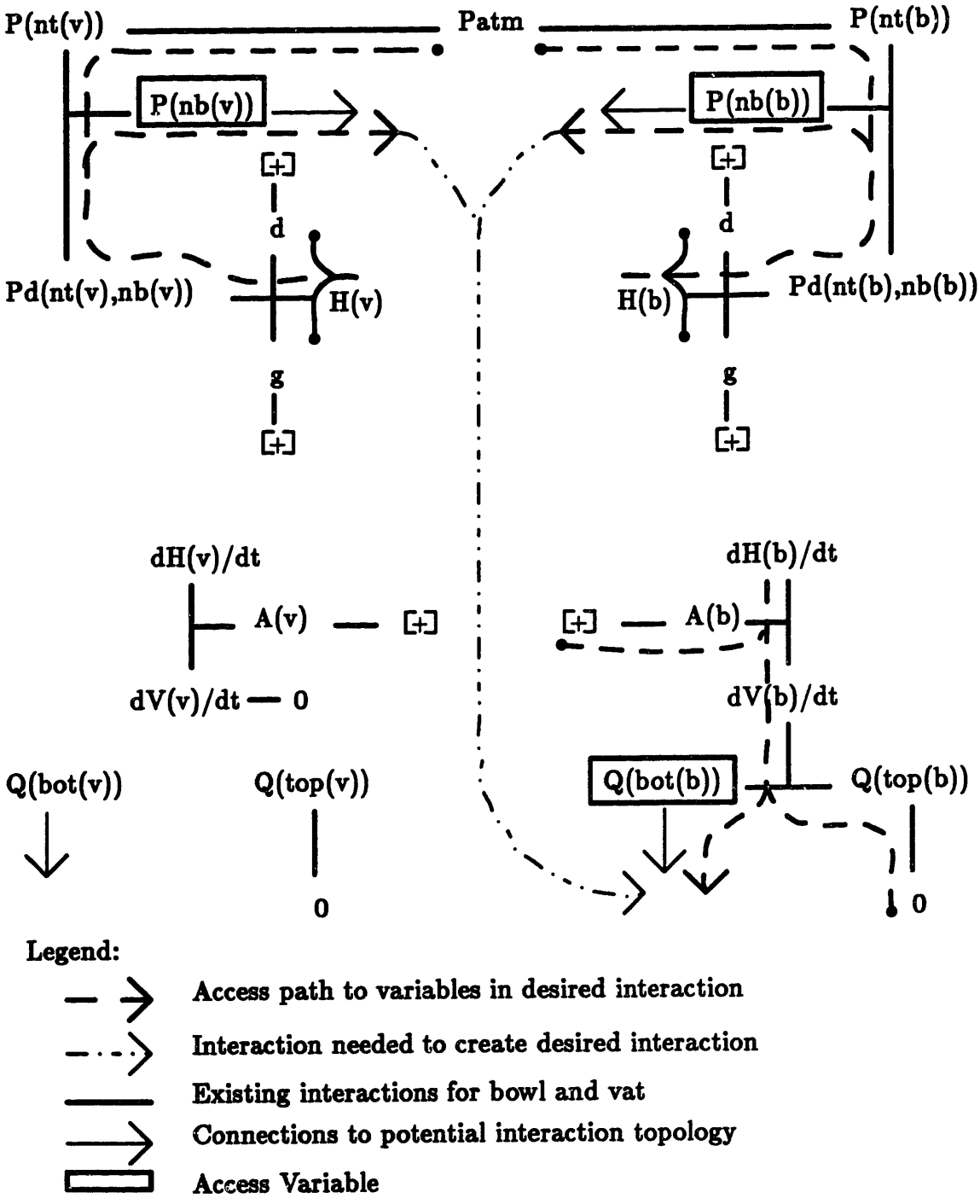


Figure 3.15: Identifying places to inject new interactions for the punch bowl problem.

$P_{nt(v)}$ traverses the interaction:

$$\langle P_{nt(v)}, P_{atm} \rangle$$

where it terminates, since P_{atm} is a constant. Examining the other branch, $P_{nb(v)}$ is an access variable; thus, the first part of the search is completed – we have found an access variable, $P_{nb(v)}$, and access path for H_v. This path is shown as a dashed line from H_v to $P_{nt(b)}$ in the upper left corner of Figure 3.15.

Paths for the other two variables in the desired interaction, H_b and dH_b/dt, are constructed in a similar manner and are also shown in Figure 3.15. H_b is accessible by $P_{nb(b)}$; its access path is shown in the upper right corner of the figure. And dH_b/dt is accessible by $Q_{bot(b)}$; its access path is to the lower right. Given these three access paths and variables, the desired interaction can be produced by constructing an interaction between $P_{nb(v)}$, $P_{nb(b)}$, and $Q_{bot(b)}$. This is shown as a dash-dot line in the figure.

Next, paths are traced out from the access variables, through the potential interaction topology, until they meet. The result is a path connecting the variables in the desired interaction. The path through the topology of potential interactions is shown in Figure 3.16 as a dash-dot line traced between the upper right and lower left corners.

Starting at the first access variable, $P_{nb(v)}$, Ibis uses the link going to P_{n1} of the compatibility law to cross from the topology of existing interactions to the potential interaction topology (upper left corner of Figure 3.16). Likewise, $P_{nb(b)}$ crosses to P_{n2} of the compatibility law (upper left corner), and $Q_{bot(b)}$ crosses to $Q_{t1(N)}$ of the continuity law (lower right corner). Moving out from the first two variables, P_{n1} and P_{n2}, the two paths meet immediately at the interaction supplied by the compatibility law:

$$\langle P_{n1}, P_{n2}, Pd_{n1,n2} \rangle$$

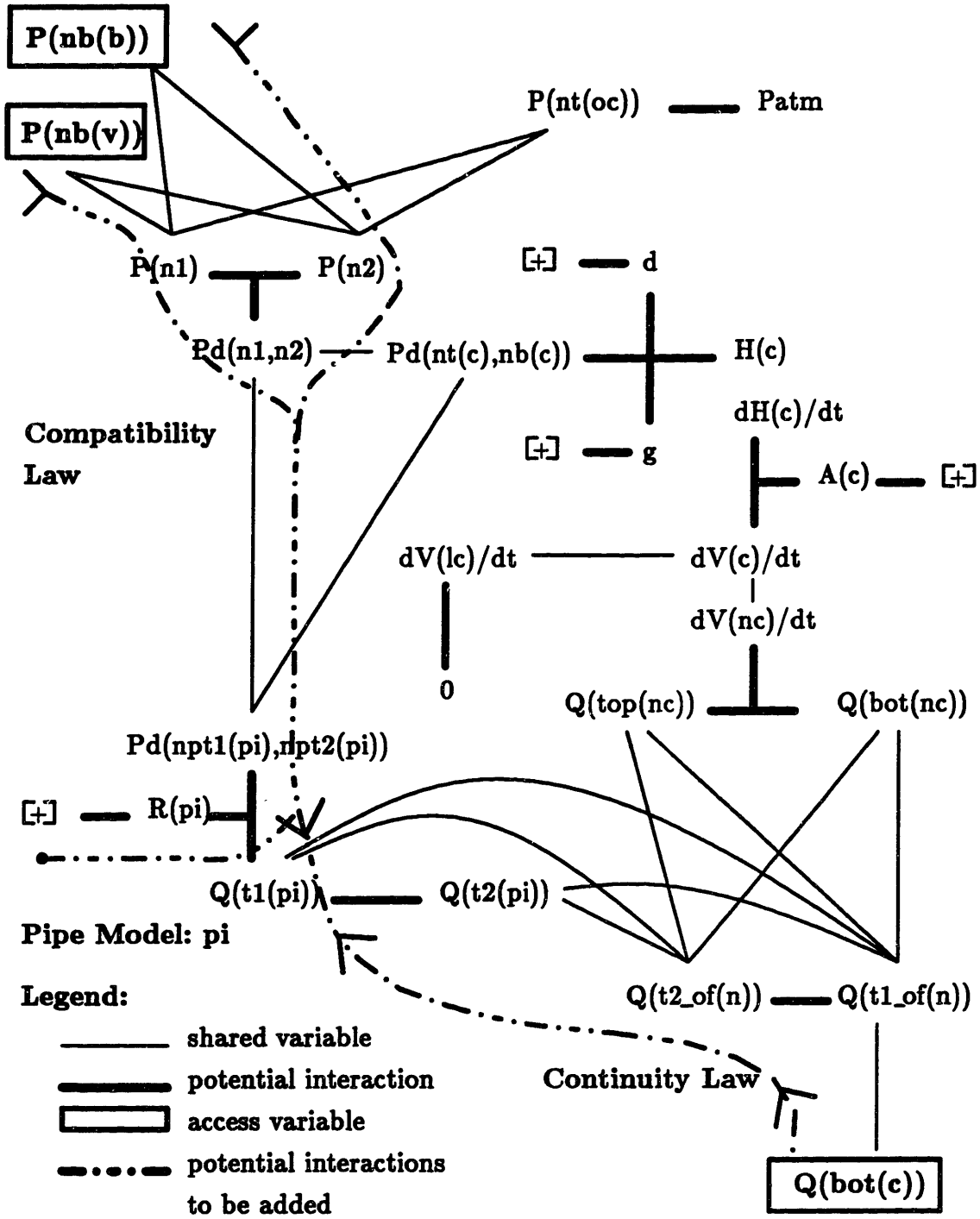


Figure 3.16: Identifying augmentations to the interaction topology for the punch bowl problem.

The merged path continues to $Pd_{n1,n2}$, and then crosses the link to $Pd_{npt1(pi),npt2(pi)}$, which is part of the pipe model. Next the path moves through the pipe interactions, starting with:

$$\langle P_{npt1(pi),npt2(pi)}, R_{pi}, Q_{t1(pi)} \rangle$$

The first branch terminates on the constant R_{pi}, and the second branch goes to $Q_{t1(pi)}$. This path continues through a second pipe interaction:

$$\langle Q_{t1(pi)}, Q_{t2(pi)} \rangle$$

reaching $Q_{t2(pi)}$.

At this point the path meets up with the third path, which by this time has moved out from $Q_{bot(c)}$ (lower left corner of the figure) to $Q_{t1\_of(n)}$, through the interaction supplied by the continuity law to $Q_{t2\_of(n)}$, and across the link to $Q_{t2(pi)}$ of the pipe model. All the branches have converged or terminated on constants; thus, the combined set of paths constitute a coarse solution – a topology of interactions that relates the variables in the desired interaction (Figure 3.17).

In this example the paths started in the topology of existing interactions, crossed once to the topology of potential interactions, and then came together. In general the paths may cross back and forth between the two topologies any number of times. This allows Ibis to exploit existing interactions whenever possible, thus minimizing the number of augmentations that must be made to physical structure.

3.5.3 Augmenting the Device's Interaction Topology and Physical Structure

Ibis augments the topology of existing interactions, according to the proposed solution, by making augmentations to physical structure that produce these new in-

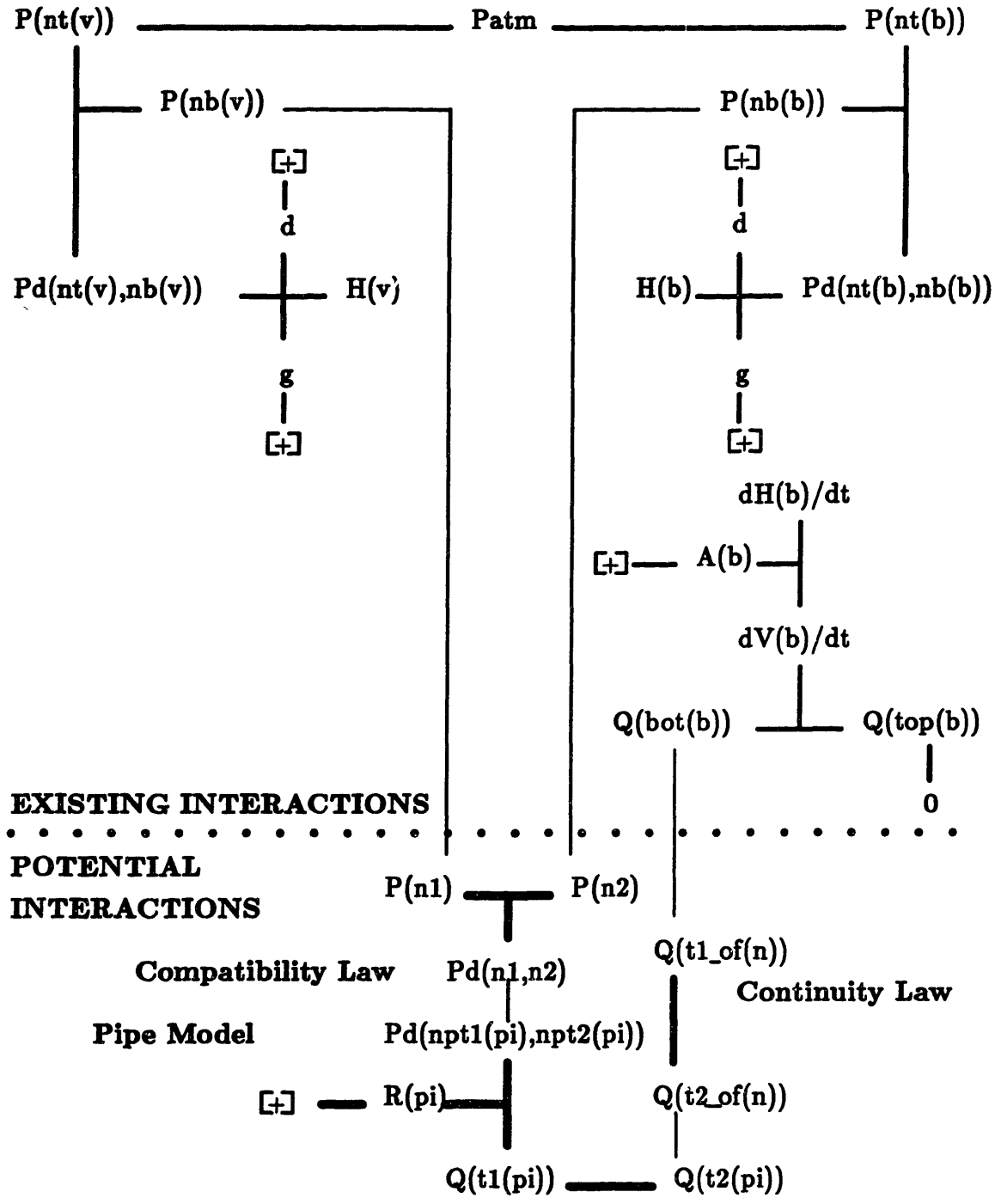


Figure 3.17:

A coarse solution to the punch bowl problem – interactions in the topologies of existing and potential interactions that lie along the path traced between the variables in the desired interaction.

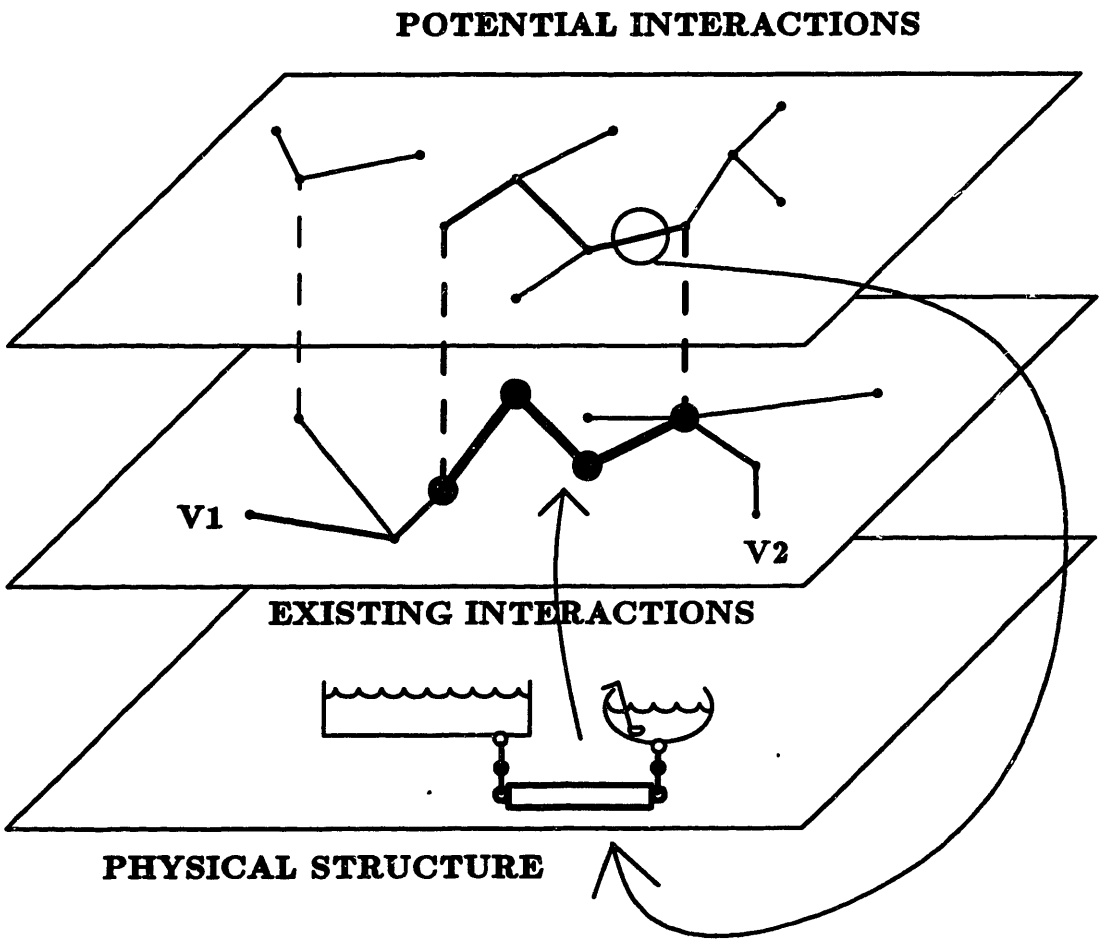


Figure 3.18: Making augmentations to a device's topology of existing interactions, by augmenting its physical structure.

teractions (Figure 3.18). The process of augmenting the topology of existing interactions breaks into two parts: instantiating the appropriate potential interactions, and connecting them together by instantiating the relevant links. This is accomplished through analogous operations on physical structure – interactions are instantiated by introducing physical parts, and links are instantiated by equating physical parts. This in turn equates existing variables, thereby connecting the interactions together. Consider how Ibis performs these two steps for the punch bowl example.

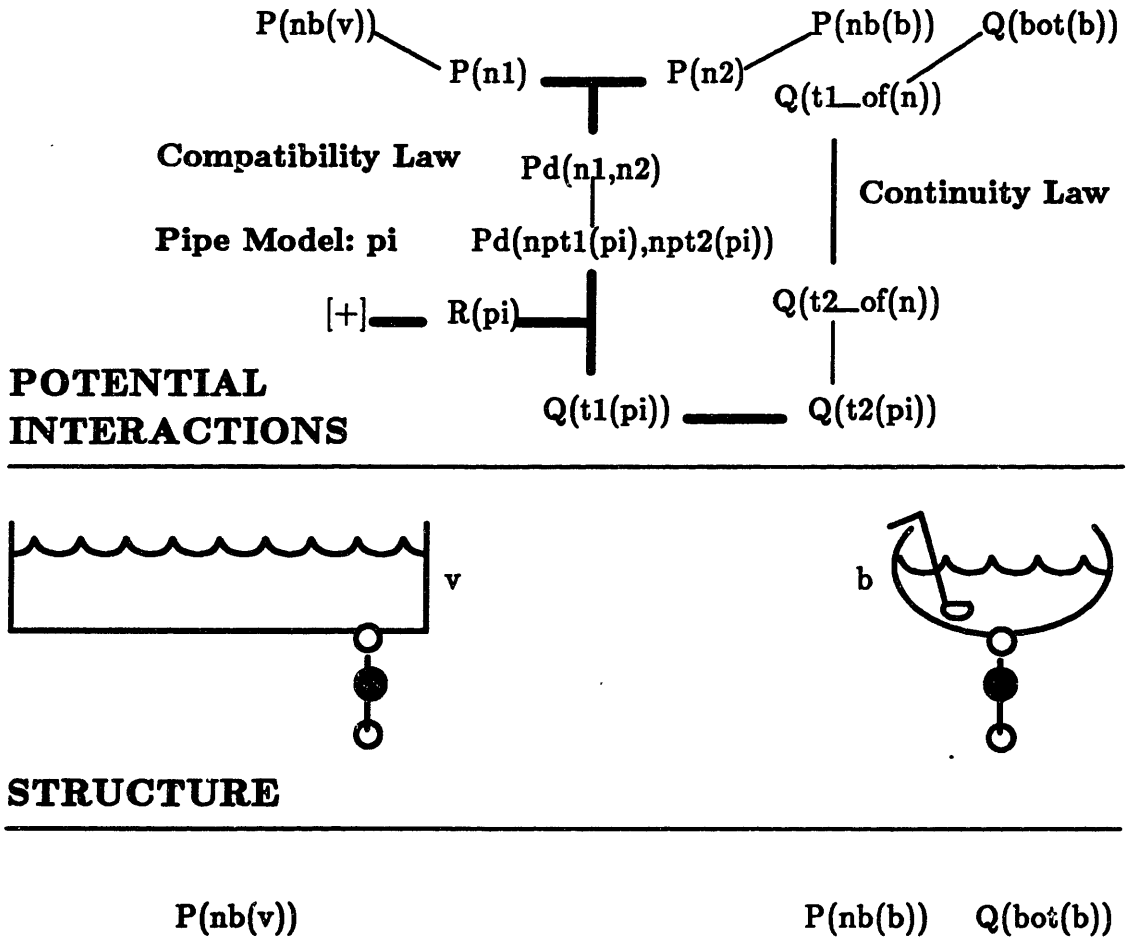


Figure 3.19: Relevant components of the topology of potential interactions, physical structure, and topology of existing interactions, immediately before making augmentations.

Figure 3.19 shows the initial situation for the punch bowl example, when the augmentation process begins. At the top is the part of the potential interaction topology that was traversed (Figure 3.16). In the middle of Figure 3.19 is the initial physical structure. Note that the pair of open circles connected to each node, n (a closed circle), are the two terminals $t1\_of(n)$ and $t2\_of(n)$. At the bottom are the variables from the topology of existing interactions that the augmentations will be connected to.

Ibis begins by instantiating potential interactions traversed by the path. In the example, five of these interactions were traversed (Figure 3.19): the interaction from the continuity law, the three interactions from the pipe model, and the interaction from the compatibility law.

First Ibis instantiates the interaction from the continuity law. The continuity law describes the interactions of any single node, n; thus, we produce the interaction by introducing a node [4] – call it $N3$. Note that, along with $N3$, its two terminals, $t1\_of(N3)$ and $t2\_of(N3)$, are introduced. Introducing $N3$ results in a single interaction being added to the topology of existing interactions, as is shown in Figure 3.20.

The remaining four interactions are introduced in a similar manner, as is shown in Figure 3.21. Three of the interactions are part of the pipe model, and are instantiated by introducing a pipe – call it $P1$. Note that, along with the pipe, Ibis introduces the pipe's terminals and the nodes they are connected to. The remaining potential interaction is part of the compatibility law, which applies to any pair of nodes. Thus, Ibis introduces two additional nodes – call them $N1$ and $N2$.

To complete the interaction topology Ibis connects the interactions together by instantiating the traversed links. Recall that a link denotes a pair of variables, con-

[4]Introducing a node (or any physical part) does not imply that it is distinct. For example, below Ibis infers that $N2$ and $N3$ refer to the same node.

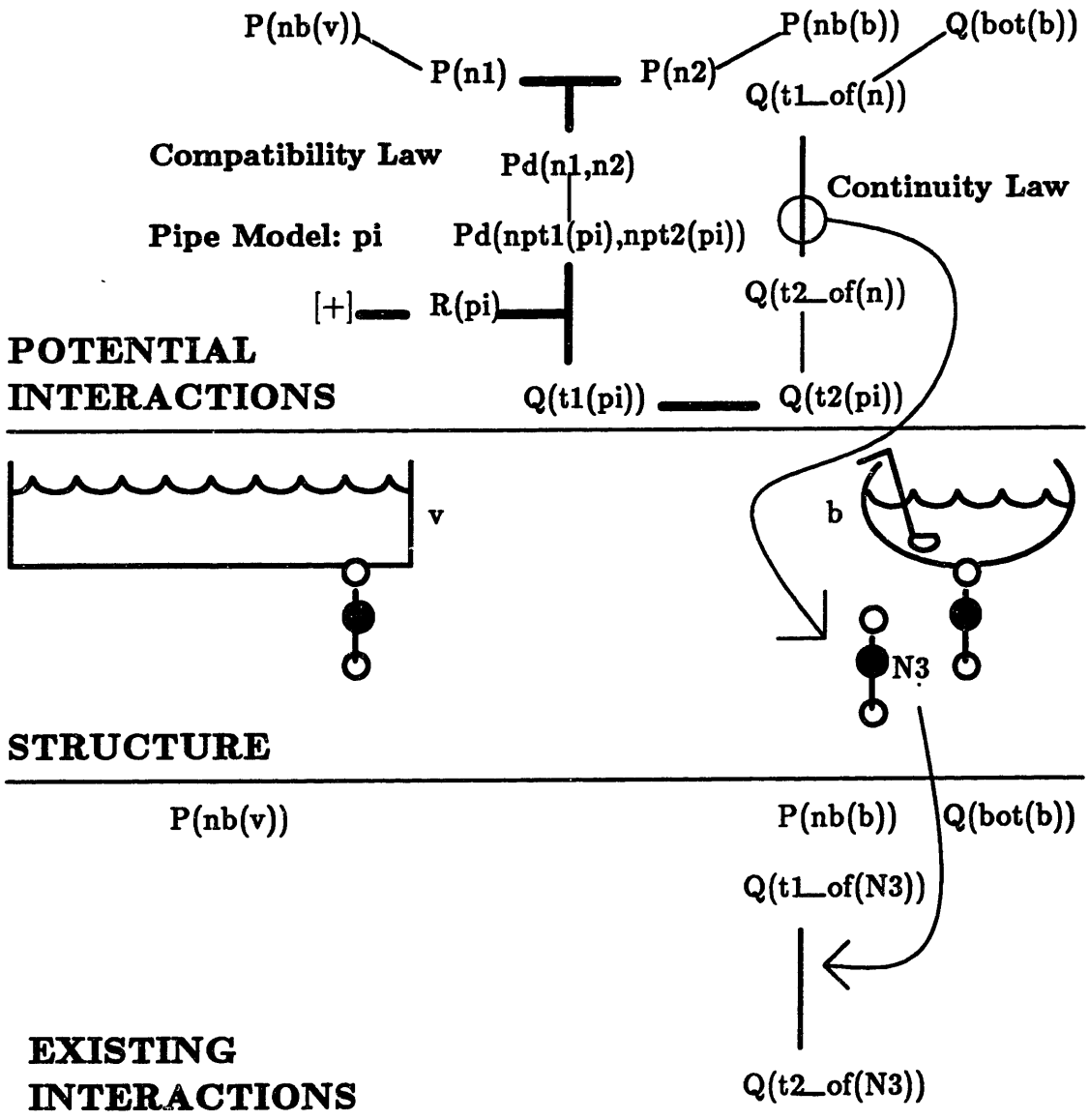


Figure 3.20: The potential interaction from the continuity law is added to the topology of existing interactions by introducing a node $N3$, together with its corresponding terminals, $t1\_of(N3)$ and $t2\_of(N3)$.

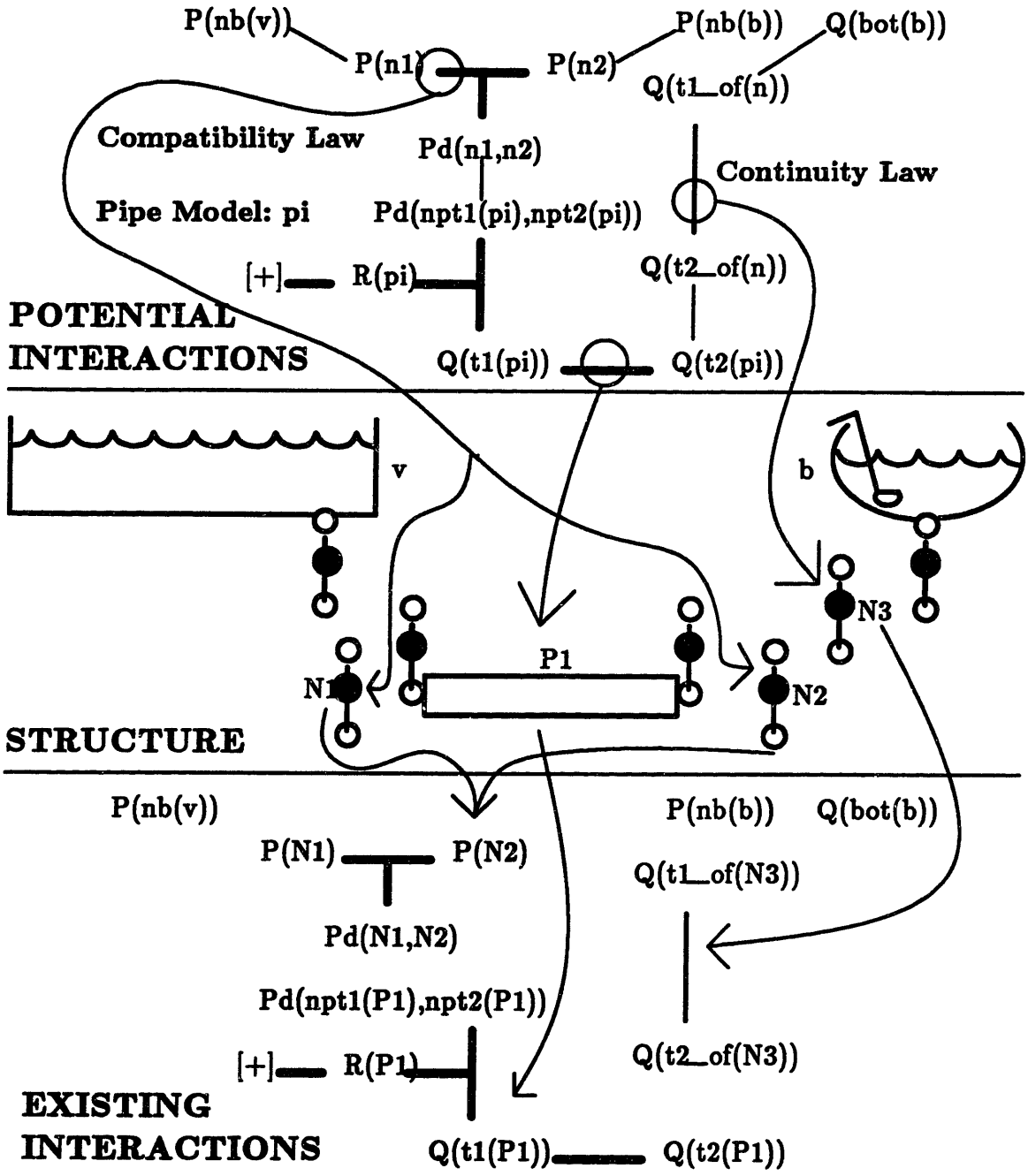


Figure 3.21: Introducing the remaining potential interactions into the topology of existing interactions.

tained by two potential interactions, that are to be shared. To instantiate the link, we equate the two variables within the two corresponding interactions in the topology of existing interactions. In the example, the path traversed a link between $P_{nb(v)}$ and P_{n1}. The former is an access variable in the topology of existing interactions. The latter is contained by the interaction for the compatibility law – its corresponding variable in the topology of existing interactions is P_{N1}. Thus the link is instantiated by the equivalence:

$$P_{nb(v)} \equiv P_{N1}$$

Producing this equivalence in turn forces pieces of physical structure to be equated. In this example P is a 1-1 function; that is, each node has a single unique pressure. Thus the equivalence is produced by:

$$nb(v) \equiv N1$$

That is, the bottom of the vat is connected to $N1$ (Figure 3.22).

This is how instantiating links results in components being connected into a network. Another possible result is that two components can be merged (i.e., they are equated). This results in function sharing (i.e., sharing the interactions of a component with several paths or separate parts of a single path). It may also have the net effect of specializing a component, since the combined component must satisfy the type restrictions (e.g., closed container and very large container) of the components that were equated.

In the example five links are traversed. Instantiating them connects the interactions together through the following five equivalences:

$$\begin{aligned} P_{nb(v)} &\equiv P_{N1} \\ P_{nb(b)} &\equiv P_{N2} \end{aligned}$$

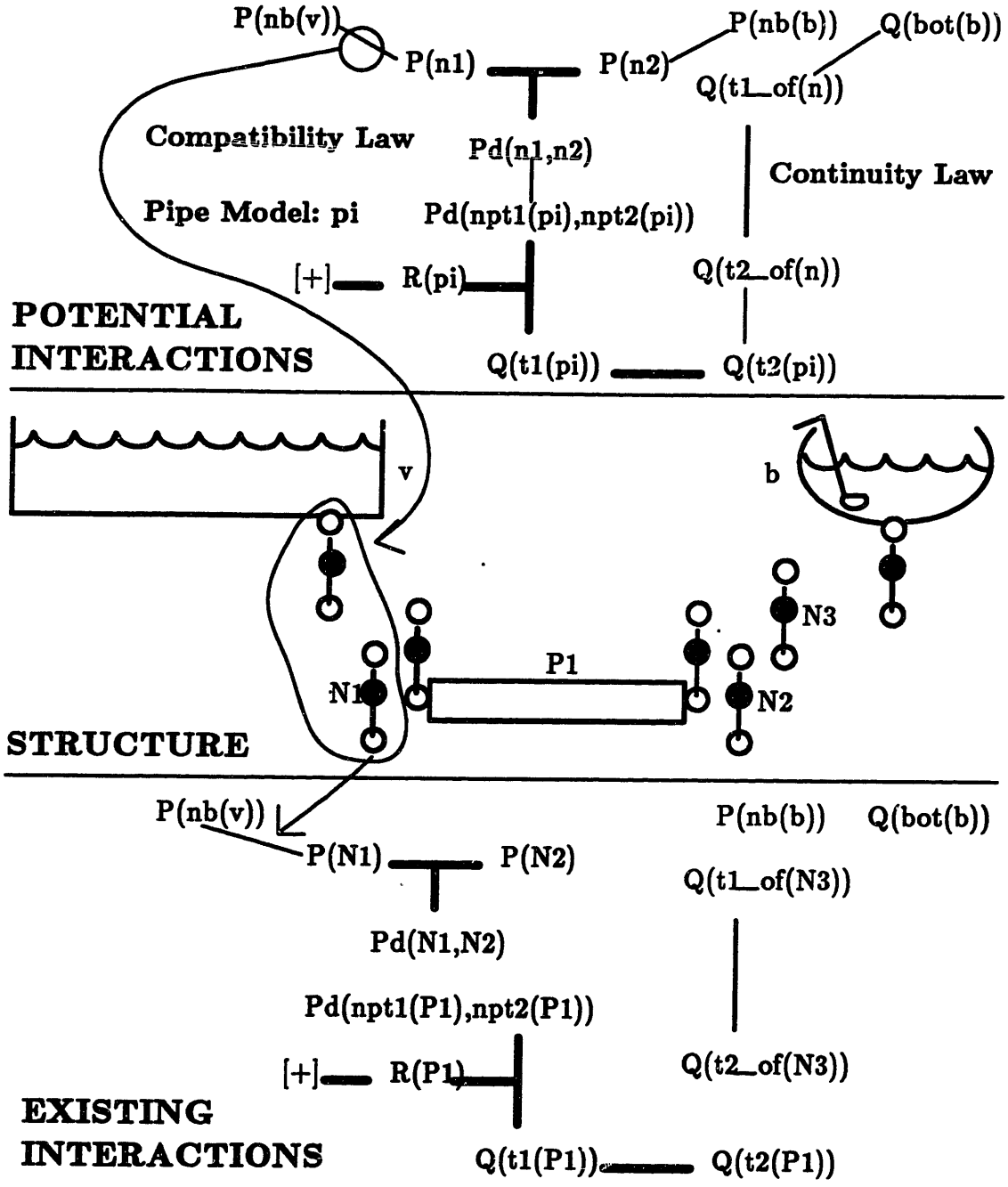


Figure 3.22: A link between $P_{nb(v)}$ and P_{N1} is introduced by connecting the bottom of the vat to node $N1$.

$$
\begin{aligned}
Pd_{N1,N2} &\equiv Pd_{npt1(P1),npt2(P1)} \\
Q_{t2(P1)} &\equiv Q_{t2\_of(N3)} \\
Q_{t1\_of(N3)} &\equiv Q_{bot(b)}
\end{aligned}
$$

The impact of instantiating these links on the device's interaction topology and physical structure is shown in Figure 3.23. That is, $T1$ of the pipe has been connected to the vat through the shared node $N1$. $T2$ of the pipe has been connected to the bowl through the shared node $N2 = N3$. Consider these consequences in more detail. The first two links tell Ibis that the bottom of the vat is connected to $N1$ and the bottom of the bowl is connected to $N2$:

$$
\begin{aligned}
nb(v) &\equiv N1 \\
nb(b) &\equiv N2
\end{aligned}
$$

The third link tells Ibis that terminal $t1$ of pipe $P1$ is connected to $N1$, and $t2$ is connected to $N2$. That is, the pipe is connected between the bottoms of the vat and bowl:

$$
\begin{aligned}
N1 &\equiv npt1(P1) \\
N2 &\equiv npt2(P1)
\end{aligned}
$$

The fourth link tells Ibis that the terminal connected to $N3$, called $t2\_of$, is $t2$ of the pipe. And conversely it tells Ibis that $t2$ of the pipe is connected to $N3$. In addition, since by definition a terminal is connected to exactly one node, and $t2$ is already connected to $N2$, this tells Ibis that $N2$ and $N3$ are really the same node:

$$
\begin{aligned}
t2\_of(N3) &\equiv t2(P1) \\
N3 &\equiv npt2(P1)
\end{aligned}
$$

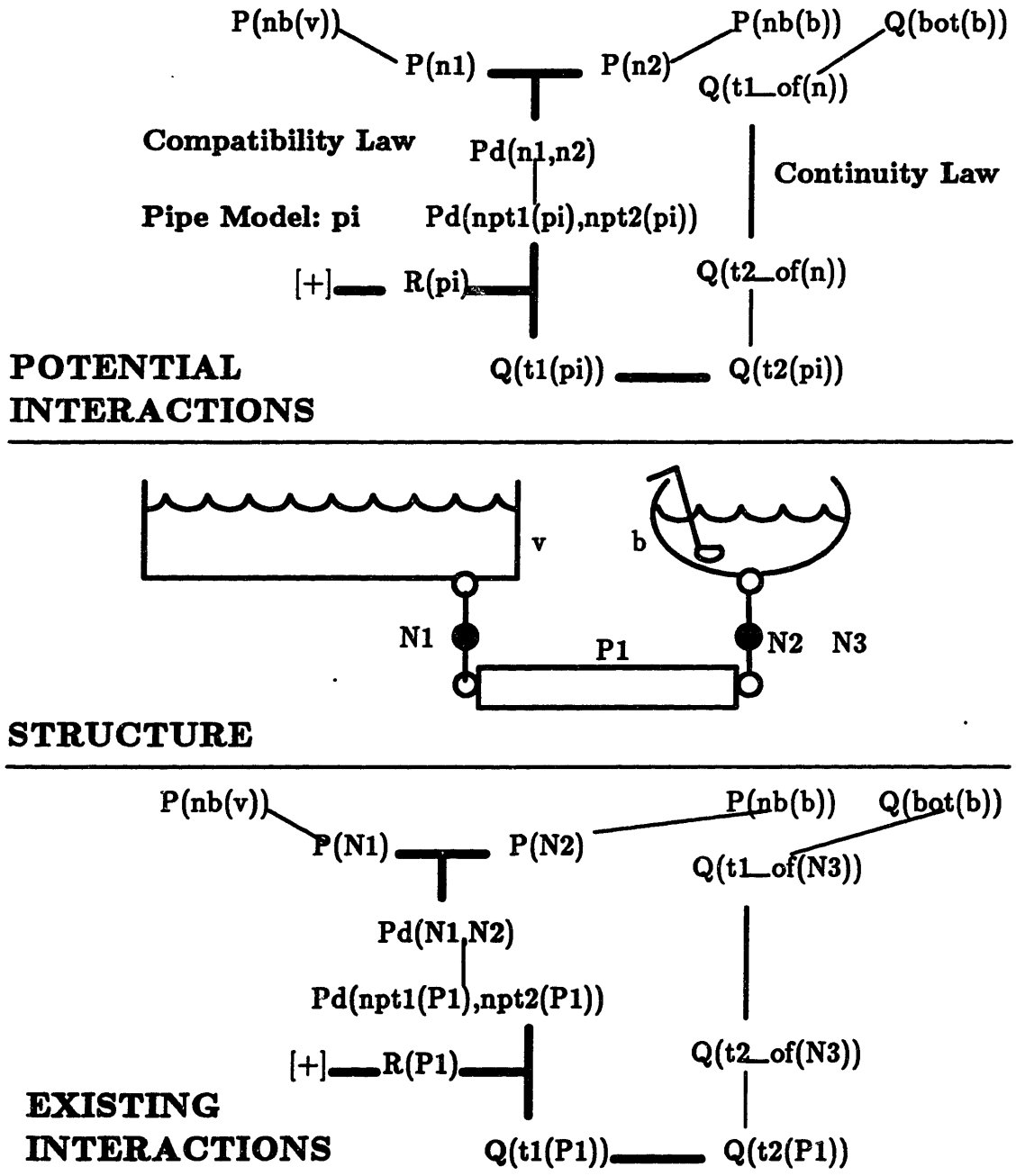


Figure 3.23: Introducing the remaining links interactions into the topology of existing interactions, results in a pipe being connected between the vat and the bowl.

$$N3 \equiv N2$$

Finally, the fifth link tells Ibis that the terminal connected to $N3$, called $t1\_of$, is the bottom of the bowl. The consequences of this are similar to that for the fourth link:

$$\begin{aligned} t1\_of(N3) &\equiv bot(b) \\ N3 &\equiv nb(b) \\ N3 &\equiv N2 \end{aligned}$$

Each of these inferences about equivalences are performed by what we refer to as a constructive reasoning component, called Iota. Iota performs these inferences based on relationships between classes of objects like terminals, nodes and variables, defined by physical system dynamics, the domain and technology. These are combined with relationships between instances of these classes, imposed by the existing interaction topology and physical structure. More specifically, when instances of components and nodes are introduced, the types of these objects are recorded (e.g., pipe or normal size container). As links are instantiated, the corresponding equivalences are supplied to Iota. Iota then uses this information to infer, not only additional equivalences, but restrictions on the types of objects and logical inconsistencies. How Iota works and the knowledge it uses are described in Chapter 8.

Reviewing the results obtained in this section: Ibis produces the interactions traversed in the topology of potential interactions by introducing a pipe, called $P1$, and three nodes, called $N1$, $N2$ and $N3$. To connect the interactions into a topology, Ibis instantiates the links traversed. The consequences of this physical structure are: the pipe is connected from the bowl to the vat via the shared nodes $N1$ and $N2$ respectively, and $N2$ and $N3$ are different names for the same node. The complete interaction topology and physical structure resulting from this instantiation are shown in Figures 3.24 and 3.25.

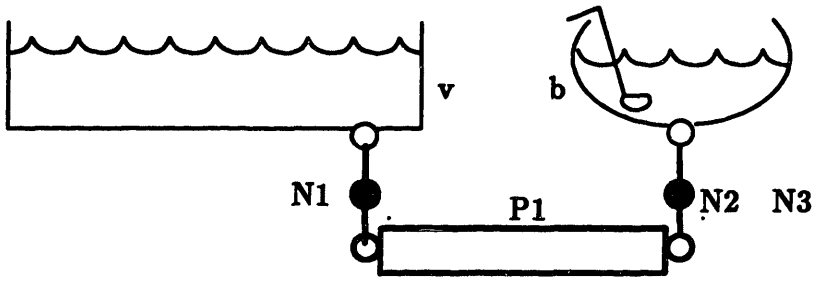


Figure 3.24: Augmentations to physical structure necessary to instantiate potential interactions and links in the coarse solution.

3.5.4 Testing if the Structure is Physically Realizable

In Section 3.4 we observed that instantiating a link may force two pieces of structure to be equivalent that must be distinct. The example we used was where terminals from distinct components are equated. This can't happen, since by definition terminals are not shared between components. Thus, as Ibis instantiates links, and has Iota infer equivalences from them, Ibis also asks Iota whether these equivalences are globally consistent. How Iota performs this test is discussed in Section 8.2. In this example, none of the equivalences are inconsistent; thus, the augmentations to physical structure can be built. Section 3.6 explores an alternative path that leads to an inconsistent structure.

3.5.5 Testing if the Desired Interaction is Produced

The remaining step is to determine whether or not the combined behaviors of the interactions along the path produce the desired interaction:

$$[H_v - H_b] = [dH_b/dt]$$

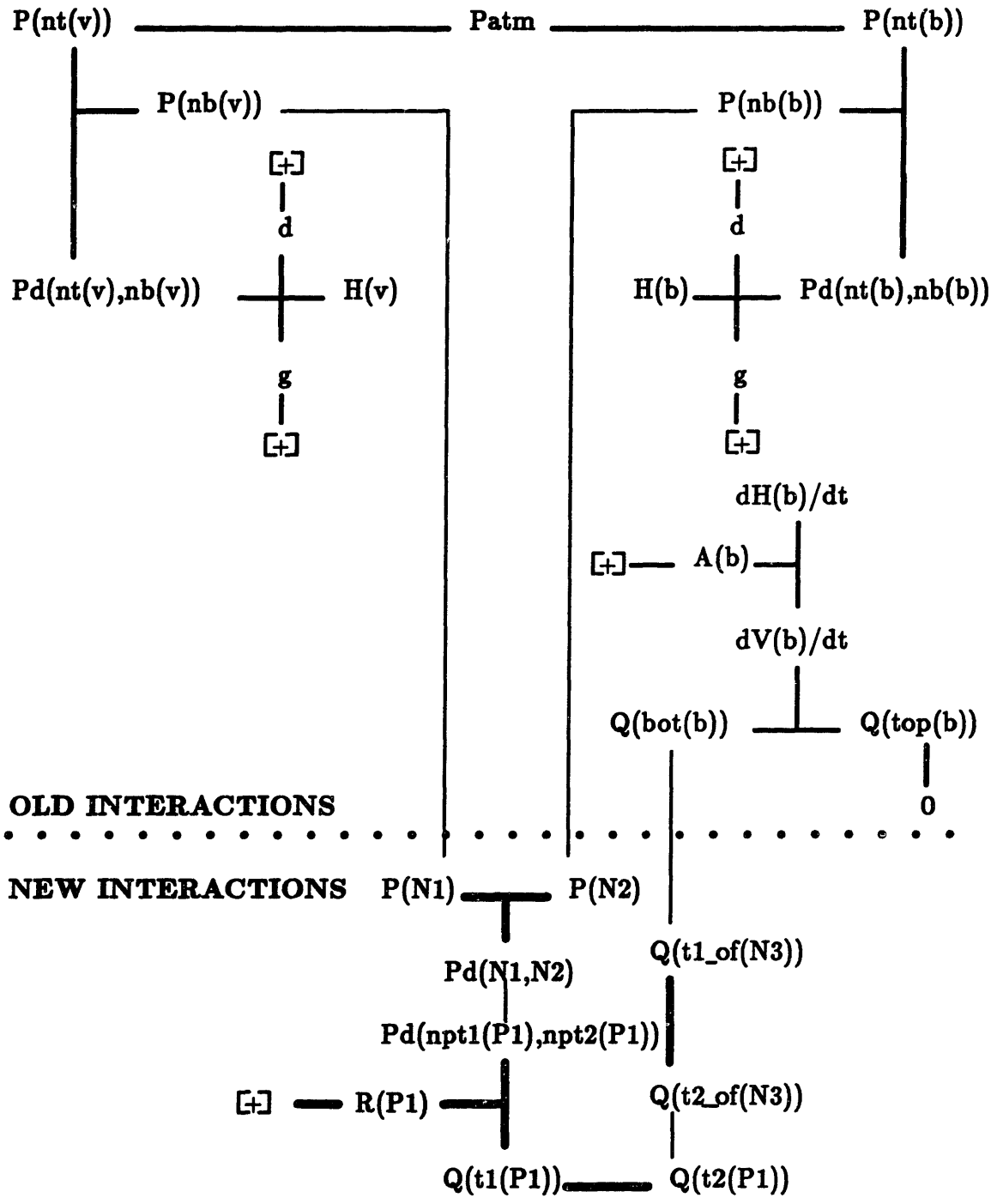


Figure 3.25: Instantiation of proposed interaction topology for relating variables of the desired interaction. The thin lines connecting expressions for variables (e.g., $P(N1)$ and $P(nb(v))$) indicate that they denote the same variable.

Each interaction is described by a qualitative equation, thus composing the interactions involves performing qualitative symbolic algebra. To combine all the interactions traced along the path, Ibis starts with the variable being controlled, walks along the path accumulating a symbolic expression, and then tests whether the expression matches the desired behavior. For example, to verify the solution to the punch bowl problem, Ibis starts with $[dH_b/dt]$, the right side of the desired interaction, and compares the accumulated expression with $[H_v - H_b]$, the left side of the desired interaction.

At each step of the process, the expression is accumulated by solving for a shared variable using substitution of equals for equals, and then simplified. For example, starting with $[dH_b/dt]$, the fist interaction encountered along the path is one of the bowl interactions:

$$V_b = H_b \times A_b$$

This is the interaction labeled 1 on the right side of Figure 3.27, near the middle. The shared variable is V_b. Solving for it in the interaction we get:

$$V_b = V_b/A_b$$

Substituting into $[dH_b/dt]$ produces:

$$[d(V_b/A_b)/dt]$$

Differentiating and using the fact that A_b is positive (i.e., the interaction labeled 2 in Figure 3.27), this simplifies to:

$$[dV_b/dt]$$

To perform these algebraic manipulations Ibis uses the symbolic algebra system, called *Minima* for the hybrid qualitative, quantitative algebra $Q1$. Chapter 6 provides a precise definition of $Q1$, highlighting its unique features. It also analyzes the

properties of $Q1$, and develops the symbolic algebra procedures used in Minima.

The sequence of interactions that are walked through during the accumulation are numbered in Figure 3.27. The corresponding expressions produced at each step of the accumulation are shown in Figure 3.26. From the initial expression propagated, $[H_v - H_b]$ (labeled 0 in the figure), and the expression at the end of the propagation, $[dH_b/dt]$ (right side of 17 in the figure), we get:

$$[H_v - H_b] = [dH_b/dt]$$

Thus, the interaction topology produces the desired interaction – connecting the pipe solves the punch problem.

Note that, although the resulting solution is quite compact, this is because the design exploits a substantial set of interactions produced by existing structure. The complete set of interactions directly contributing to the desired behavior is quite extensive. Furthermore, we have both designed and verified a device *whose behavior couldn't even have been described using traditional qualitative representations.*[5]

3.6 Refining Failed Solutions

The example just discussed demonstrates an ideal case: a path is identified, it is instantiated with no problems, and the behavior matches the desired interactions. In many cases the process won't be quite so simple – either the topology can't be built, or doesn't produce the desired behavior. In this section we demonstrate, through an example, how Ibis recognizes and handles these failures.

[5]Chapter 5 demonstrates that existing qualitative algebras are not powerful enough to describe the punch bowl problem, and that existing qualitative simulation techniques are not powerful enough to uniquely predict the resulting design behavior.

	before simplification	after simplification
0)	$[dH_b/dt]$	
1)	$[(dV_b/dt)/A_b]$	$[dV_b/dt] \otimes [A_b]$
2)	$[dV_b/dt] \otimes [+]$	$[dV_b/dt]$
3)	$[Q_{bot(b)} + Q_{top(b)}]$	$[Q_{bot(b)} + Q_{top(b)}]$
4)	$[Q_{bot(b)} + 0]$	$[Q_{bot(b)}]$
5)	$[-Q_{t2\_of(N3)}]$	$\ominus [Q_{t2\_of(N3)}]$
6)	$\ominus [-Q_{t1(P1)}]$	$[Q_{t1(P1)}]$
7)	$[Pd_{npt1(P1),npt2(P1)}/R_{P1}]$	$[Pd_{npt1(P1),npt2(P1)}] \otimes [R_{P1}]$
8)	$[Pd_{npt1(P1),npt2(P1)}] \otimes [+]$	$[Pd_{npt1(P1),npt2(P1)}]$
9)	$[P_{N1} - P_{N2}]$	$[P_{N1} - P_{N2}]$
10)	$[(Pd_{nt(v),nb(v)} - P_{nt(v)}) - P_{N2}]$	$[(Pd_{nt(v),nb(v)} - P_{nt(v)}) - P_{N2}]$
11)	$[(Pd_{nt(v),nb(v)} - P_{nt(v)}) - (Pd_{nt(b),nb(b)} - P_{nt(b)})]$	$[Pd_{nt(v),nb(v)} + P_{nt(b)} - P_{nt(v)} - Pd_{nt(b),nb(b)}]$
12)	$[Pd_{nt(v),nb(v)} + P_{nt(b)} - P_{atm} - Pd_{nt(b),nb(b)}]$	$[Pd_{nt(v),nb(v)} + P_{nt(b)} - P_{atm} - Pd_{nt(b),nb(b)}]$
13)	$[Pd_{nt(v),nb(v)} + P_{atm} - P_{atm} - Pd_{nt(b),nb(b)}]$	$[Pd_{nt(v),nb(v)} - Pd_{nt(b),nb(b)}]$
14)	$[H_v/(d \times g) - Pd_{nt(b),nb(b)}]$	$[H_v/(d \times g) - Pd_{nt(b),nb(b)}]$
15)	$[H_v/(d \times g) - H_b/(d \times g)]$	$[H_v - H_b] \otimes [d] \otimes [g]$
16)	$[H_v - H_b] \otimes [d] \otimes [+]$	$[H_v - H_b] \otimes [d]$
17)	$[H_v - H_b] \otimes [+]$	$[H_v - H_b]$

Figure 3.26: Accumulated expression produced at each step of the propagation, showing that the topology of interactions proposed produces the desired interactions through symbolic propagation. Each step shows the expression immediately after substitution (left side) and after simplification (right side). The interaction incorporated at each step is numbered on Figure 3.27.

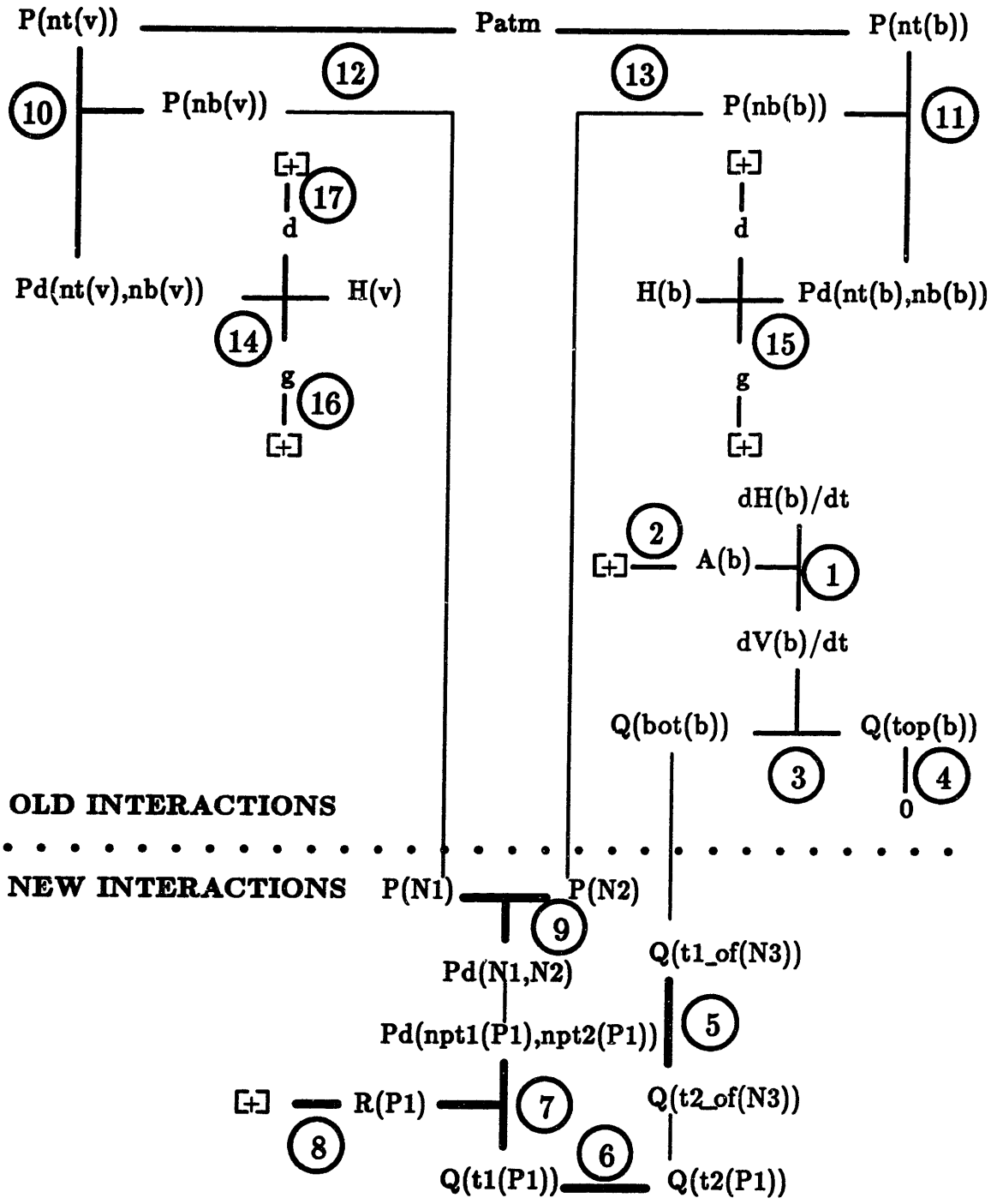


Figure 3.27: Sequence of interactions combined to verify that the interactions along the path produce the desired interaction.

When a proposed solution fails, rather than simply starting from scratch, Ibis tries to build upon what it has constructed thus far. To do this Ibis throws away parts of the instantiated path that are responsible for the failure, then looks for new paths that bridge the resulting gaps.

More specifically, the process begins the same as above: Ibis starts by tracing a set of paths through the potential and existing interaction topologies, and selects one path as an initial solution (e.g., the one involving the least number of components). Next, it constructs the links and interactions along the path, successively instantiating them from one end of the path to the other.

During the construction process, Ibis tests whether any of the instantiations produces an inconsistency. Recall that an inconsistency is produced when, by instantiating a link, two objects are equated that are necessarily distinct. When this happens, Ibis determines which links produce the inconsistency, and then removes the contributing links until all the inconsistencies vanish (i.e., through a form of dependency directed backtracking[14]). Next, for each of the links removed, Ibis uses the topology of potential interactions to identify alternative paths between the two variables in the link. If all such gaps are bridged, then Ibis continues the instantiation process. If no path is found, then Ibis must explore an alternative solution.

Consider how this applies to the punch bowl problem. As a coarse solution we have Ibis start with the path shown in Figure 3.28. This is similar to the path pursued in the previous section, except that in the lower left corner it traces through the container model, instead of the continuity law directly.

Next, Ibis begins instantiating the interactions along the path. The result is the same as the previous example – up to the point that the continuity law is reached (including the instantiation of the continuity law). During instantiation, when Ibis reaches this point, instantiating the interactions has resulted in three new nodes ($N1$, $N2$ and $N3$) and a pipe ($P1$). Instantiating the first four links has introduced the

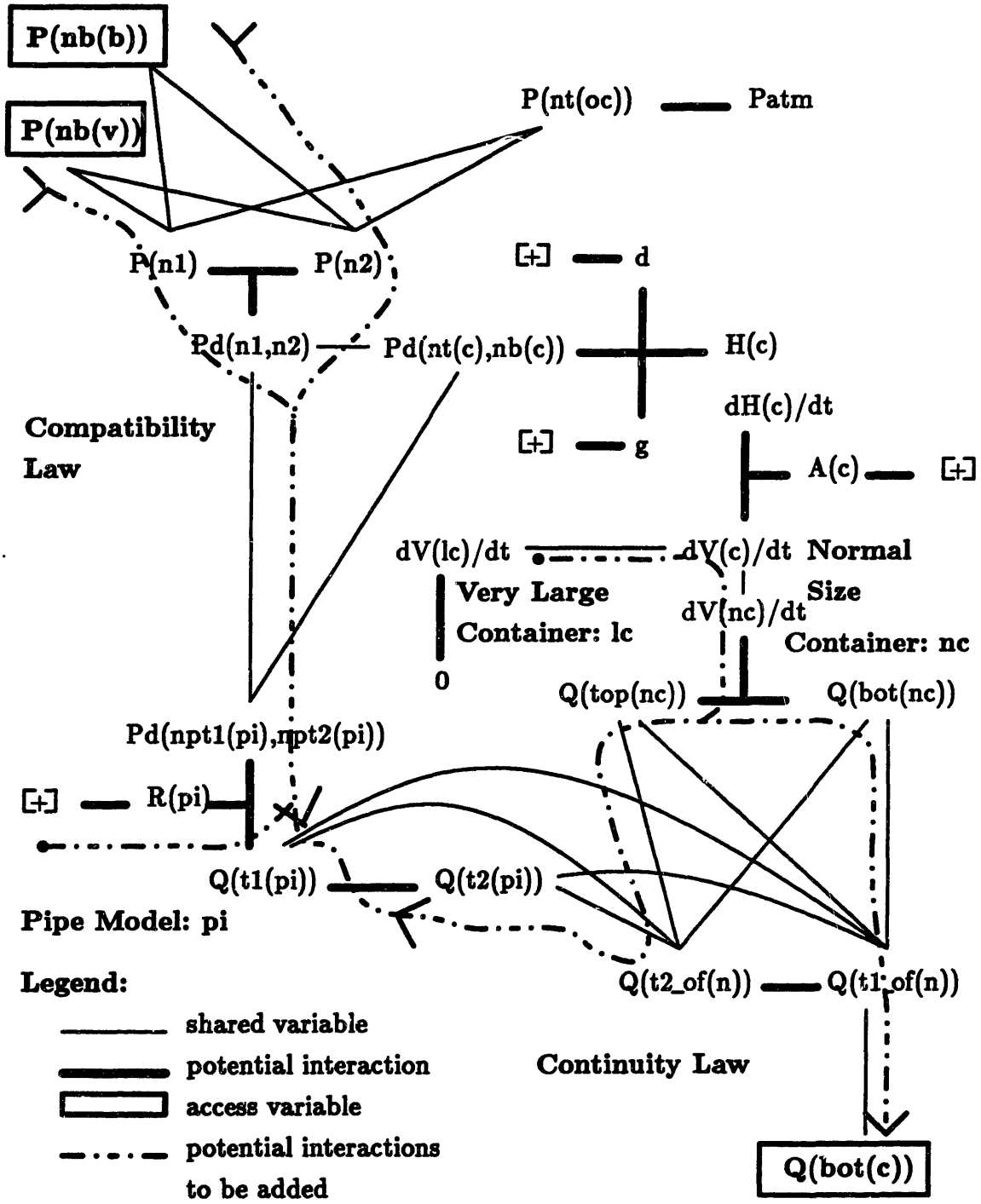


Figure 3.28: A path for the punch bowl problem, resulting in an interaction topology that can't be built.

following equivalences:

$$\begin{aligned} P_{nb(v)} &\equiv P_{N1} \\ P_{nb(b)} &\equiv P_{N2} \\ Pd_{N1,N2} &\equiv Pd_{npt1(P1),npt2(P1)} \\ Q_{t2(P1)} &\equiv Q_{t2\_of(N3)} \end{aligned}$$

and their consequences have been determined: The bottom of the vat and $t1$ of the pipe have been connected to $N1$; the bottom of the bowl and $t2$ of the pipe have been connected to $N2$, and $N2$ and $N3$ are made the same node. That is, the structure, thus far, is equivalent to the solution to the punch bowl problem in the previous section (Figure 3.24).

At this point the example takes a new direction; the path crosses the link from $Q_{t2\_of(n)}$ to $Q_{top(nc)}$ of the normal size container model. To create this link Ibis creates an instance of normal size container (call it $C1$) and introduces the equivalence:

$$Q_{t2\_of(N3)} \equiv Q_{top(C1)}$$

One of the consequences of this equivalence is:

$$N3 \equiv nt(C1)$$

That is, the top of container $C1$ is connected to node $N3$.

But this equivalence is inconsistent. Ibis determined during earlier instantiations that the bottom of the bowl is connected to $N3$:

$$N2 \equiv N3$$

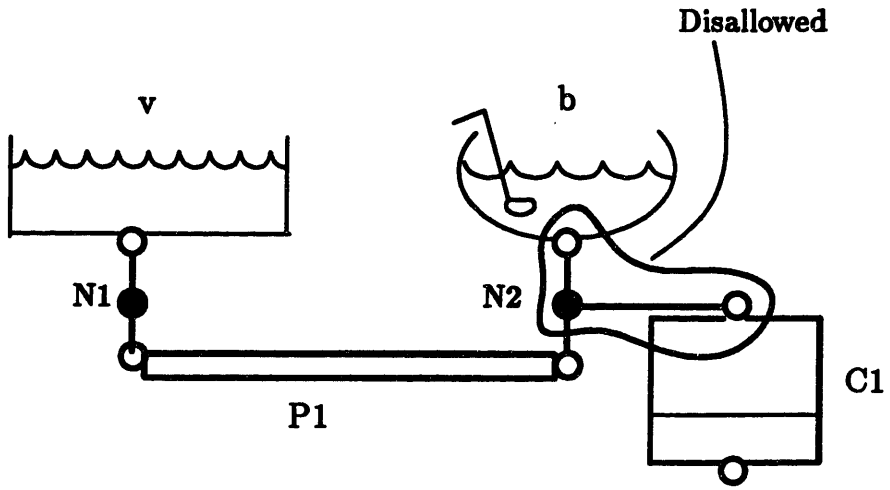


Figure 3.29: Proposed paths sometimes leads to inconsistent structures. In this example, instantiating a path requires the terminals of two containers to be connected together, a restriction on the technology disallows this.

$$N2 \equiv nb(b)$$

and by *definition the terminals of two containers cannot be connected to the same node* – this would require the containers to occupy the same space (Figure 3.29).

The solution as it stands is unworkable, thus Ibis tries altering the path. This first involves removing the source of the inconsistency. The inconsistency is the consequence of four of the instantiated links:

$$\begin{aligned} P_{nb(b)} &\equiv P_{N2} \\ Pd_{N1,N2} &\equiv Pd_{npt1(P1),npt2(P1)} \\ Q_{t2(P1)} &\equiv Q_{t2\_of(N3)} \\ Q_{t2\_of(N3)} &\equiv Q_{top(C1)} \end{aligned}$$

To continue constructing a solution, Ibis breaks one of these links and then tries

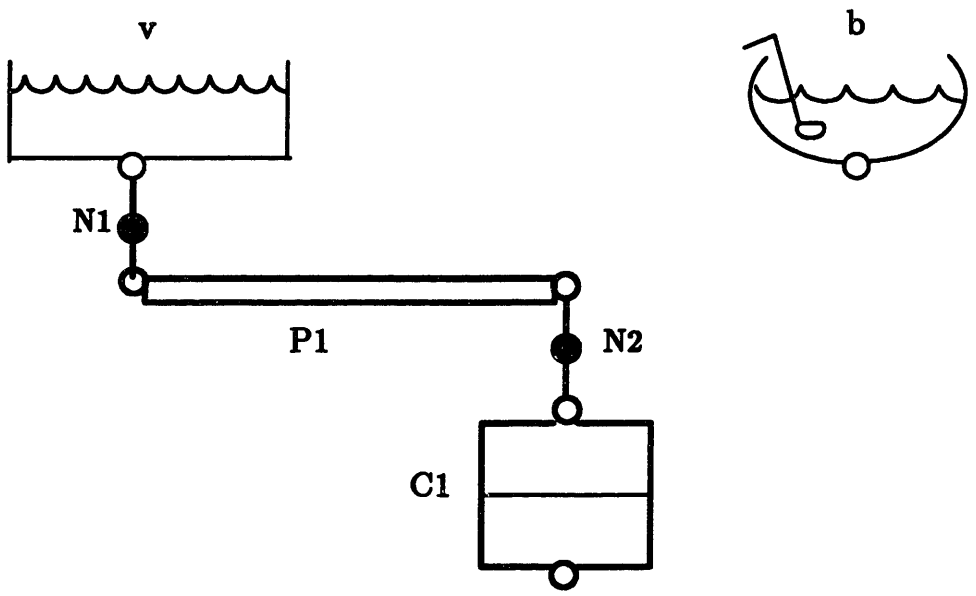


Figure 3.30: The remaining proposed structure after removing an inconsistent link.

to close the gap. Suppose Ibis chooses the first link – which goes from $P_{nb(b)}$ to P_{N2}, of the compatibility law (top left corner of Figure 3.28). Removing this link removes the constraint that $N2$ is connected to the bottom of the bowl (Figure 3.30).

To close this gap, Ibis searches for an alternative path between $P_{nb(b)}$ to P_{N2}. Note that, this new path is allowed to cross the link that just created the inconsistency. For example, in the upper left hand corner of Figure 3.28, the new path can cross over from $P_{nb(b)}$ to P_{n2} of the compatibility law – the link crossed before. The only restriction on this path is that the new instance of P_{n2} has to be different from P_{N2} – the old instance – otherwise the inconsistency is recreated.

This is a crucial point. A link represents a class of equivalences, while an instance of the link represents one of these equivalences. Thus, the fact that one instance of a link produces an inconsistency, does not imply that the other instances are inconsistent. We can use the link in a path as long as, when it is instantiated the instance is not the same as the inconsistent one.

Returning to the example, to avoid recreating the inconsistency while still using this link, Ibis creates a new instance of the compatibility law with two new nodes $N4$ and $N5$. Ibis then continues the search process, moving outward from $P_{nb(b)}$ through this new instance of the compatibility law, where the path branches. The first branch crosses from P_{N4} to P_{N2}, thus closing the gap. The second branch traces a path from $Pd_{N4,N5}$ to $Pd_{nt(c1),nb(c1)}$ and through $C1$'s interactions where the path terminates. After one additional round of refinements, Ibis comes up with a consistent solution. When this process ends, the proposed solution introduces a closed container and two pipes, connected between the bottoms of the vat and bowl. The topology of interactions added, and the physical structure after augmentation is shown in Figures 3.31 and 3.32.

Note that, at this point we have shown that the solution passes the first two tests: It relates the right set of variables and it can be constructed. However, it doesn't necessarily produce the desired behavior. A problem that can get in the way is that the fluid in the container produces a pressure offset (i.e., it is like a battery). If positive, this offset will cause the flow into the bowl to stop before the heights equilibrate – the wrong behavior. Ibis will only accept this solution if the the initial volume of fluid in the intermediate container can be set. Then, to satisfy the desired behavior, this value must be 0 – the container is empty.

Although connecting up this empty container seems a bit absurd, it does satisfy the desired behavior, and thus is perfectly valid. Note that what makes this design absurd is that the closed container serves no useful function. We are currently exploring ways to embody this type of teleological criterion into Ibis.[6]

[6]This criterion is something to the effect that the paths constructed during the coarse solution should be minimal (e.g., they shouldn't contain loops). The initial path for the design just discussed (Figure 3.28) does not meet this criterion, and thus would not be considered.

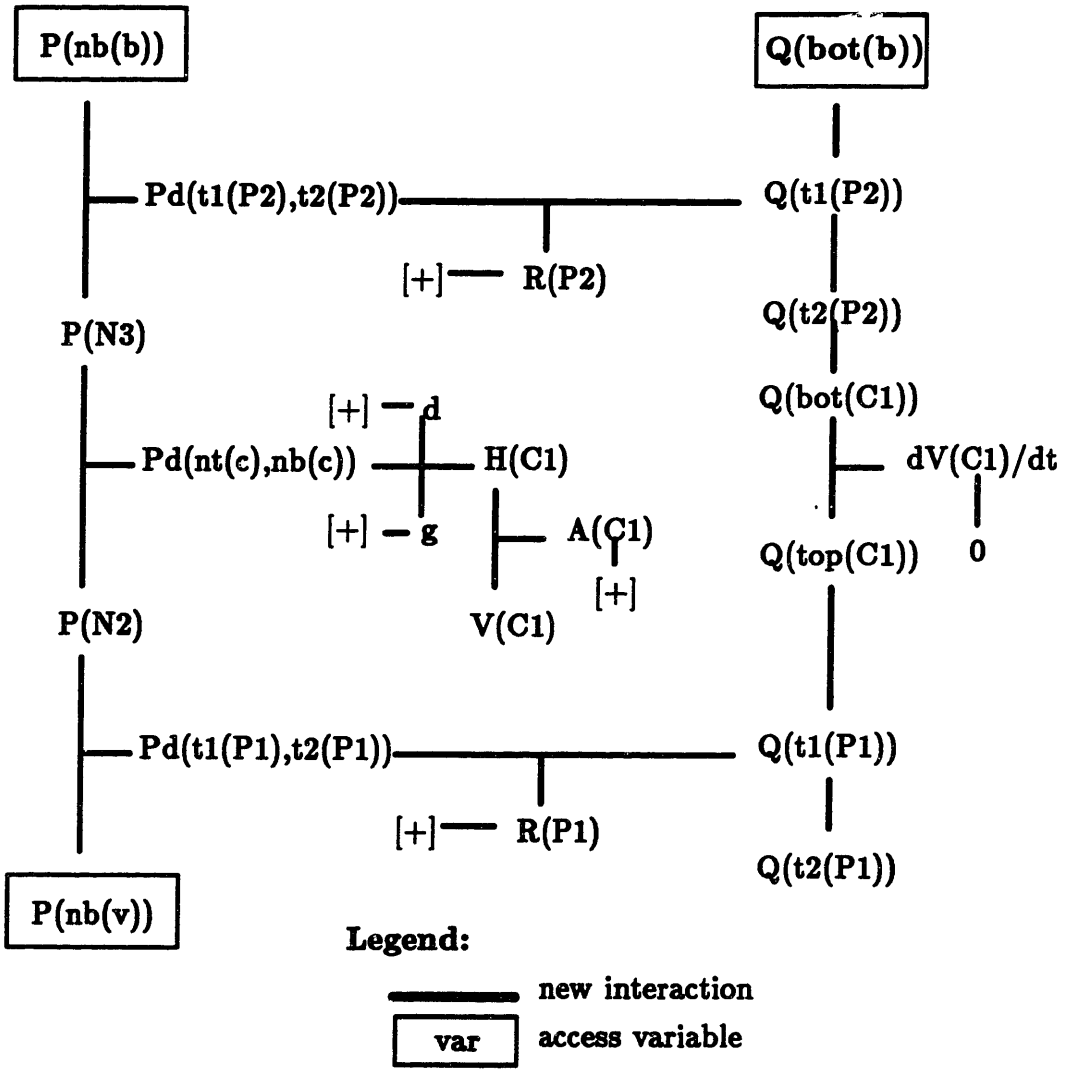


Figure 3.31: Topology of existing interactions, resulting from augmentations to physical structure. This path incorporates an additional pair of pipes and a covered vat into the solution.

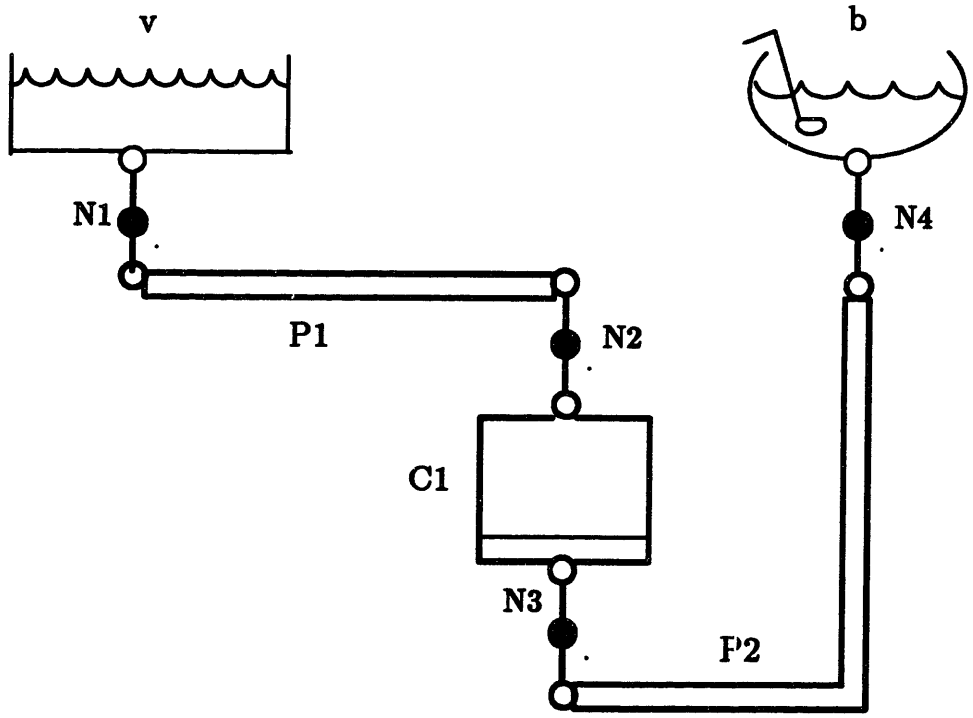


Figure 3.32: Augmentations to physical structure identified after all inconsistencies are eliminated from the initially proposed path.

3.7 Examples of Ancient Inventions

In the last few sections, to demonstrate the major concepts without being lost in the details, we made the punch bowl problem quite restrictive. First, the initial device structure is very constrained: both the vat and bowl are open to the air, connections are only allowed to the bottoms of the containers, and the height of the bowl must reach the height of the vat (as opposed to some other fixed height). Second, the technology is extremely limited; only pipes and containers are available.

Next we consider a variety of devices that can be developed with our approach when some of these restrictions are dropped, and a more extensive technology is supplied. Many of these devices are quite surprising.

Before exploring these examples it is important to understand how Ibis arrives at these designs. For each example Ibis was manually guided through the solution. Consider the rationale for this. As we mentioned earlier, the first implementation of Iota was not powerful enough to make all the necessary inferences about equivalences. As a result, some of these questions had to be answered manually. For extensive examples, like those considered here, answering all these questions manually is too time consuming to let Ibis explore blind paths before arriving at the desired solution. Instead, we intervened at points where Ibis selects between alternative paths, choosing the path that leads to the desired design. The new version of Iota, described in Chapter 8, overcomes these problems. However, until this reimplementation is completed, we have no data on how well the search strategy focuses Ibis to the desired conclusions. It is our intuition that for devices of this complexity, additional focusing techniques will be required.

Given this, the purpose of these examples is to demonstrate how the devices can be accounted for from first principles, and more specifically, how they fit into the interaction-based invention metaphor. The invention of devices of this type and com-

plexity has not been explored previously; thus our work is a substantial contribution towards a theory of invention.

Returning to the examples, the punch bowl problem is similar to the problem of fluid regulation, first explored around 300 B.C. – both problems try to maintain a constant height. A fluid regulator is the key component of a water clock, which during that time was a major focus of kings and inventors. There are records of a variety of these devices[25], which we have used in this thesis to gain insights into invention. Two innovative examples of this type of device were explored in the thesis introduction (Section 1.1). Heron's weight regulator is a device that maintains a constant height of fluid in a container by controlling fluid flow into the container based on its weight. Philon's oil lamp maintains a constant height of oil in the lamp, controlling flow out of the supply vessel indirectly by controlling air inflow. These are two of over a dozen variations developed over the centuries. In this section we use the interaction-based invention metaphor to explore one way these devices might have been constructed, if they had been developed from first principles alone.

To construct these devices the inventors used a wide variety of hydraulic and mechanical components, which provided the inventors much greater flexibility than the limited technology of containers and pipes explored in the last few sections. Many of the designs exploit components like levers, valves, floats, and springs. In this section we supply Ibis with models and laws for these hydro-mechanical devices, and then explore how they are used to construct a variety of these ancient devices. In this section we summarize three designs and their corresponding interaction topologies – one is similar to Heron's weight regulator, a second is similar to an additional invention by Heron called a float regulator, and the last one is similar to Philon's oil lamp.

To begin with, consider the hydro-mechanical models and laws used to capture the domain and technology being explored. The new set of laws and models supplied

include a variety of mechanical, hydraulic and hydro-mechanical devices. Mechanical devices and laws relate force and position; fluid devices relate pressure and fluid flow, and fluid-mechanical devices relate any of these four types of variables. To avoid dealing with vector quantities, position is treated as one dimensional – we model height but not horizontal position.

Available mechanical devices consist of springs, linkages, levers and shelves (e.g., to hold up containers). Hydraulic devices consist of pipes, open and closed containers, and "infinitely" large tanks (e.g., lakes), and devices combining hydraulic and mechanical properties consist of valves and floats. A more extensive model for containers is used that includes mechanical properties such as mass and weight.

Two laws have been added to describe the mechanical properties of connections. These laws are the mechanical analogs of the fluid continuity and compatibility laws. The mechanical continuity law is commonly known as Newton's law for the equilibrium of forces at a point – it says that forces at a point sum to zero.[7] Mechanical compatibility says that height difference can be measured from an absolute reference – this law holds in a Newtonian frame, but not a relativistic one. The interaction topologies for the new models and laws are shown in Figures 3.33, 3.34 and 3.35.

Consider once again the punch bowl problem; we alter the problem in three ways: First, the desired height of the punch bowl is not necessarily the same as the height of the vat. Instead it is a fixed height $H_{desired}$:

$$[H_{desired} - Hd_{nms(b),nmb(b)}] = [-d(Hd_{nms(b),nmb(b)})/dt]$$

Second, connections are allowed to the top of the punch bowl and vat, as well as their bottoms. Third, the vat may be covered or uncovered.

[7]This law assumes that the point has no mass. The colloquial version of this law is "for each action there is an equal and opposite reaction."

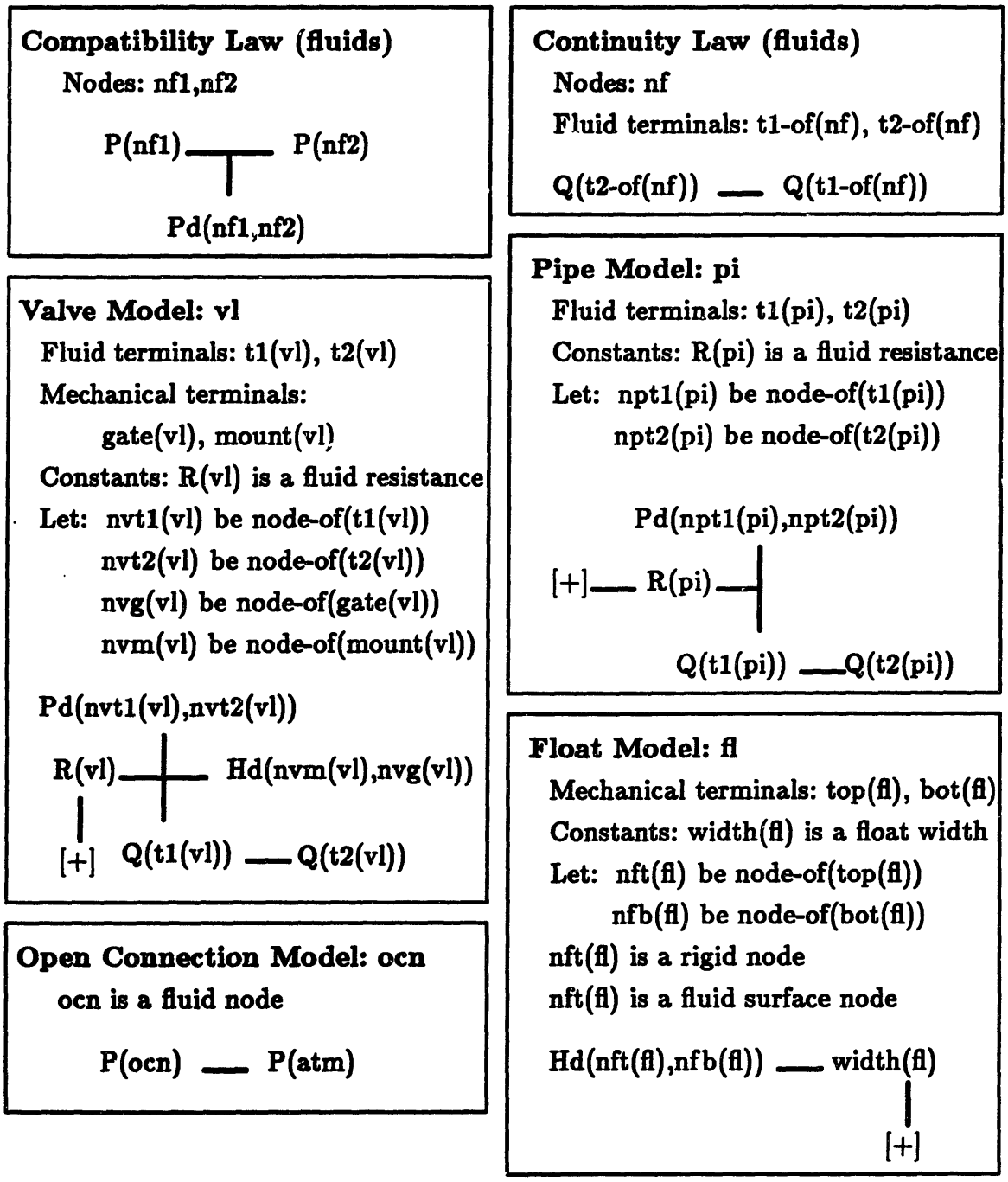


Figure 3.33: Topologies of interaction for models and laws of fluids.

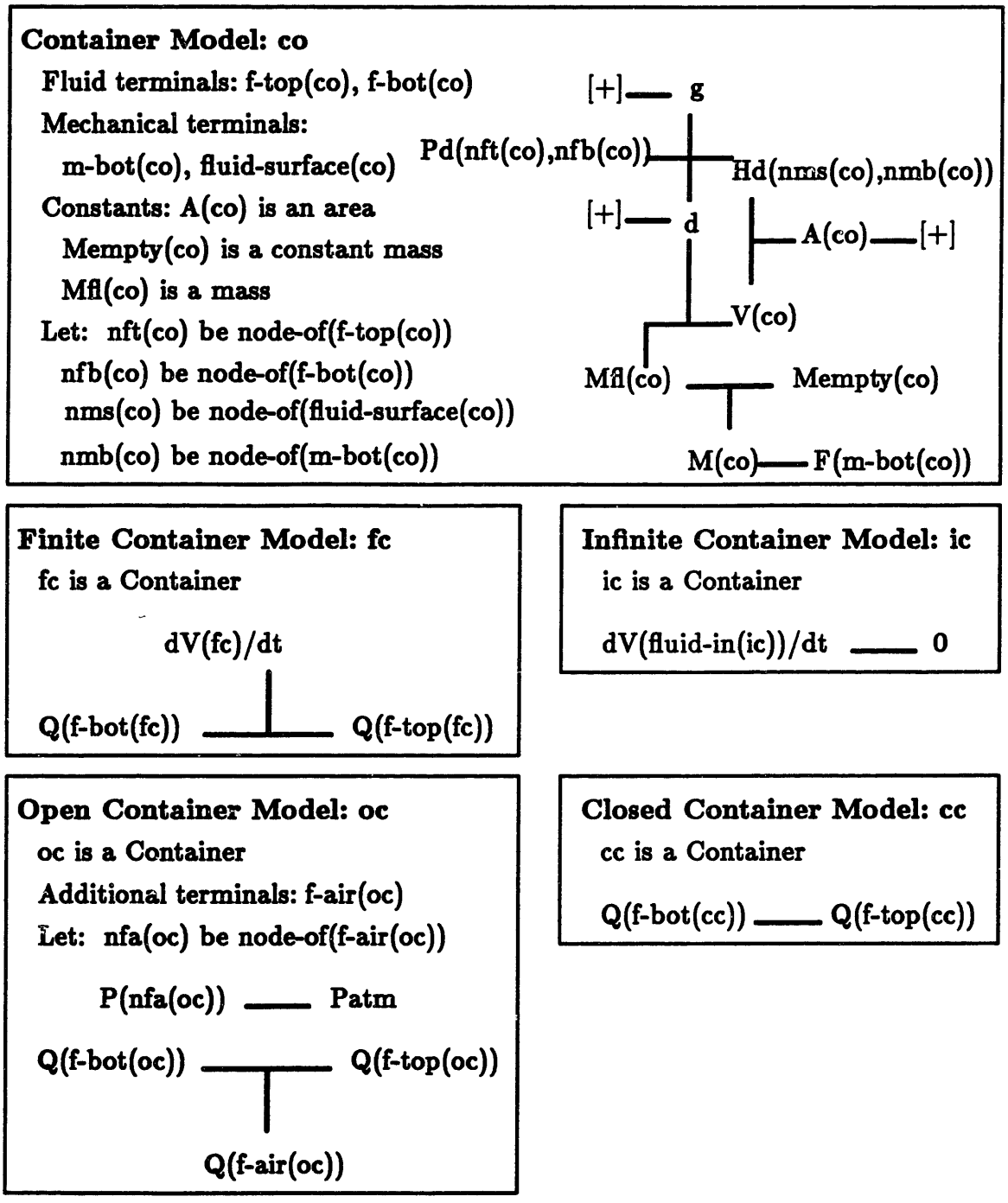


Figure 3.34: Additional topologies of interaction for various fluid container models.

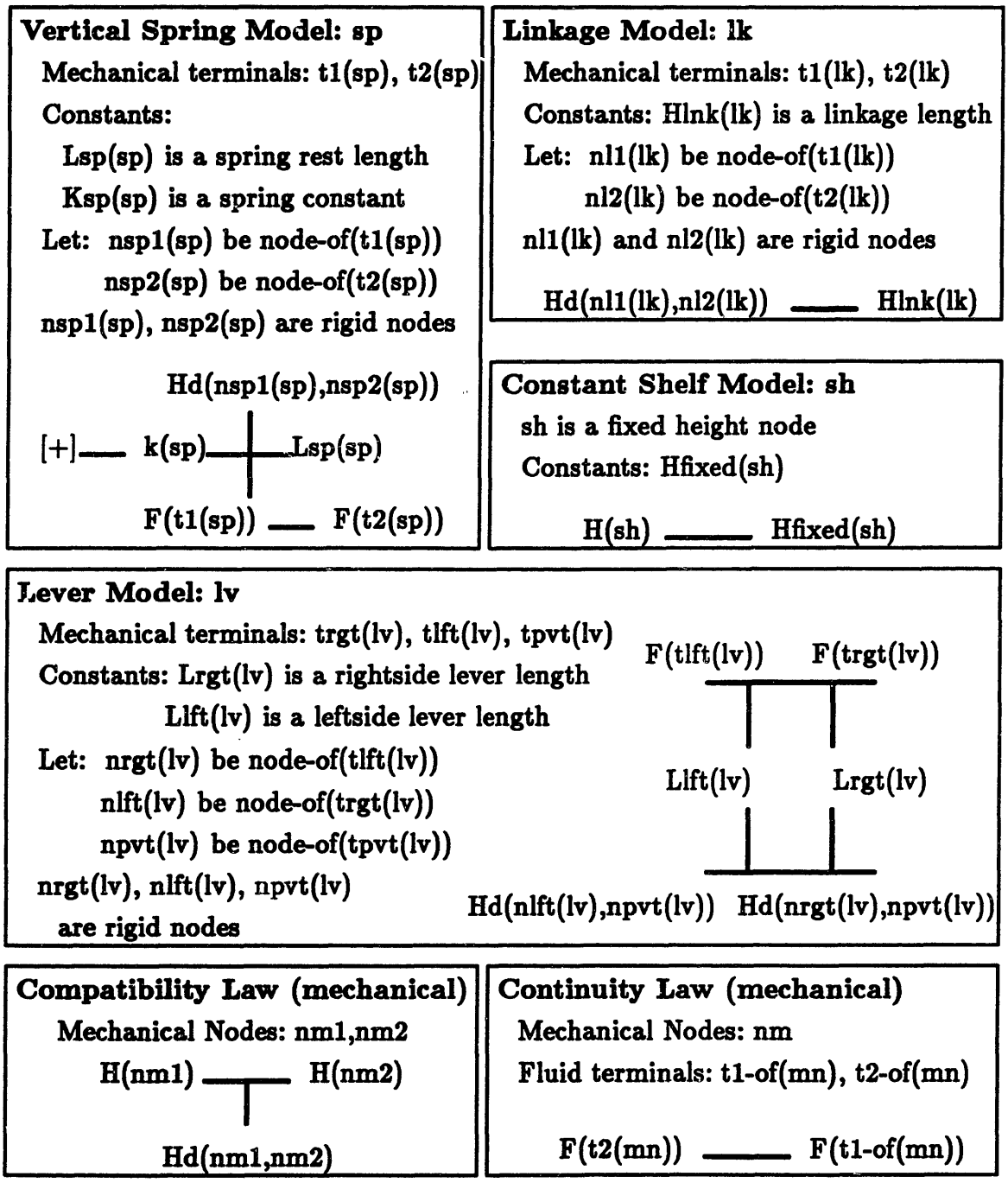


Figure 3.35: Topologies of interaction for models and laws of mechanics.

Using the new models and laws we construct a topology of potential interactions (Figure 3.36) and a topology of existing interactions for the punch bowl problem (Figure 3.37) – both are significantly more complex than the topologies used in the previous section (Figures 3.6 and 3.11).

To solve this problem, as before, we first identify the set of access variables and access paths that allow potential interactions to relate to variables in the desired interaction; this is shown in Figure 3.38. In the previous section, where just hydraulic models were used, fluid height of a container was accessible only through pressure, while change in height was accessible only through fluid flow. Here fluid height difference is accessible three ways: through height, by pressure, or through force at the bottom of the container. Change in height is also accessible three ways: by regulating either fluid flow at the top or bottom of the container, or air flow into the container top.

Each alternative way of accessing the variables in the desired interaction provides an opportunity for novel designs. For example, we observed in the introduction (Section 1.1) that much of what made Heron's and Philon's designs innovative was the path of interaction used to relate fluid height and height difference. The following three designs exploit many of the new ways of accessing variables in the desired interaction.

The first design, shown in Figure 3.39, senses height directly using a float, and then controls change in height by fluid flow into the top of the punch bowl. The linkage and valve provide the interaction between fluid height and fluid flow, while the vat provides a source of pressure and fluid flow; the interaction topology for this device is shown in Figure 3.40. This design is similar to Heron's weight regulator, discussed in the introduction chapter. The primary difference is that weight is sensed using a coil spring, rather than a balance beam.

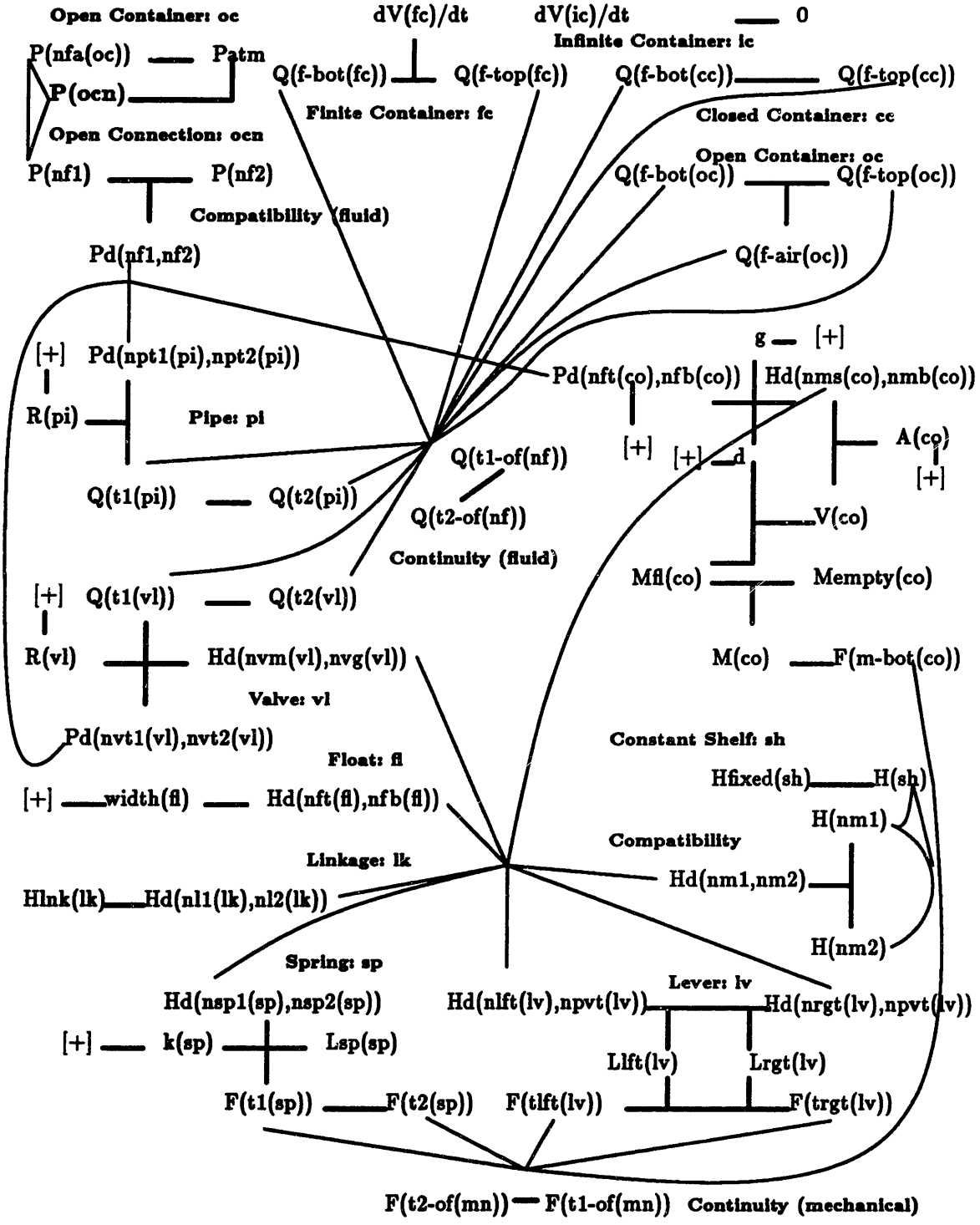


Figure 3.36: Topology of potential interactions for the hydro-mechanical models and laws.

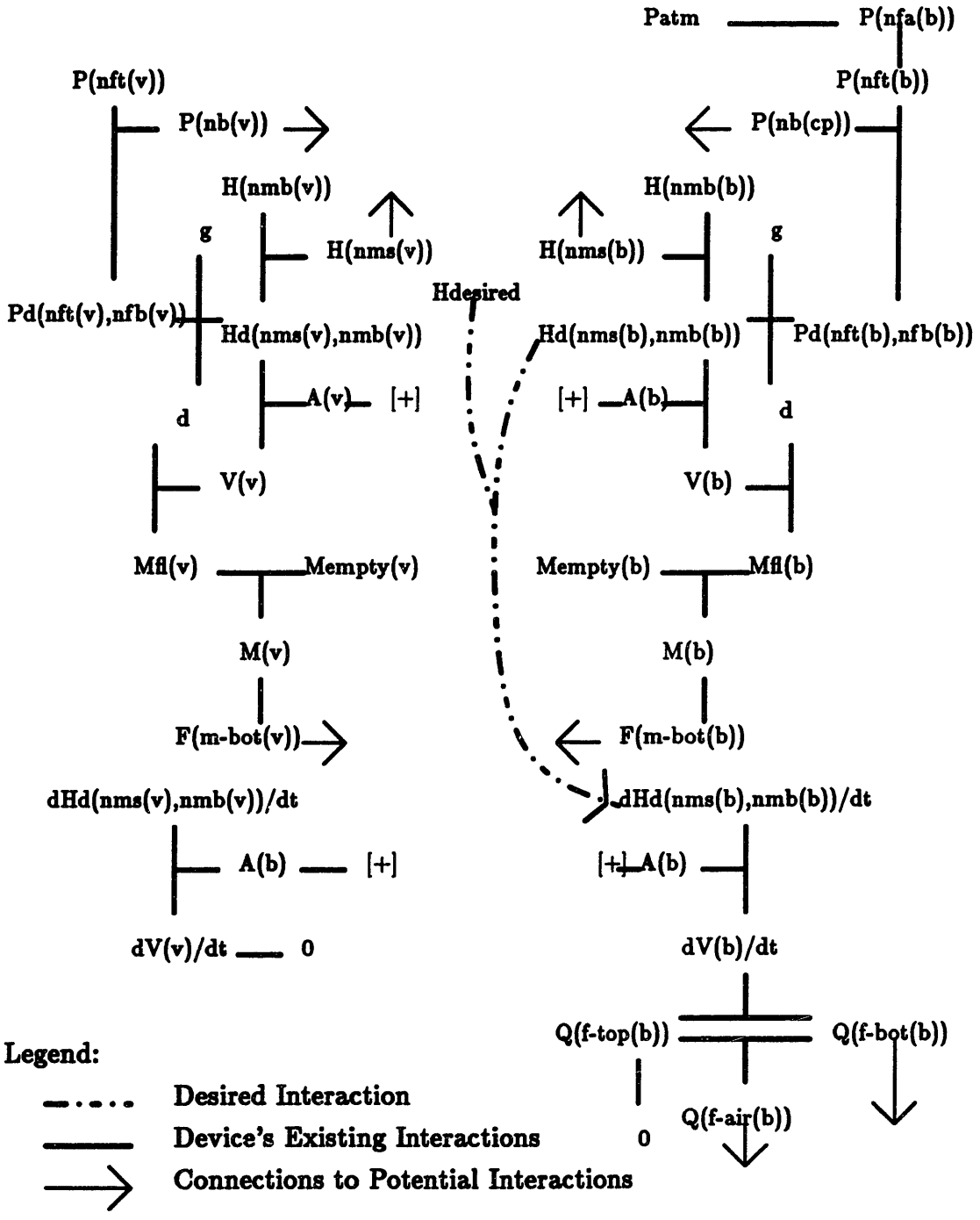


Figure 3.37: Topology of existing interactions, using hydro-mechanical principles, for the initial specification of punch bowl problem.

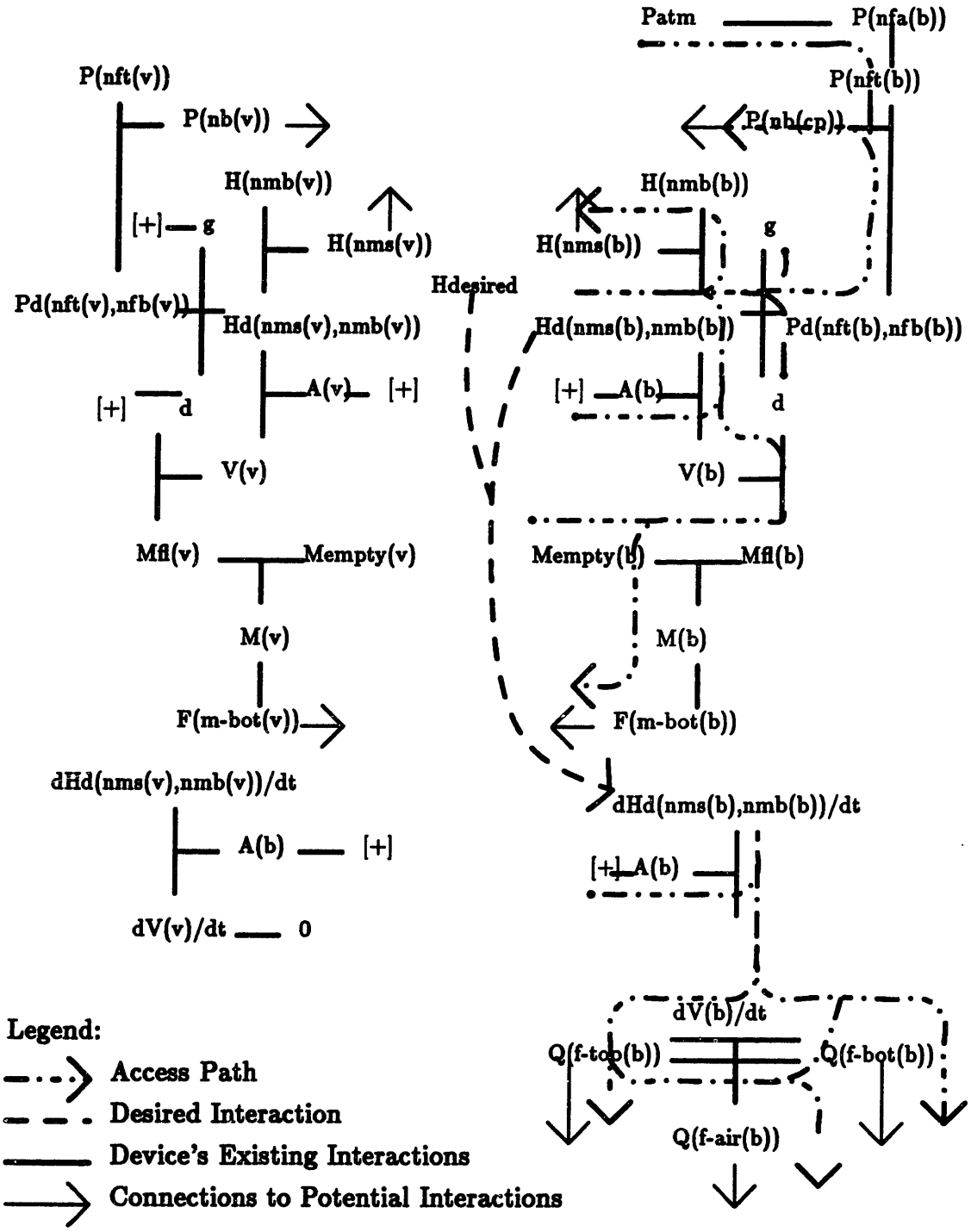


Figure 3.38: Identifying places to inject new interactions for the punch bowl problem using hydraulic and mechanical properties.

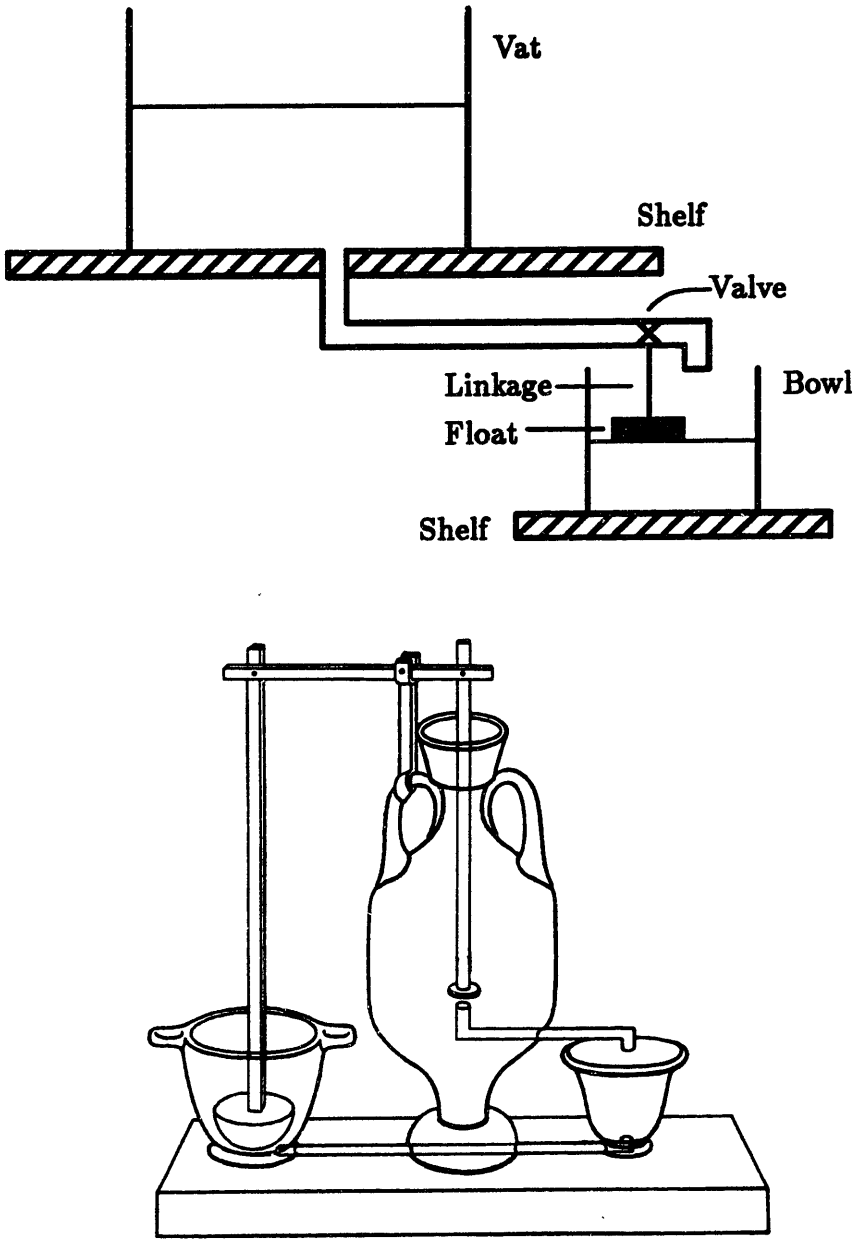


Figure 3.39: At the top is hydro-mechanical design 1, proposed to solve the punch bowl problem. This device is roughly analogous to Heron's float regulator, shown at the bottom.

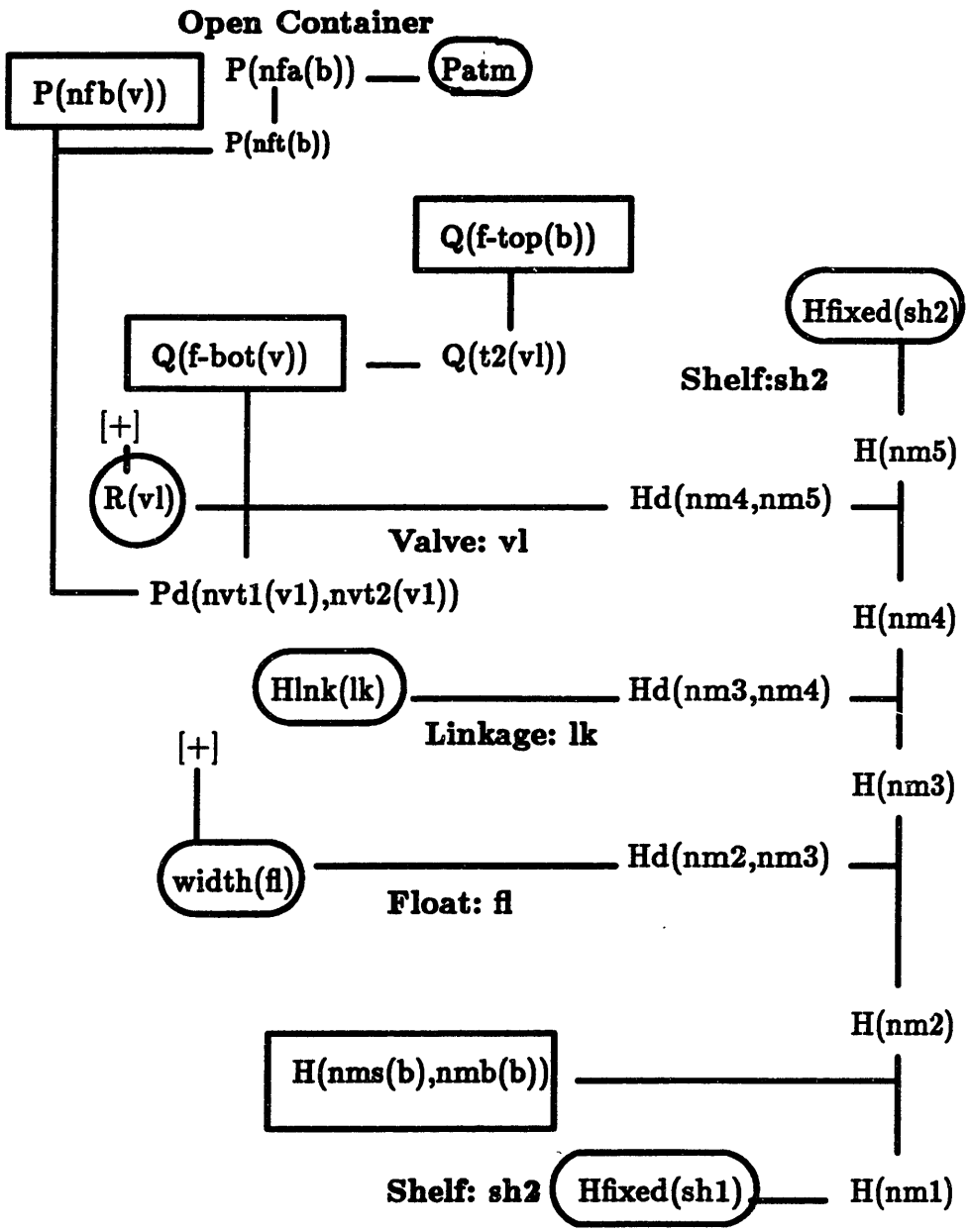


Figure 3.40: Augmentations to interaction topology for hydro-mechanical design 1, proposed to solve punch bowl problem. Boxes indicate access variables and circles indicate constants.

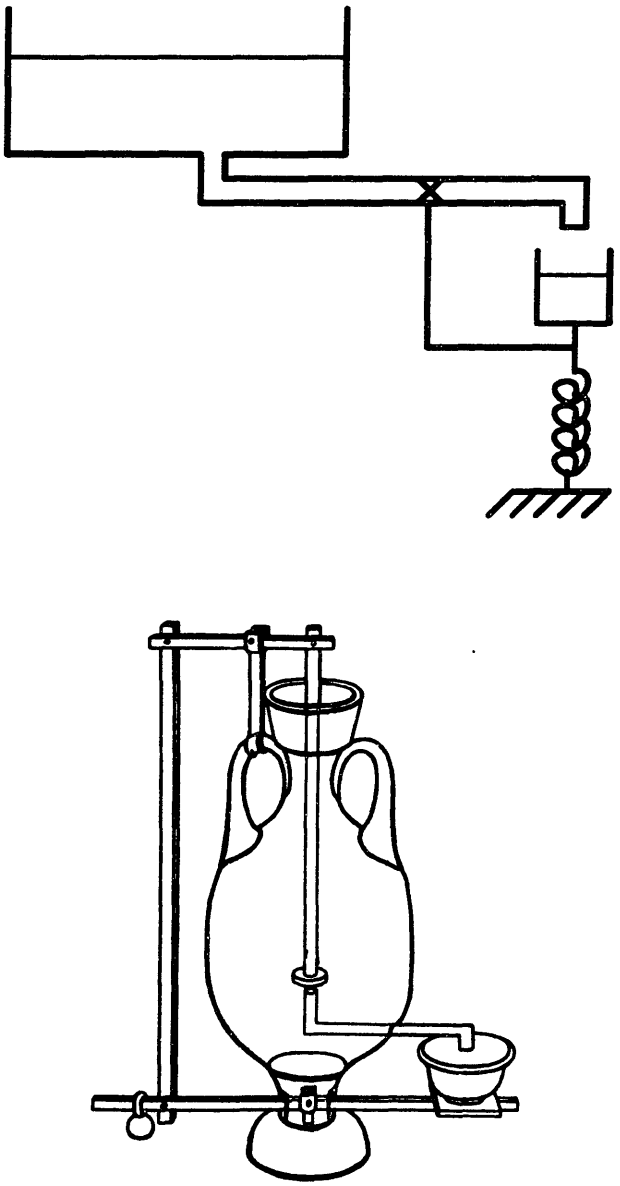


Figure 3.41: At the top is hydro-mechanical design 2, proposed to solve punch bowl problem. This device uses the same concept of weighing the container to sense fluid height, used in Heron's weight regulator (shown at the bottom).

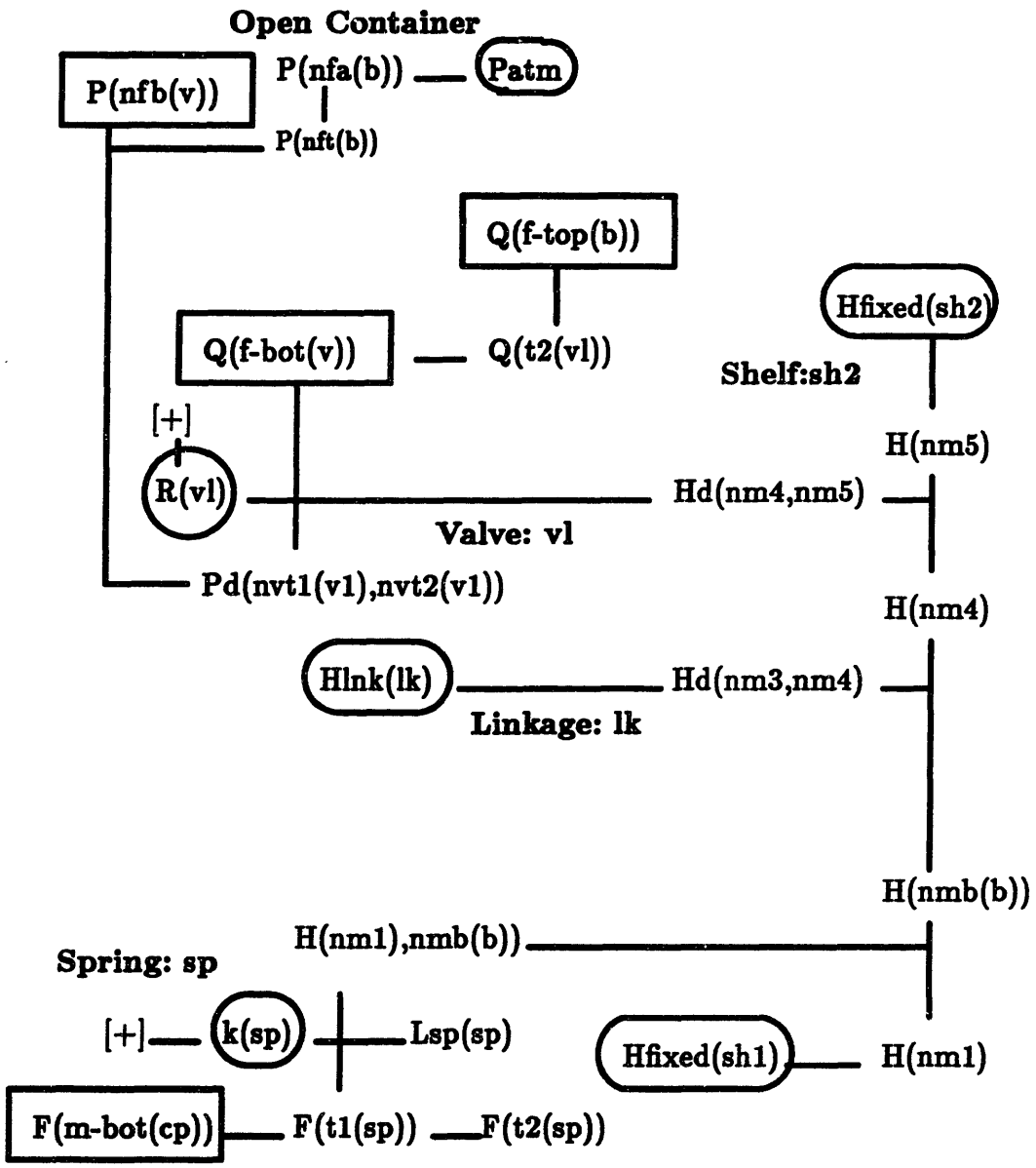


Figure 3.42: Augmentations to interaction topology for hydro-mechânical design 2, proposed to solve the punch bowl problem. Boxes indicate access variables and circles indicate constants.

The second design, shown in Figure 3.41, senses height indirectly using the force applied downward by the punch bowl. This force is due to the weight of the volume of punch in the container and is proportional to fluid height. As in the previous design, change in fluid height is controlled using a valve to regulate the flow of punch into the top of the container. The spring under the container transforms downward force into height, while a linkage attached to the spring uses this height to adjust the valve position; the interaction topology is shown in Figure 3.42.[8] This design is similar to one developed by Heron of Alexandria, believed to live in the first century A.D.[25].

The third design, shown in Figure 3.43,[9] is similar to the first design in that it senses height using a float and regulates change in height through fluid flow into the top of the container. What is novel about the design is the way it regulates fluid flow. Instead of directly controlling fluid flow coming *out of* the vat with a valve, the valve is used to control air flow going into the tank. This indirectly controls the fluid flow out of the tank since the tank is covered, and thus flow in and out of the tank must balance. The basic principle of regulating fluid outflow by controlling air inflow is also the basis of Philon's oil lamp.

To recap, using the concept of interaction-based invention, we have guided Ibis through the design of three devices that embody important innovations in ancient devices invented by Philon and Heron between 300 B.C. and 100 A. D.. Because of these devices and the specific innovations we have explored, these inventors are considered to be the fathers of modern feedback control.

[8]Note that to simplify the treatment of the mechanical continuity law we assumed that the downward force of the linkage onto the spring is zero. Specifically, to avoid introducing summation over sets we restricted the law to two forces. In this example there are three forces at a point. We eliminated the linkage force by assuming it is zero.

[9]At the time of this writing we have only partially walked through this example using Ibis, although it has been explored by hand. We do not believe it should present any problem.

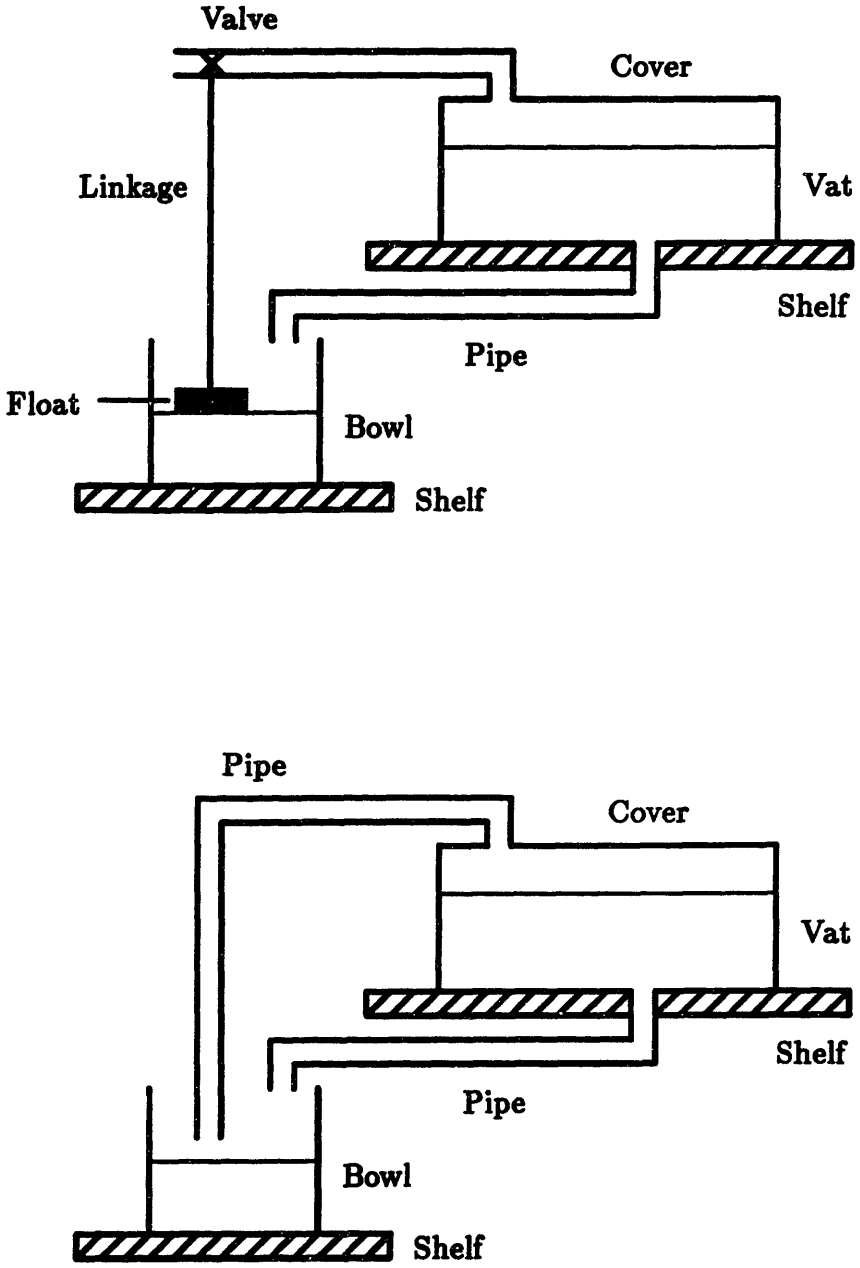


Figure 3.43: At the top is hydro-mechanical design 3, proposed to solve the punch bowl problem. This device uses the same concept as in Philon's lamp (shown at the bottom) of regulating all outflow indirectly through air inflow. The difference between the two devices is that our design uses a float to sense height, and a valve to control air inflow, while Philon's lamp performs all these functions with a pipe.

3.8 Path Tracing Criteria: Rationale and Limitations

This section has two objectives: First, to provide a rationale for the criteria used to determine valid paths when constructing a coarse solution (Section 3.5.2). Second, to pinpoint limitations to the criteria.

3.8.1 Rationale for the Path Tracing Criteria

Recall from Section 3.5.2 that a coarse solution is constructed by tracing a path through the topologies of existing and potential interactions. Furthermore, *when constructing a path, every branch must terminate on a variable in the desired interaction or a physical constant.* Consider why this is a reasonable approach.

Recall that, to be a solution, the interactions along a path must combine to produce the desired interaction. For example, if we canonicalize the equation describing the composite of these interactions, then it should produce the same equation as that describing the desired interaction. This requires that the composite interaction contain only those variables mentioned in the desired interaction. A simple way to insure this is that none of the paths terminate on variables other than those in the desired interaction.

It is important to note that this approach doesn't prevent the paths from terminating on physical constants not contained in the desired interaction. The reason this is allowed is that physical constants can typically be set to any value desired (e.g., the area of the vat's bottom). Thus, they can be made to have values that are consistent with constants in the desired interaction.

For example, suppose for the moment that the desired interaction for the punch

bowl problem was specified as:

$$H_v - H_b = dH_b/dt \times C$$

where C is some constant (indicating how fast the level should rise). The difference between this interaction and the one worked with earlier is that this interaction is quantitative, while the earlier one was qualitative. Also suppose we use the solution we explored earlier of attaching a pipe. The composite of the interactions produced by this solution is:[10]

$$H_v - H_b = dH_b/dt \times (A_b \times R_{pipe}/(d \times g))$$

To make this composite interaction satisfy the desired interaction we introduce the constraint:

$$(A_b \times R_{pipe}/(d \times g)) = C$$

This type of equation constrains the values of parameters selected for the components of the design, in this case punch bowl area and pipe fluid resistance.[11]

Conceptually, introducing this equation adds an interaction that produces a path from each of the constants – A_b, R_{pipe}, d, and g – that were terminated on, to a constant appearing in the desired interaction – C. Thus, by adding this interaction all the paths terminate only on variables and constants in the desired interaction.

This type of interaction doesn't appear explicitly in the potential interaction topology since the topology only contains interactions produced by augmentations to physical structure (i.e., interactions in the models and laws). This interaction

[10]The process of constructing this composite interaction is identical to that discussed in Section 3.5.5. In this case the expressions that are propagated are purely quantitative.

[11]It is not necessary to be able to set the values of all constants, only that enough constants can be set so that the constraints don't overconstrain their values.

doesn't represent an augmentation to structure, and is not explicit in the models and laws.

Furthermore, it doesn't make sense to add this type of interaction to the potential interaction topology, because we don't know anything about what it looks like *a priori*. This type of interaction can involve any subset of the constants mentioned in the models. Instead, we get the desired effect, while leaving these interactions implicit, by allowing paths to terminate on constants.

To summarize:

> To construct the coarse solution, a path is constructed between the variables of the desired interaction, through a set of variables, such that each branch of the path terminates on one of the variables in the desired interaction or a physical constant.

3.8.2 Limitations of the Path Tracing Approach

We refer to the above approach as aggressive, because there are some solutions, albeit unusual ones, that it excludes – the approach is incomplete. More precisely, it is possible for there to be solutions where the corresponding path has a branch terminating on a variable not mentioned in the desired interaction. Consider why this is the case. The fact that a branch terminates on a variable means that the variable appears in only one interaction. The fact that the path is a solution means that this variable is somehow eliminated when the interactions are combined.

Consider a trivial example. Suppose we have a desired interaction:

$$[x] = [y]$$

and we construct a path which goes through exactly one interaction:

$$[x + (b \times z) - (c \times z)] = [y]$$

where x and y, are variables, and b and c are constants. In this case a branch of the path terminates on z, which is not in the desired interaction. Furthermore, by introducing the following interaction between the constants:

$$b = c$$

the composite interactions simplify to the desired interaction. That is, substituting b for c we get:

$$[x + (b \times z) - (b \times z)] = [y]$$

which simplifies to:

$$[x] = [y]$$

the desired interaction. Thus, it is theoretically possible for solutions to be missed if we disallow paths terminating on variables not specified in the desired interaction.

A more conservative approach that avoids this incompleteness is to allow paths to terminate on any variable even if not specified in the desired interaction, while hopefully trying to keep this down to a minimum. This avoids the possibility of incompleteness, but will probably introduce many spurious candidates. There are, of course, more restrictive yet conservative approaches based on specific cases where these single variables cannot be simplified away. A trivial one is not to terminate on a variable (not in the desired interaction), if it appears only once in an interaction. Which approaches are most appropriate is a topic for future research.

3.9 Limitations and Future Work

How well does the invention strategy presented in this chapter stand up to the robustness and performance criteria? That is, to what extent is the strategy sound, complete and efficient? To answer this we analyze the strategy in terms of its basic constituents:

- the description of the search space,
- the search strategy for constructing a coarse solution,
- the process of elaborating the coarse solution,
- the test for consistency, and
- the test for whether the proposed solution produces the desired interaction.

Going through each constituent in detail requires a lengthy discussion. Here we summarize what we consider to be the most important results. The invention strategy stands up pretty well under an informal analysis, although a formal analysis is required. The topologies of potential and existing interactions are complete, relative to the first principles supplied, and to the extent that Iota can determine all pairs of variables that can be shared. In Chapters 6 and 8 we argue for the efficiency, soundness and completeness of the basic tools – Iota and Minima – used by Ibis to perform all the basic operations of the invention strategy. While some work remains, it is our belief that achieving completeness and proving it formally are well within reach.

There are four major areas of the invention strategy which require extensive future work: modeling, verification of proposed solutions, efficiency of the search strategy, and expressivity of languages for behavior and structure.

The modeling issue concerns the soundness and completeness of the models and laws that characterize the physics, and is outside the stated scope of the thesis. However, the models establish the only link between physical structure and behavior; thus, their soundness and generality directly impact the soundness and generality of the overall design approach. The models used during design are crucial to how the inventor perceives the devices he is constructing, and the features of these objects he focuses on. In addition, modeling and design are integral tasks – a good designer continually tests the limits of the models he is working with, and is continually changing them based on details of component behavior that a particular device highlights. Thus the role of modeling with respect to invention is an important line of future work.

The second issue affects completeness, and is the ability of the system to determine that a proposed solution implements the desired behavior. This is called the design verification problem, and is an open research topic in general. Our verification task hinges on the properties of the Q1 qualitative algebra. This thesis is one of the first efforts to characterize the formal properties of qualitative algebras, and to embody them in a symbolic algebra system. In addition, it is the first work to pursue these goals for a hybrid qualitative-quantitative algebra. The verification task consists of determining that a device's interaction topology produces a desired interaction. That is, the composite of some of the interactions in this topology is equivalent to the desired interaction. This task has two parts: Identifying a set of interactions to be composed and performing the individual compositions. Our analysis of qualitative algebraic manipulation has focused on pairwise compositions. The technique used for selecting the set of interactions to be composed is heuristic. Developing a complete verification procedures is a topic of future work.

The third factor is the efficiency of the search strategy. In Section 2.4.3 of the last chapter we argued how our task is a superset of satisfiability for first order logic. Thus a complete and efficient strategy is unlikely. Our approach is based on informal

arguments about the major costs in the search process, together with ways of cutting these costs, based on insights drawn from historical analysis. These arguments need to be backed by extensive empirical data. Furthermore, the search strategy described is only a first cut solution. We expect it to evolve into a more sophisticated and guided approach. The details highlighted in the topology of potential interactions are similarly a first cut solution. No doubt additional features will need to be introduced as Ibis moves towards more sophisticated behavioral and structural representations. At this point we view Ibis primarily as an experimental test bed for identifying major computational costs incurred within the metaphor of interaction-based invention.

Thus far our practical experience with Ibis is limited. As was mentioned earlier, this is due to weaknesses in the current implementation. In Chapter 8 we discuss certain limitations with the initial implementation of Iota together with detailed solutions to these limitations. We are currently in the process of implementing them. Once this is finished we will be able to pursue more extensive tests.

The fourth factor is the expressivity of the behavioral language. This can be broken into the language for describing the relationships between quantities, and the language for time-varying behavior. We have explored one language for relating quantities ($Q1$). In Chapter 5 we argue its appropriateness for describing quantity shifting devices, and have demonstrated it on examples in three domains: fluid, mechanical and electrical devices (only the first two are discussed here). Furthermore, $Q1$ includes the algebra on the reals, and thus can be used to design devices that are described by systems of linear or non-linear equations. For example, Ibis can be used to design devices described at a quantitative level, such as amplifiers. There are a variety of devices whose behavior cannot be described by $Q1$, such as those described by transcendental or complex equations (e.g., radio circuits and filters). In addition, $Q1$ may not be able to capture, at the right level of abstraction, some classes of devices that can be described by nonlinear equations.

Next consider time-varying behavior. Section 1.8 of the introduction, described our more general perspective on device behavior as follows: In order for a continuous device to work, it must orchestrate a set of competing influences that combine to move a quantity in a particular direction. A device's behavior changes over time by strengthening, weakening, adding or eliminating interactions during windows of time. Thus a device's interaction topology can be imagined as a tapestry of dynamically changing interactions.

In the current strategy of tracing topologies of interaction, the temporal component of behavior is described simply as state transitions, and this temporal component is abstracted away from a problem during the first stage of the design process. It is our belief that the temporal component of interactions plays a more integral role in the design process, which involves explicitly constructing this dynamic tapestry.

To this end we have developed the concise histories representation[59] which compactly describes the dynamics of devices through temporally bounded, local interactions. This representation is used by the evolutionary design strategy to pinpoint changes to the interaction topology that will improve a device's dynamic performance, and is central to the success of the approach. It is crucial that this representation be integrated into the process of constructing new parts of interaction topologies as well.

The last factor is the expressivity of the language for structure. The use of a lumped element model allows Ibis to focus on innovations resulting from compositions of primitive components that work in novel ways. This model has proven appropriate for accounting for many of the innovations arising in our ancient fluid regulation examples (e.g., Heron's float and weight regulators, and Philon's lamp). However, there are many additional innovations that cannot be captured using a lumped element model. Instead these require a more sophisticated spatial model of structure. For example, two innovative aspects of Philon's lamp (Figure 1.1) where

sophisticated spatial reasoning is required are: the use of capillary tubes to restrict fluid flow in a downward direction only, and the use of a pipe end together with the oil surface to implement a valve.

3.10 Summary

In this chapter we sketched an approach that uses the interaction topology to focus the process of invention from first principles. With this approach we are able to construct several substantially different solutions to the fluid regulation problem. Each solution produces its behavior through a different topology of interactions – our criteria for innovation – and three of the devices invented are analogous to classic examples of early invention.

This approach is based on the conjecture that constructing new innovative devices is similar to understanding existing inventions. To invent:

- Visualize all possible interactions producible in the current technology.
- Identify interactions contributing to the desired behavior.

The first part of this chapter presented the representation that allows Ibis to "visualize" the space of interactions – a graph combining topologies of potential and existing interactions. The graph focuses Ibis on innovative devices through the following properties:

- *It is a good abstraction.* It highlights salient features – what makes composing component interactions difficult. And the graph is easy to search – it is compact and is searched quickly through path tracing.

- *It is a space of interactions.* Thus it focuses on what makes a device work, which is of primary importance, as opposed to structure, which is simply the means to this end.
- *It makes evident the physical structure.* This is because the topology is built so that each interaction came from a type of component or connection.
- *It is a topology.* It tells us not only what interactions are producible, but how they can combine.
- *It highlights innovations.* Different paths produce innovative alternatives, since their topologies are different.
- *It captures first principles.* The graphs are constructed from the primitive topologies of all the component models and connection laws. These are the first principles.

These last two properties are what satisfy our claim of doing innovative design from first principles.

The second half of the chapter presented the design strategy that uses this representation to guide interaction-based invention. Called tracing topologies of interactions, it performs the following steps:

1. Construct the graph (the two topologies)
2. Trace a path between variables in the desired interaction.
3. Build the device's interaction topology and physical structure.
4. Check the consistency of the structure.
5. Check that the desired interaction is produced.
6. Refine the solution when necessary.

To construct quantity shifting devices, Ibis couples the strategy discussed here with the first two transformation stages discussed in Chapter 2.

Basic to this approach, and interaction-based invention are the procedural skills used to connect together the interaction topology, and to show that its composite behavior produces the desired interactions:

- Interactions are connected together through constraints on physical structure. Determining when and how these constraints produce desired connections, requires reasoning about the relations imposed by definitions.
  (*Iota*)

- Determining that an interaction topology produces the desired behavior involves composing and comparing interactions – a qualitative algebraic process.
  (*Minima*)

This completes our discussion of the basic strategy used by Ibis to construct simple, innovative devices from first principles. The success of Ibis critically depends on, not only the design strategies, but also the representations, principles and procedural skills used. These are discussed in the second part of this thesis, after a review of the central claims of interaction-based invention, and an analysis of related work in design.

Chapter 4

Conclusion

4.1 Contributions

The goal of this thesis has been to capture important elements of a theory of invention – specifically the design of innovative devices from first principles of physics. To motivate this goal, we argued at the beginning of this thesis that existing library-based approaches to design are not powerful enough to handle modern design tasks. Designs proposed by these approaches often fail to meet the specifications, typically because they have poor performance (e.g., they are too slow or use too much power).

Further, we claimed that these systems are inadequate exactly because they can't innovate. They need to be able to exploit new and existing technologies in novel ways. And this often requires reasoning directly from the first principles of physics that characterize these technologies.

Finally, we concluded that, because of the rapid rate of change of modern technologies, being a competent designer requires being able to innovate. Thus invention – specifically the process of constructing innovative devices from first principles –

is a crucial component of a robust theory of design. As a consequence, the central question of this thesis is:

> How might an inventor use his understanding of first principles of physics to develop innovative devices?

This in turn raises the following list of subsidiary questions:

- What is an innovative device?
- What light is shed on invention?
- How do we maximize the breadth of the inventions discovered?
- How can simple devices emerge from first principles?
- And how might complex devices evolve?

To gain insight into these questions we have argued the importance of viewing the art of design from a historical perspective. That is, it is valuable to understand the way ancient innovative devices, like Heron's weight regulator and Philon's lamp, might have arisen. The rationale is that we are interested in how innovations could arise from first principles, and at the time these ancient devices were developed, little else was known but these principles. Using this perspective has provided the following insights:

What is an innovative device?

- One way a device is innovative is if it works in a fundamentally new way.
- How something works is captured by the topology of interactions between quantities.
  (*the Interaction Topology*)

- Differences in the topology of interactions make evident fundamentally different ways of producing a desired behavior.
- Thus a device is innovative if its topology of interactions is distinct.

What light is shed on invention?

- How a device works, and thus the interaction topology, is the focus of invention. The physical structure is the means to this end.
- Invention involves constructing a topology of interactions that both produces the desired behavior, and makes evident a topology of physical devices that produces it.
  (*Interaction-based Invention*)
- To focus on innovative alternatives, we must concentrate foremost on structural differences between interaction topologies.

How do we maximize the breadth of the inventions discovered?

- This arises from using the finest grained structures, and by not presuming their purpose. To achieve this we reason from fundamental principles of physics.
  (*Design from First Principles)*

How can simple devices emerge from first principles?

We construct new innovative devices in a manner similar to how we understand existing inventions:

- Using the first principles, visualize a topology of all interactions producible and all ways they interrelate.
  (*The Potential Interaction Topology*)

- Identify, by tracing a path, those interactions contributing to a desired behavior. (*Tracing Paths of Interaction*)
- Use distinct paths to propose innovative alternatives.

Through Ibis we have demonstrated the plausibility of this approach for the design of simple hydro-mechanical devices, whose behavior is described as quantity shifts, and whose structure is described as components and connections. In addition, using Ibis we are currently exploring the answer to the remaining question:

How might complex devices evolve?

We conjecture that they evolve through a sequence of focused innovations, directed towards higher performance. At each step the following actions occur:

- Push a leading edge design for performance until it fails.
- Identify critical interactions that lead to this failure.
- To improve performance, alter this subset of the device's interaction topology, using the above technique.

4.2 Related Design Work

The work developed in this thesis draws substantially upon and contributes to a wide body of AI research in problem solving, temporal and causal reasoning, qualitative physics, theorem proving, and knowledge representation. Much of this research is analyzed extensively throughout the document. In this section we focus on work that addresses the task of design directly.

Ultimately, the goal of a theory of design is to be both robust and efficient. As we discuss the related work it is worth keeping in mind some of the factors that contribute

to robustness and efficiency. Robustness is affected by the expressiveness of the representations for behavior and physical structure, the granularity of the structures that can be composed, limitations on their composition, and presumptions about the behavioral role played by primitive components. For example, for Ibis the behaviors are quantity shifts or Q1 equations (i.e., non-linear equations and abstractions of them), structures are lumped element networks, granularity is primitive components, and there are no restrictions on how these primitives can be composed.

Performance is affected by granularity and composability, the ability to focus on interesting solutions, and how the search space is abstracted. For example, Ibis focuses on one type of innovation using the interaction topology. It uses the Q1 algebra to view behavior abstractly, and uses the topology of potential interactions to compactly represent salient features of the space of interactions.

Many of these factors have not been greatly explored in earlier design research. For example, most design systems adopt standard engineering representations for behavior (e.g., the digital abstraction, equations on the reals, and piecewise linear functions) and structure (typically components and connections, often with a one to one correspondence between components and function – the behavioral role of the component).

The factors that have received the greatest attention in the design community are granularity, limitations on composability, and presumptions about behavioral role. Robustness improves with finer granularity, less restrictions on composability, and fewer presumptions about behavioral role. These same factors tend to degrade performance. In the remainder of this chapter we organize the different bodies of research in terms of increasing degrees of robustness, based on these three factors.

4.2.1 Design Compilation

The most restrictive systems in terms of robustness, and the most successful in terms of performance, are those falling under the heading of design compilers. These systems are most prevalent in the integrated circuit design community, and include programmed logic array (PLA) generators[27,56], data path generators[42], and silicon compilers[19,44]. These systems receive the name design compilers because of their intellectual roots in program compilation. Each design behavior specification is deterministically mapped to a single solution. For example, a PLA generator takes a set of boolean expressions, reduces them to disjunctive normal form, and then implements them by spatially arranging gates in a fixed pattern. All conjunctions are implemented by arranging gates, representing literals, in an "AND" plane configuration, and disjunctions are implemented by arranging gates, representing conjunctions, into an "OR" plane configuration.

The robustness of design compilers is limited by extreme restrictions on how primitive components can be composed together, the granularity of the structures composed, and the behavioral role they play. For most compilers the composition of components is limited to regular structures – a single repeated structure, organized as an array. In many cases the structures that are being repeated are large composites of primitive components. For example, for data path generators they are digital structures like arithmetic logic units. In other cases the granularity of these structures are closer to the level of primitive components, but their behavioral role is significantly restricted. For example, for a PLA generator the repeated structures are single gates; however, their behavioral role is restricted to achieve a particular implementation of logical AND and OR for disjunctive normal form expressions.

4.2.2 Library Design

Library design approaches [47,29,30,31,24,26,36,39] provide a somewhat greater degree of flexibility. As discussed in the thesis introduction, design knowledge is organized as a set of schemas describing the behavior and structure of "modules" in terms of submodules and their connections. Designs are constructed by a series of hierarchical refinements followed by constraint satisfaction, where each refinement step selects a schema implementing a module and instantiates its submodules and connections.

The added flexibility is first that the systems are not restricted to a single repeated structure, and second that the set of structures are extensible through augmentations to the library. These systems are restricted in three ways: First the robustness of the system is bounded below by the most fine grained modules. These sometimes tend to be large, relative to primitive components (i.e., the smallest modules are aggregates of primitives). Second, the only ways of composing structures, considered by the design approach, are those explicitly specified by the modules. In addition how structures can be composed is fundamentally limited by the hierarchical organization of the database - there is no opportunity, without additional techniques, for function sharing. Finally, the modules presume the purpose the physical structures are supposed to achieve. Thus, even though a physical structure might produce multiple useful functions, it can only be used for those functions specified explicitly in the modules.

4.2.3 Extensions to Basic Library Design

Within the library-based or similar paradigms, several important research contributions have been made towards extending their performance or robustness. REDESIGN [30] is an example of improving performance. Given a design specification,

and a completed design that an engineer decides is close to the specification, REDESIGN determines subportions of the design that can be reused. This is accomplished by resimulating the steps leading to the old design, but evaluating each step based on the new design specifications. Roughly speaking, REDESIGN then keeps subparts of the old design that are still applicable, and constructs new designs for those that aren't. To the extent that the engineer has selected a good match between the design specification and the old design, the performance of the design process can be substantially improved. The simulation of the old plan is fast, because it doesn't involve search. New parts of the design, where search is required, are hopefully quite small. In terms of robustness, REDESIGN has no effect. It can only use a device if it has a plan for simulating its design. Thus any device constructed through redesign could have been constructed directly from the design library alone.

LEAP[29] is an example of extending the generality of the library approach by facilitating the process of acquiring new schemas. Using LEAP, design involves user guided hierarchical refinement, where each refinement uses a library design fragment, or a design fragment proposed by the user. In the latter case, LEAP enters a new schema in the library by generalizing on the schema. This involves a form of goal-directed generalization[28] where LEAP preserves only those constraints on the fragment that are relevant to achieving its stated purpose. The learning technique improves generality by extending the knowledge in the library.

4.2.4 Design Debugging

A number of design approaches have been proposed for improving performance or robustness that use design debugging[52,36,53,43,30,33]. Much of this work is intellectually based on a combination of Sussman's work on "problem solving as debugging almost right plans"(HACKER)[52], and Stallman and Sussman's work on dependency directed backtracking[45]. The basic approach is to construct an initial

solution, identify inconsistencies in the design (bugs), pinpoint commitments that lead to these inconsistencies, and modify the design to remove the inconsistencies.

The various design debugging techniques differ in the way they modify the design in response to a bug. Dependency directed search [45] uses knowledge about the cause of past failures to cut off the pursuit of dead ends early on. When an inconsistency is identified, dependency directed search backs up by removing commitments that lead to the inconsistency, and then continues the search. This does not affect the possible solutions arrived at (robustness), only the speed at which they are obtained.

The systems in references [52,36,30] modify a faulty design using a set of user supplied, situation specific debugging rules. These rules specify a set of conditions on the cause of failure, and propose augmentations to the design when those conditions occur. HACKER [52] explores this approach for reordering steps in a plan. CIROP [36] uses debugging rules, during bipolar amplifier design, to change the selection of schemas instantiated. REDESIGN [30] uses debugging rules during digital design to correct signal mismatch. For REDESIGN this involves introducing a module that transforms a pair of mismatched signals (e.g., a serial output and a parallel) from the one type to the other (e.g., using a serial to parallel converter). None of the debugging rules used in these three systems appear to change a design in a way that couldn't have been arrived at through straight hierarchical refinement. Thus, while the debugging rules may improve performance, they do not affect robustness.

GORDIUS[43] explores the use of debugging on geological interpretation and planning problems. Rather than using a set of situation specific rules, GORDIUS uses a set of general modification procedures, based on principles about time, persistence, causality and the effects of events. By founding the debugging approach on a causal model of how the world works, GORDIUS appears to be able to propose modifications to the "design" (plan or interpretation) that could not be arrived at directly through instantiations of library schemas (called associational rules for this system).

The basic approach taken in GORDIUS shares commonalities with the approach of evolutionary design through focussed innovation used in Ibis. Ibis pinpoints places where innovation is required through a causal model of how physical devices work (i.e., a dynamic interaction topology[59]), drawing upon principles of time, persistence, and causality. The approaches differ in two ways. The first has to due with the model of time used, and its impact on the design system's ability to pinpoint where changes are required. GORDIUS uses a model of time based on global states. We argue in [59] that this model introduces substantial irrelevant temporal orderings and events into the causal description of a device's behavior. These causal descriptions are used to pinpoint the places where modifications are required; thus, the irrelevant details degrade the design system's ability focus on the required modifications. Our approach is based on a causal-temporal representation, called concise histories, that uses a local model of time to eliminate such irrelevant details. The second is the purpose and form of the modifications proposed. GORDIUS' debugging technique modifies event sequences to correct faulty plans, while Ibis' technique improves design performance by altering the interaction topology – what we have argued is a type of innovative change to how a device works.

We consider one remaining approach which is not strictly design debugging, but is similar in that a set of rules are used to modify an initially proposed design. A common limitation of pure library-based approaches is that they do not support function sharing. The hierarchical refinement decomposes a design into a hierarchy of functions and subfunctions, such that each function is implemented by a distinct physical structure – no components are shared between functions.[1] Thus the systems typically do not propose designs in which the same structure is used to achieve multiple, different functions in a design. In [54], Ulrich and Seering propose an approach where a device is designed initially so that distinct functions correspond to distinct struc-

[1]Although for some approaches, if the same desired function appears in multiple submodules, then they may be implemented by a single physical structure.[26]

tures. Next a function sharing procedure is applied that changes the device when possible, so that distinct structures are replaced with one that implements both their functions. This approach can be used to expand upon the sets of solutions proposed by library-based design, and thus improves robustness.

Ibis also exploits function sharing extensively, but through a substantially different approach. To produce a desired interaction, Ibis tries to exploit existing interactions whenever possible. These existing interactions were often introduced in order to achieve some other desired interaction. In this case exploring them to produce additional interactions results in function sharing. Using this approach Ibis is able to propose solutions not possible by first decomposing a design hierarchically, and then collapsing structures through shared functions. Recall that one of the major reasons for introducing the topology of existing interactions is the inaccessibility of variables we need to influence. In this case the only way to produce the desired behavior is to share functions. If the initial solution is not allowed to share functions, then problems with inaccessible variables cannot be solved.

4.2.5 Approaching Design from First Principles

A number of approaches have highlighted the importance of using fine-grained structures to improve robustness. In [2], Barstow highlights the importance of capturing knowledge about programs and their differences as a sequence of incremental refinements. Roylance[39] takes a similar approach for the design of analog ramp generators. The structures in his design library are close, although not strictly isomorphic, to primitive circuit elements, such as resistors and capacitors. Roylance also relaxes restrictions on how these structures are composed. Rather than using a hierarchical decomposition, structures are composed together by backward chaining on an equation specifying the desired circuit behavior. In Roylance's approach two limitations remain with respect to robustness. The first is that, while the structures are fine

grained, they do not include all the primitive elements. Thus, there are structures that his approach cannot construct. Furthermore, the library presumes the purpose of the primitive elements. For example, the use of a capacitor is restricted in terms of which variable it can be used to control. Thus, the system does not entertain functions for structures other than those specified in the library.

Ideally, a robust approach should not impose any additional restrictions on structure beyond those specified in the model of physics. For most approaches the physics specifies that structure be represented as components and connections. Thus a robust approach is one whose granularity is at the level of primitive components, that imposes no restrictions on how components can be composed, and that does not presume the role that components play. By not presuming a component's role, we mean that the desin approach should be able to exploit any property of behavior that is explicit in the component models and connection laws. This is what we call design from first principles, and is exactly what Ibis accomplishes.

Ulrich[53] proposes an approach, called pre-parametric design, that also shares some of the properties we have outlined for design from first principles. The problem Ulrich addresses is the design of single-input, single-output devices constructed by connecting together lumped elements. The goal of the design is to relate two variables. The behavior of the relationship is not specified. In this approach each component is modeled as an n port device (drawn as a bond graph [38]) and the design goal is satisfied if there exists a path through the n port devices between the two variables; that is, there is a sequence of bonds connecting the two variables. The behavior produced by the sequence of bonds (i.e., the relationship between variable values) is not considered. A path is constructed by generating all paths.

This approach has several desirable properties – the granularity is that of primitive components, and no limitations are placed on the role of the components, or how they can be composed. Also to the extent that a sequence of n ports reflects how a device

works, then by focussing on different sequences of bonds that relate two variables, the approach is focussing on innovative differences analogous to those explored by Ibis.

The approaches differ in several important respects: the treatment of function sharing, parasitics, restrictions on the classes of systems designs, focussing the search, and the models of behavior. For example, consider how the two approaches treat behavior. In Ibis the initial behavior specified for a design is a set of quantity shifts. These are then transformed into intensional interactions, described by quantitative equations or their qualitative abstractions usin Q1. When constructing a first cut topology Ibis ignores behavior, focusing on what interacts – the connectivity of the interactions. It then elaborates the solution, taking into consideration behavior.

In contrast in Ulrich's approach making two variables interact is, not the means to the end, but the end itself. As we noted earlier, a more detailed description of the behavior of the interaction is never considered. This suppresses a wide variety of issues that make design from first principles difficult, such as reasoning about continuity, feedback, parasitic behavior, rates of change, the complexity of equations on the reals and time-varying behavior. Many of the contributions of interaction-based invention, such as Q1, Minima, causal dominance, the integration rule, and concise histories, deal with exactly this complexity.

Ignoring behavior, a second difference between the two approaches is the strategy used by Ibis to construct a set of interactions that connect two variables, versus that used in preparametric design for constructing the corresponding set of bonds. In this thesis we have seen that a first cut interaction topology is identified by tracing paths through the potential interaction topology. The potential interaction topology represents a compact abstraction of all possible interactions and all ways they interconnect. This topology takes into consideration the consistency of pairwise connections only.

In pre-parametric design, a different abstraction is used. A first cut solution is

constructed by exhaustively generating all possible compositions of 2 port elements up to a certain number, and testing whether they relate the two specified variables. When constructing this solution, the order of the variables are ignored (e.g., whether it is a first or second derivative). Because of this abstraction the order of the variables in the solution may not agree with those we want to relate. For example, the first cut solution relates q and dp/dt, when we are interested in relating dq/dt and p. In this event, a series of heuristics are used to modify the sequence of n ports so that they relate the variables of the right degree.

Many of the techniques used in these two approaches are complementary. For example, ignoring the degree of a variable provides another type of abstraction that could be incorporated into the topologies of potential and existing interactions. Conversely, in preparametric design the bond graph elements could be organized as topologies of existing and potential bonds, and candidate solutions could be identified by tracing paths.

Both Ibis and Ulrich's approaches operate at the finest level of granularity that can be achieved when working completely within the framework of a lumped element model (i.e., components and connections). To go further we must look into how individual components are designed, based on shape and material properties. Design at this level is an important part of innovation. For example, an innovative aspect of Philon's lamp is the use of a pipe and the surface of the oil to produce a valve. Developing, theories to account for this type of design will require much more sophisticated representations of shape and models of shape's impact on behavior. Only recently have researchers begun to attack some of these difficult issues.[20,33].

Part II

Representations, Principles and Procedural Skills

Chapter 5

Representing Intensional Interactions: The Q1 Algebra

As we saw in Chapter 3 the central focus of interaction-based invention is constructing the topology of intensional interactions. Thus far the representation for this topology has been presented informally through examples. the objective of this chapter is to make this representation more precise.

Intuitively, a topology of interactions is a set of conduits, used to push a device between different regions of its state space during specified time periods (i.e., quantity shifts, such as raising a fluid level, turning off a valve, or saturating a transistor). These conduits accomplish this by combining signals and passing them from one point to another. In some cases an interaction transforms a signal as it is passed, but normally using very simple operations (e.g., negating the signal).

The representation for interactions is central to Ibis' performance and robustness. It defines the search space explored, and determines how easy or difficult it is to analyze features within this space. Given the complexity of the design space resulting

from using first principles, Ibis will become quickly lost without a good representation.

This chapter presents a hybrid qualitative/quantitative algebra, called $Q1$, that provides a good representation for intensional interactions. Our concern here is the expressivity of $Q1$, characterized in terms of its sufficiency for our invention task and its expressiveness relative to other qualitative algebras and representations. Chapter 6 describes the formal properties of $Q1$ with respect the symbolic algebraic manipulations required to building interaction topologies.

The chapter begins by reviewing what it means to be a good representation, and argues that qualitative algebras provide an appropriate starting point for representing intensional interactions (Section 5.1). Next we demonstrate the inadequacy of existing qualitative algebras for our task (Sections 5.2 and 5.3) – the nature of the problem is that they over-abstract. First we show that the most common qualitative algebra – the sign algebra – cannot even describe the punch bowl problem, let alone solve it. Then we discuss the failure of recent attempts to overcome this inadequacy.

The remainder of the chapter presents a novel algebra, $Q1$, that allows us to capture the salient features of interactions, by overcoming the problem of over-abstraction without introducing superfluous details. This is accomplished by the fact that $Q1$ is a hybrid qualitative/quantitative algebra that allows us to select the right level of abstraction to express interactions along a qualitative/quantitative spectrum. Section 5.4 introduces the domain and operators of the algebra, and shows how these operators are used to construct expressions at different levels of abstraction. Section 5.5 discusses the semantics of equations. First it demonstrates that the standard treatment of equations in qualitative algebras makes it difficult to compose interactions – the fundamental operation for interaction-based invention. The nature of this difficulty is that these algebras use an overly abstract definition for equality. We then show that $Q1$ overcomes this problem by strengthening this definition to a full equivalence relation. Finally, to be adequate for inventing quantity shifting devices,

$Q1$ must be able to describe the levels or intervals being shifted between, and the region boundaries of components with multiple operating regions, such as valves and MOS transistors. Section 5.6 demonstrates how this is accomplished using $Q1$.

5.1 Requirements for a Good Representation of Interactions

Consider three properties of a good representation with respect to intensional interactions. A good representation:

1. focuses on the class of objects central to the task,
2. highlights the salient features of those objects, and
3. facilitates the basic operations on those objects.

Consider the first property. One class of objects central to a task are those used to evaluate the appropriateness of a solution. For invention these are objects that make a device innovative. The thesis introduction (Section 1.1) argued that a property that makes a device innovative is that it works in a way fundamentally different from those previously developed. How a device works is captured by the topology of interactions it produces, and fundamental differences correspond to structural differences in these topologies. Thus a topology of interactions satisfies the first property of a good representation.

The first property primarily serves as a justification for developing a representation for interactions. The justification for the representation being intensional is provided in a moment. The second two properties affect what the representation of these interactions must capture.

Consider the second property – highlighting salient features. For invention these are features of interactions that affect how a device behaves. Specifically, those that affect features of device behavior that determine whether or not it meets the design specification. For our task these are the required quantity shifts specified for the device. Quantity shifts are accomplished by pushing quantities in particular directions during specified time periods. This is achieved by constraining the sign of their derivatives (the first stage of interaction-based invention, discussed in Section 2.4). Thus the salient features of interactions are those that constrain the signs of these derivatives directly, or indirectly through other quantities and interactions. The next few sections develop a qualitative algebra, $Q1$, that captures these salient features.

Finally, consider the third property – facilitating basic operations. Chapter 3 developed the strategy for constructing intensional interaction topologies from first principles. Two basic operations used during this construction are composing and comparing interactions. Interactions are composed by identifying a shared variable and using substitution of equals for equals. Thus the representation for interactions should make performing this substitution as simple as possible. Section 5.5 discusses properties of qualitative equations, used to describe intensional interactions, that are necessary to satisfy this property.

The third property (facilitating operations) also provides a justification for why the representation of interactions is intensional. First, in some cases intensional descriptions are more compact and easier to manipulate than their equivalent extensional descriptions. More important, in some cases the extensional descriptions are so large that they can't even be constructed. In Section 2.2, when first describing the qualitative/quantitative dimension of interactions, we argued that in some situations a quantitative description of an interaction is required, in order to determine its impact on how a device works. These interactions can be described intensionally as algebraic equations; however, they cannot be described extensionally since the

extension of a quantitative relation is not enumerable.

To satisfy the last two properties of a good representation we begin by exploring the applicability of qualitative algebras; specifically algebras on signs. Consider why this is reasonable. To describe interactions at a quantitative level, a scientist or engineer uses a set of algebraic equations on the reals. A basic tenant of qualitative physics – the study of qualitative reasoning about continuous physical systems – is that, when reasoning at a qualitative level, a scientist or engineer is manipulating an abstraction of equations on the reals. That is, an algebra is used whose domain is an abstraction of the reals; for example, the sign of a quantity. A primary focus of qualitative physics is to identify appropriate "*qualitative algebras*" that capture the abstractions scientists and engineers use.

Thus far the qualitative reasoning community has focussed primarily on describing the behavior of devices in terms of quantity shifts (i.e., in terms of movements between different regions of a device's state space[63]). The most common qualitative algebra used for this purpose is an algebra on the signs of quantities. A sign algebra seems like an appropriate starting point for our purposes as well, since as we argued above, quantity shifts are accomplished by constraining the sign of the derivative of the quantity being shifted.

5.2 A Qualitative Algebra on Signs

The most common abstraction used in qualitative reasoning is the sign of a quantity, $S = \{ +, 0, - \}$, used to describe the direction that quantities move in a continuous physical system. For example, the phrase "the ball is rising" is captured by the equation:

$$[V_{ball}] = [+]$$

where square brackets ("[ ]") around a quantity denotes the sign of the quantity,[1] and square brackets around elements of $\mathcal{S}$ are used to distinguish them from addition and subtraction on the reals (e.g., + denotes real addition, while [+] denotes a positive value).

One example of an algebra on signs is *confluences* [12], which consists of two operators – sign addition (denoted $\oplus$) and sign subtraction (denoted $\ominus$). Intuitively the expression $[x] \oplus [y]$ answers the question: "what is the sign of $x + y$ given only the signs of x and y?" While the expression $[x] \ominus [y]$ answers the analogous question: "what is the sign of $x - y$ given only the signs of x and y?"

$\oplus$	$-$	0	$+$	?
$-$	$-$	$-$	?	?
0	$-$	0	$+$	?
$+$	?	$+$	$+$	?
?	?	?	?	?

$\ominus$	$-$	0	$+$	?
$-$	?	$-$	$-$	?
0	$-$	0	$+$	?
$+$	$+$	$+$	?	?
?	?	?	?	?

Figure 5.1: Definitions for sign addition ($\oplus$) and subtraction ($\ominus$) used in confluences.

These operators are summarized by the tables shown in Figure 5.1. Consider $\oplus$: if x and y are positive, then their sum is also positive; on the other hand, if x and y have opposite signs then the result is ambiguous – the sum could be any real value: positive, negative or zero. This situation is denoted by a fourth qualitative value "?".[2] The set of all qualitative values is denoted $\mathcal{S}' = \{+, 0, -, ?\}$. As we will see in a moment, this ambiguity is the major source of weakness in a sign algebra.

[1] More formally, let $\Re$ denote the reals, then we define the mapping [ ]: $\Re \rightarrow \mathcal{S}$ as:

For any $x \in \Re$, $[x] = \begin{cases} + & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ - & \text{if } x < 0 \end{cases}$

Thus the operator [ ] partitions $\Re$ into three intervals, $(0, +\infty)$, $[0, 0]$ and $(-\infty, 0)$ corresponding to $+, 0$ and $-$.

[2] That is, ? is used to represent an undetermined sign (i.e., the value may lie in any one of the three intervals). Thus ? corresponds to the interval $(-\infty, +\infty)$.

5.3 Inadequacy of Existing Qualitative Algebras

Current qualitative algebras, such as confluences, are not sufficiently expressive to describe the design tasks we are considering. To see this consider the punch bowl problem. Recall from Section 2.1, that the problem is to invent a device that automatically restores the level of punch in a bowl from a vat (Figure 2.2) and that Ibis decides to accomplish this by making the direction of bowl height change have the same sign as the vat/bowl height difference (Section 2.3). We can capture this desired behavior by the following equation:

$$[dH_b/dt] = [H_v - H_b]$$

where H_b and H_v denote the height of punch in the bowl and vat, respectively.

Unfortunately we can not express the above equation using a sign algebra – the equation involves real subtraction ($[H_v - H_b]$), but we have only sign addition ($\oplus$) and subtraction ($\ominus$). Instead the best we can do is the following equation, which replaces real subtraction with sign subtraction:

$$[dH_b/dt] = [H_v] \ominus [H_b]$$

However, this does not express at all what we want. For example, most of the time both the height of the vat and height of the bowl will be positive. However, the difference of two positives is unknown (i.e., $[H_v] \ominus [H_b] \Rightarrow [+] \ominus [+] = [?]$); thus, most of the time the equation provides no information. This means that, using a sign algebra, we can't even express the punch bowl problem, let alone solve it. The basic difficulty is that the use of a sign algebra forces us to over abstract – it forced us to replace $[H_v - H_b]$ with $[H_v] \ominus [H_b]$.

Another indication of the weakness of the sign algebra is that it lacks one of the

most basic properties of algebraic systems – an additive inverse.[3] One consequence is there is no cancellation rule for $\oplus$, thus we can't solve systems of sign equations using any currently known techniques.

A number of attempts have been made to strengthen sign algebras, by introducing additional landmarks into a qualitative representation. A landmark is a value that breaks the real number line into regions of interest. For example, the sign algebra uses 0 as a landmark, breaking the number line into [+],[−] and [0]. Forbus[17] introduced the notion of a quantity space that allows a qualitative representation to be described in terms of any number of landmarks, or equivalently in terms of any number of subintervals, while Williams[63] introduced a qualitative representation that breaks the space of values a set of quantities can take on (i.e., *state space*) into a set of open regions separated by boundaries. Unfortunately, a recent analysis by Struss[51] shows that the introduction of a finite number of landmarks cannot overcome certain fundamental weaknesses, such as the ambiguity of addition and the lack of an additive inverse.

Consider, for example, the inherent ambiguity of addition. Intuitively a qualitative algebra can be viewed as an interval algebra. Introducing a finite number of landmarks breaks the reals into a finite number of intervals. Struss shows that in this case the problem is that, given only the intervals that two quantities lie within, we cannot always determine a unique interval that the sum lies within. The problem is analogous for subtraction; thus, a unique additive inverse does not exist.

A third indication of this weakness was supplied in Section 2.2, during our introduction of the qualitative/quantitative dimension of interactions. We demonstrated that, even though a desired interaction is expressed as a relationship on signs, showing that a topology of interactions produces the desired interaction typically requires

[3]More specifically, there is no additive inverse in the sign algebra for any qualitative value except [0] (i.e., if $v \in \{+,-,?\}$, there is no $w \in \mathcal{S}'$ such that $v \oplus w = [0]$).

some of the interactions to be described using equations on the reals.

5.4 Q1 – A Hybrid Algebra

To overcome these problems we construct a hybrid algebra called $Q1$ by merging the real and sign algebras. That is, expressions in $Q1$ are composed of the standard real operators ($+, -, \times, /$ operating on $\Re$), together with the analogous sign operators ($\oplus, \ominus, \otimes, \oslash$ operating on $\mathcal{S}'$), and [ ], which maps expressions from $\Re$ to $\mathcal{S}'$. Definitions for $\oplus$ and $\ominus$ appeared in Figure 5.1. The definitions for the additional sign operators, $\otimes$ and $\oslash$, are shown in Figure 5.2.

$\otimes$	$-$	0	$+$	?
$-$	$+$	0	$-$	?
0	0	0	0	0
$+$	$-$	0	$+$	?
?	?	0	?	?

$\oslash$	$-$	0	$+$	?
$-$	$+$	U	$-$	U
0	0	U	0	U
$+$	$-$	U	$+$	U
?	?	U	?	U

Figure 5.2: Definitions for sign multiplication ($\otimes$) and division ($\oslash$) used in $Q1$. U denotes values for which an operator is undefined; that is, division by an interval containing zero (i.e., [0] and [?]).

Because $Q1$ includes the standard real operators as well as sign operators, it allows us to express equations like:

$$[dH_b/dt] = [H_v - H_b]$$

which accurately describes the desired behavior for the punch bowl problem.

$Q1$ overcomes the problems of qualitative algebras with finite landmarks by using the infinite resolution of the reals for selected sub-expressions of equations. Recall

that the second property of a good representation is it must capture the salient features of interactions. Traditional qualitative algebras do not accomplish this because they over-abstract. For example as long as a finite number of landmarks is used addition will be ambiguous. However, as we move to an algebra on the reals this is no longer a problem. Because the reals have infinite resolution, this ambiguity is removed. Thus the first key is that, in some cases, using the full resolution of the reals is absolutely crucial.

But we cannot use the reals indiscriminately; otherwise, under-abstraction becomes a problem. Thus the second key is to use the most abstract, qualitative description that is sufficient. The resolution of the reals should only be used in those cases where it is necessary in order to avoid an ambiguity that matters with respect to how a device works. To accomplish this a subexpression of an equation is expressed using real operators only if it would otherwise introduce an unacceptable ambiguity. By constructing hybrid qualitative/quantitative equations we are able to represent only the salient properties of interactions, thus satisfying the second criteria for a good representation (Section 5.1).

We can gain further intuition into $Q1$ through a spatial metaphor. At a quantitative level the state of a device is a single point within state space – an infinite space of points defined by the independent variables of the device. A set of qualitative equations breaks the state space into a finite set of regions. At this qualitative level the state of the device is the region it lies within (i.e., the qualitative values of its state variables). By introducing a hybrid qualitative/quantitative algebra we allow sub-portions of a device's state space to be described with infinite resolution (i.e., points) while other areas are described qualitatively as regions.

Earlier we stated that $Q1$ permits statements to be made at different levels of abstraction along a qualitative/quantitative spectrum; consider when this is the case. Sign operators are more abstract than real operators. Thus if we take a real expres-

sion and start replacing its operators with sign operators, the expression becomes successively more abstract (Figure 5.3).

$$[w - x + y - z]$$
$$[w] \ominus [x + y - z]$$
$$[w] \ominus [x] \oplus [y - z]$$
$$[w] \ominus [x] \oplus [y] \ominus [z]$$

Figure 5.3: Expressions in $Q1$ become successively more abstract as real operators are replaced with sign operators.

Another way to view this is that expressions with more real operators decrease the opportunity for ambiguity to arise by delaying the point at which they move $\Re$ to $\mathcal{S}'$ (Figure 5.4). For example, in Figure 5.3 suppose that w is positive and x is negative. Then for the bottom expression, an ambiguity can arise if either y is negative or z is positive. On the other hand, the second to bottom expression only introduces an ambiguity if, in addition, the difference of y and z is negative. Furthermore, for the top expression there is no opportunity for an ambiguity to arise. By making a judicious selection of sign and real operators we can select the exact level of abstraction appropriate for a task.

5.5 Semantics of Q1 Equations

Equations in $Q1$ have a very different semantics from those in other qualitative algebras. This arises from the fact that most qualitative algebras except $Q1$ use a non-standard definition for "=". The result for these other algebras is that substitution of equals for equals no longer applies (*i.e.*, $a = b, b = c \not\Rightarrow a = c$). Recall that substitution is central to composing interactions; thus, these algebras do not

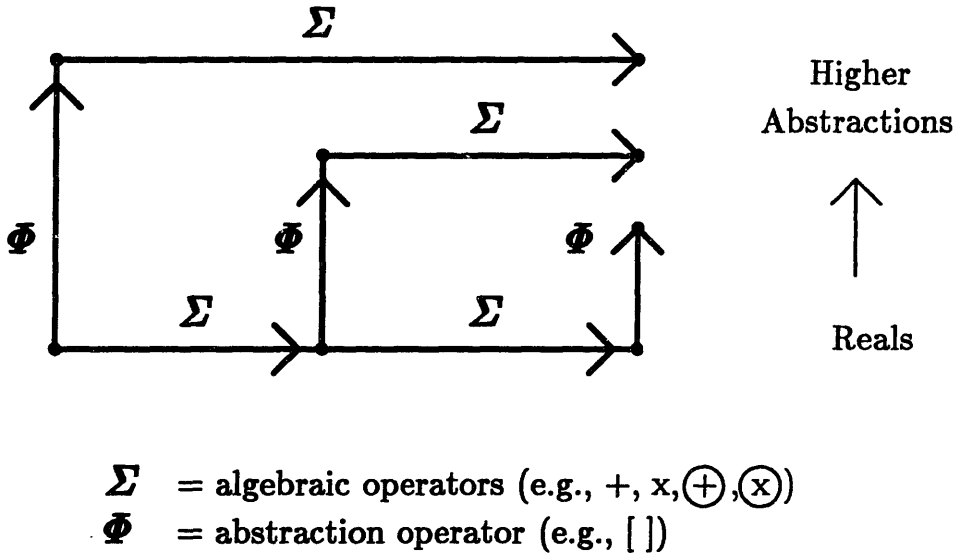


Figure 5.4: Replacing an expression's real operators with sign operators increases the opportunity for ambiguity to arise by moving from $\Re$ to $\mathcal{S}'$ at an earlier point.

facilitate a basic operation on interactions – they fail to meet the third criteria of a good representation (Section 5.1). $Q1$ uses the standard definition of "=" and thus does no have this problem.

For most qualitative algebras[15,51], such as confluences [12], two sign expressions $S1$ and $S2$, are said to be equal if $S1$ and $S2$ denote the same sign *or if either* $S1$ *or* $S2$ *denote* [?]. For example, the equation

$$[x] \oplus [y] = [0]$$

is satisfied if $[x]$ is $[+]$ and $[y]$ is $[-]$. This might seem intuitive; after all, a positive and negative do sum to 0.

However, the nonstandard treatment of equality in confluences has the serious consequence that equality is no longer transitive. For example, the following equations

are true for confluences:

$$[+] = [?]$$
$$[?] = [-]$$

however, it's not the case that:

$$[+] = [-]$$

If we treat the values of $\mathcal{S}'$ as intervals on the reals then the semantics of "=" in confluences is a test for whether two intervals overlap. This relation is not transitive since two intervals a and c overlapping a common interval b does not suggest that a and c overlap.

This has the serious consequence that *in confluences we can't substitute equals for equals* – a fundamental operation for combining equations. This is catastrophic for design and verification tasks, since showing that a device produces a desired interaction involves combining equations. Thus these algebras fail to meet the third criteria of a good representation – facilitating basic operations on the objects of the representation.

The problem again is one of over-abstraction. Taking "=" to mean that two intervals overlap is much weaker than the standard meaning that the two intervals are the same. Furthermore, it is much weaker than necessary. Our experience is that for all qualitative equations used in our invention examples the stronger meaning of equality applies.

$Q1$ overcomes this problem by treating equality as a true equivalence relation. $Q1$ permits two types of equations – equivalences between expressions denoting real values, (e.g., $a + b = c$), and equivalences between expressions denoting signs (e.g.,

$[a + b] = [c]$). In $Q1$ "=" takes on the standard meaning – equal expressions denote the same value. For example, the equation

$$[x] \oplus [y] = [0]$$

is satisfied only if x and y are 0.

5.6 Using Q1 to Define Traditional Qualitative Representations

To be adequate for inventing quantity shifting devices, $Q1$ must be able to describe the values or intervals being shifted between, and the region boundaries of components with multiple operating regions, such as valves and MOS transistors. Traditional qualitative representations are used to produce this type of description. This section demonstrates how this is accomplished using $Q1$. Furthermore, through this demonstration we show that $Q1$ is strictly more expressive than qualitative algebras based on traditional representations. That is, this demonstration shows that $Q1$ subsumes these traditional representations, and we have already seen that $Q1$ can express relations not possible in these other representations. Note that this material is presented here for completeness, but is not critical to understanding the remainder of this thesis.

In [63] we defined a qualitative representation as one that breaks the state space of a device into a set of regions separated by boundaries. A qualitative description of a device's dynamic behavior is then a sequence of regions the device moves through over time. This representation is a generalization of qualitative representations used in other formalisms, such as the sign of quantity changes [12], partial orders on quantities [17,21], and interval decompositions [51]. Thus, it subsumes the class of

behaviors analyzed by current qualitative simulation approaches.[4]

A qualitative representation based on state-space regions is useful for several reasons. It is appropriate for describing the large signal behavior of many different types of devices within a variety of domains (e.g., water clocks, pulley and lever systems, and electrical memory circuits). In addition, the representation is crucial for modeling a component whose interactions change depending on the region of state space it is operating in. For example, the interactions produced by a fluid vessel change depending on whether it is empty, partially full, or full; analogously the interactions for an MOS transistor depend on whether it is operating in the cutoff, unsaturated or saturated regions. Finally, a representation based on state space regions captures the relevant features of behavior for quantity-shifting devices – the devices we have focussed on in this dissertation.

Next consider how a state space decomposition into regions is accomplished using $Q1$. The approach taken here is similar to one we describe in [63,64] – each region is constructed from a set of *boundary equations* that describe the region's boundaries. The difference between the approach here and that of our earlier work [63,64] lies in the expressive power of the algebra used to construct the boundary equations. In [63] a sign algebra was used. However, a sign algebra is insufficient for the task – it over-abstracts in a manner similar to the problem discussed in Section 5.3. As a consequence, the regions actually described are often much larger than those intended. For example, suppose each of a region's boundaries is a line. Each line can be described by equations of the form $Y = a \times X + b$. However, this equation requires real operators, while only sign operators are available. The closest equation expressible using a sign algebra is $[Y] = [a] \otimes [X] \oplus [b]$. But this describes a region, not a line (except when $a = b = 0$). Typically, this region encompasses a much greater set of points than the line. For example suppose $a = 1$ and $b = 0$, then

[4]One category of qualitative algebras not subsumed by $Q1$ are those that capture order of magnitude reasoning. These algebras are based abstractions of the hyper-reals, rather than the reals[35,58]

the line produces a diagonal through the upper right and lower left quadrants of the X-Y plane (left side of Figure 5.5). On the other hand, the sign equation defines a region containing all of the upper right and lower left quadrants (right side of Figure 5.5) – *half the plane.* This example is not at all contrived; the case is at least this bad for any values of a and b other than $a = b = 0$.

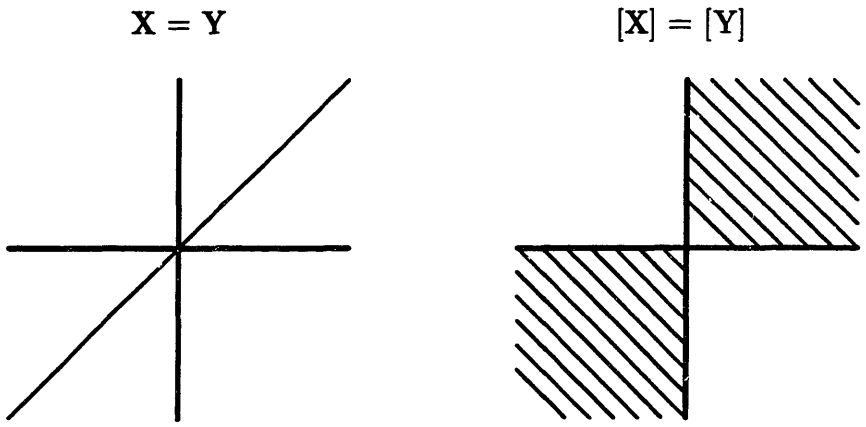


Figure 5.5: Demonstration that clean boundaries are lost through over-abstraction when using only a sign algebra.

Using $Q1$ to construct the boundary equations overcomes this problem. $Q1$ allows real as well as sign operators, thus each line can be described precisely. Consider how a qualitative representation based on state space regions is defined using $Q1$. The example used here is a mosfet (for Metal Oxide Silicon Field Effect Transistor), a component used to construct most modern day digital systems.

The mosfet is a device that has multiple *operating regions.* That is, the interactions produced by the device (the equations) change depending on what region of state-space the device is operating in. More specifically the state-space of a mosfet is broken into three regions named saturated, unsaturated and cutoff. These regions

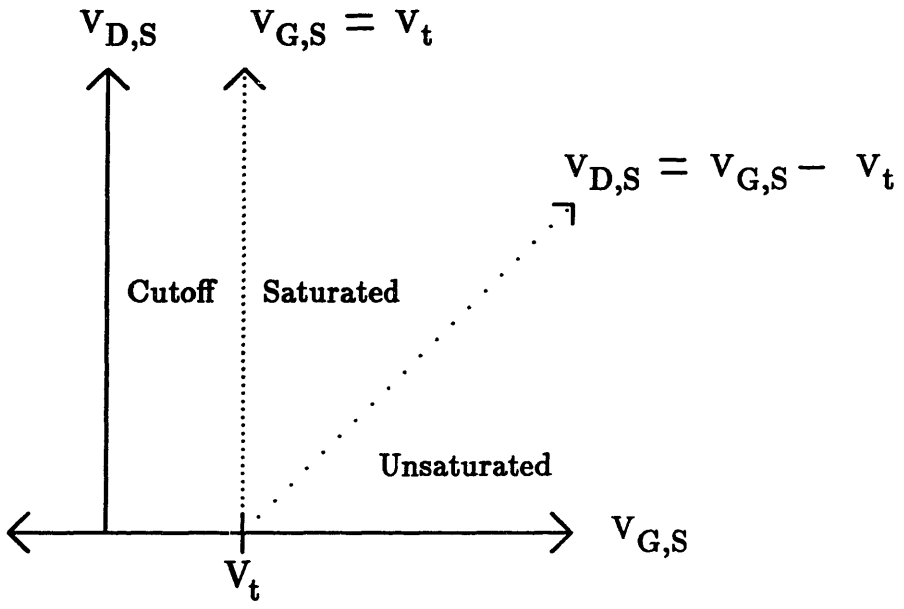


Figure 5.6: A mosfet's state space, defined by $V_{G,S}$ and $V_{D,S}$, is partitioned into three distinct regions of operation in which different interactions are produced.

are defined as shown in Figure 5.6 by three boundaries:

$$\begin{aligned} V_{G,S} &= V_t \\ V_{D,S} &= V_{G,S} - V_t \\ V_{G,S} &= 0 \end{aligned}$$

If a mosfet is in the saturated region it produces the interaction:

$$[I_{D,S}] = [V_{G,S} - V_t]^2$$

If the mosfet moves to unsaturated then the interaction changes to:

$$[I_{D,S}] = [2 \times (V_{G,S} - V_t) - V_{D,S}] \otimes [V_{D,S}]$$

Finally, if the mosfet is in cutoff then the interaction becomes:

$$[I_{D,S}] = [0]$$

To develop a model for the mosfet we first define its operating regions as a qualitative representation using $Q1$. The state space of a device D is defined by a set of independent variables $\{v_1, v_2 \ldots\}$. The state space of a mosfet is defined by two voltages: $V_{G,S}$ and $V_{D,S}$. A qualitative representation, $QR(v_1, v_2, \ldots)$, for a device is a decomposition of its state space into a finite set of regions of interest $\{R_1, R_2, \ldots\}$. For the mosfet the qualitative representation divides $V_{G,S}$ and $V_{D,S}$ into the three disjoint operating regions of Figure 5.6 – MOSFET$(V_{G,S}, V_{D,S})$ = {SATURATED, UNSATURATED, CUTOFF}. In general the decomposition may partition the state space completely or partially (e.g., the partition may be based on only one or a few of a device's state variables), and the regions may or may not be disjoint. For the mosfet these regions are disjoint.

The qualitative value of a device D is defined as the region it currently lies within. More precisely, the qualitative value of D with respect to QR is the subset of QR's regions $\{R_i, R_j, \ldots\}$ that D's independent variables are currently in. This is denoted $[D(v_1, v_2, \ldots)]_{QR} = \{R_i, R_j, \ldots\}$, or $[D(v_1, v_2, \ldots)]_{QR} = R_i$ when QR is a disjoint decomposition. For example, a mosfet $M1$ being saturated is denoted:

$$[M1(V_{G,S}, Vds)]_{mosfet} = \text{SATURATED}$$

In a moment we will show how to construct a set of expressions that determine the qualitative value of a device from its state variables and the boundary equations.

A device will often have several decompositions, each highlighting different properties of its behavior. The mosfet model demonstrates one type of decomposition – one that highlights qualitative changes to how a device's behavior is produced, such

as changes to the device's interaction topology. For the mosfet the topology of its interactions change as it moves between any of the three disjoint regions – SATURATED, UNSATURATED AND CUTOFF (Figure 5.6). That is, when in SATURATED, a mosfet's current $I_{D,S}$ interacts with $V_{G,S}$. When in UNSATURATED, $I_{D,S}$ interacts with both $V_{G,S}$ and $V_{D,S}$. And when in CUTOFF, $I_{D,S}$ is fixed at zero, independent of $V_{G,S}$ and $V_{D,S}$.

If a device is used as a component of other devices, then it may also have one or more additional decompositions. These highlight changes in the component's state space that result in changes to how its containing device produces its desired behavior. For example, if a mosfet is used to implement logic gates in a digital circuit, then each of its voltages will be broken into three regions: "1", "0" and "forbidden zone" (e.g., we define logic(V) = $\{1, 0, \text{FORBIDDEN}\}$).

Next we consider how a qualitative representations is defined, using $Q1$. To determine if a device is in some region we determine the device's position with respect to each of the region's boundaries – if the device is within each of the region's boundaries then the device is within that region. To accomplish this we place each boundary equation in the form "0 = *exp*", where *exp*, called a *boundary expression*, is composed of $Q1$ operators. The position of a device relative to a boundary is determined by the sign of its boundary expression; thus, the requisite position of a device relative to a boundary is specified by restricting the boundary expression's sign. For example, suppose some boundary consists of a line running horizontally along a Cartesian coordinate system (i.e., $0 = b - Y$). Then the device is above the boundary when $[b - Y] = [+]$, on the boundary when $[b - Y] = [0]$, and below when $[b - Y] = [-]$. Finally, the conditions for a device being in a particular region are specified as a propositional formula composing together the conditions on the sign of each boundary expression.

For example, the three regions for the mosfet (Figure 5.6) are separated by three

boundaries – all three regions are bounded below by $V_{G,S} = 0$; the boundary separating CUTOFF and SATURATED is described by $V_{G,S} - V_t = 0$, where V_t is a constant, and the boundary separating SATURATED from UNSATURATED is described by $V_{D,S} - (V_{G,S} - V_t) = 0$. The following propositions define the qualitative representation for a mosfet $M1$:

$[M1(V_{G,S}, V_{D,S})]$ = CUTOFF

if $[V_{D,S}] = [+]$ and $[V_{G,S} - V_t] = ([-]\text{or } [0])$

$[M1(V_{G,S}, V_{D,S})]$ = UNSATURATED

if $[V_{D,S}] = [+]$, $[V_{G,S} - V_t] = [+]$, and $[V_{D,S} - (V_{G,S} - V_t)] = [-]$

$[M1(V_{G,S}, V_{D,S})]$ = SATURATED

if $[V_{D,S}] = [+]$, $[V_{G,S} - V_t] = [+]$, and $[V_{D,S} - (V_{G,S} - V_t)] = ([+]\text{or } [0])$

Given this representation, the model for a mosfet, M1, is defined by three interactions:

$[M1(V_{G,S}, V_{D,S})]$ = CUTOFF

implies $[I_{D,S}] = [0]$

$[M1(V_{G,S}, V_{D,S})]$ = UNSATURATED

implies $[I_{D,S}] = [2 \times (V_{G,S} - V_t) - V_{D,S}] \otimes [V_{D,S}]$

$[M1(V_{G,S}, V_{D,S})]$ = SATURATED

implies $[I_{D,S}] = [V_{G,S} - V_t]^2$

5.7 Summary

A topology of interactions can be viewed as a set of conduits, used to push a device between different regions of its state space during specified time periods. This chapter presented a representation for describing the interaction topology intensionally – a hybrid qualitative/quantitative algebra called $Q1$. Furthermore, we demonstrated that, unlike traditional qualitative representations, $Q1$ satisfies all three basic properties of a good representation: it represents interactions, which are central to identifying innovations, it highlights their salient features, and facilitates basic operations on them (i.e., substitution of equals).

We began by analyzing the utility of existing qualitative algebras that break a device's state space into a finite set of qualitative regions. These algebras proved inadequate – they over-abstract. Next we proposed a hybrid qualitative/quantitative algebra, $Q1$, that allows statements to be made at many points along a qualitative/quantitative spectrum, thus solving the problem of over-abstraction, while capturing only salient features of interactions. The power of $Q1$ comes from a merging of a complete sign algebra and the algebra on reals, together with a definition of "equals" that facilitates reasoning about the composition of interactions. Finally we demonstrated how $Q1$ is used to describe levels or intervals that a quantity shifts between and boundaries for devices with multiple operating regions. Both are crucial for reasoning about the class of devices we are interested in inventing. In the next chapter, analyze some of $Q1$'s more interesting properties, and then use these properties to construct a set of symbolic manipulation procedures that embody the qualitative algebraic skills required for interaction-based invention.

Chapter 6

Minima: Qualitative Algebraic Skills

In Section 1.9 of the thesis introduction we argued that it is important, whenever possible, to raise the granularity of first principles, by embodying them into a set of procedural skills that perform operations basic to interaction-based invention. This removes the need to reason from the first principles explicitly, thus eliminating a substantial computational hurdle to Ibis' performance. This chapter develops Ibis' qualitative algebraic manipulation skills. These are used when checking the behavior of a proposed interaction topology (Section 3.5.5), to compose and compare interactions. The overriding concern during this development is that the algebraic skills are computationally efficient and robust – they capture the first principles.

The chapter begins by discussing a controversy within the qualitative reasoning community about the plausibility of qualitative algebraic manipulation, and the highlights of our approach which answers this controversy (Section 6.1). Next an axiomatization of Q1 is provided (Section 6.2). Third, a set of properties are derived from these axioms that are central to composing and comparing interactions (Sec-

tion 6.3), more precisely, properties for canonicalizing and solving equations. Fourth, these properties are embodied in a set of symbolic algebra procedures, called *Minima* (Section 6.4). Finally, Minima is demonstrated on a interaction composition example, the basic inference for testing the behavior of interaction topologies (Section 6.5).

6.1 A Controversy and Highlights of the Solution

The development of Q1 and Minima has important ramifications, not only for our theory of invention, but the qualitative reasoning community in general. Very little has been written about the properties of qualitative algebras, i.e., algebras defined on abstractions of the reals. The few existing studies have focused on the properties of "confluences" – equations involving only addition and subtraction on the signs of quantities[12]. It has been known for some time [11] that confluences are weak – e.g., there is no additive inverse. Recently Struss [51] has performed a detailed analysis of confluences: in replacing signs with a variety of interval representations he came to some disheartening conclusions:

> "Since the uncovered drawbacks [of confluences] turn out to be very severe, this should motivate a search for additional concepts and approaches of a completely different nature."

As a consequence of this weakness few qualitative reasoning systems manipulate qualitative equations symbolically.[1] Determining whether this conclusion is correct is central to the success of our theory of invention, since qualitative algebraic manipulation is at the heart of constructing an interaction topology that produces the desired behavior.

[1]The only current exceptions are the qualitative gauss rule [15] and the composition of the M operator[21].

Our treatment of the qualitative algebraic skills represents a significant advance in the qualitative reasoning community. In contrast to the above conclusion about the inherent weakness of qualitative algebras, through our hybrid algebra Q1, we show that qualitative algebras can be made extremely powerful. First, an axiomatization of the Q1 algebra and an analysis of their consequences highlights a number of powerful properties for symbolic manipulation (Sections 6.2 and 6.3), many that are shared with algebras on the reals and some unique to qualitative algebras. Second, by exploiting these properties we are able to develop a powerful set of symbolic algebra procedures (called *Minima*) for combining, simplifying, canonicalizing and factoring qualitative equations (Section 6.4). The unique properties of the qualitative component of Q1 allow efficient algebraic manipulation of qualitative expressions, without resorting to the expensive procedures needed to manipulate real expressions. Finally, the power of the qualitative algebra is demonstrated by its adequacy for capturing Ibis' procedural skills for algebraic manipulation of interactions. It is our belief that these algebraic skills play a central role in other scientific and engineering tasks as well, where interactions are the central focus.

The power of Minima is derived from three distinctive features of Q1. First, Q1 is a hybrid qualitative-quantitative algebra, that merges a sign algebra and the traditional algebra on the reals. Using a combination of real and sign operators allows an inventor to select the appropriate abstraction level to describe a statement along a qualitative-quantitative spectrum, thus avoiding problems of over and under abstraction. Second, Q1 incorporates a complete sign algebra, including multiplication and division. This allows any of the real operators to be selectively abstracted. In addition, much of Minima's power is derived from exploiting unique properties of the complete sign algebra previously overlooked by other researchers. Third, unlike other qualitative algebras[12,63,51,15,21], Q1 treats "=" as a true equivalence relation. This makes it possible to substitute equals for equals, the basis for how Minima combines interactions.

6.2 First Principles of Q1

This section lists a compact set of axioms that constitute the first principles of Q1. Q1 is defined as the structure $\langle \Re \cup \mathcal{S}', +, \times, \oplus, \otimes, [\,] \rangle$, where $-$, $/$ are defined in terms of $+,\times$ and $\ominus$, $\oslash$ are defined in terms of $\otimes$. The axioms of Q1 are broken into those characterizing the real and sign algebras ($\langle \Re, +, \times \rangle$ $\langle \mathcal{S}', \oplus, \otimes \rangle$), the interrelationship between them, and the definitions of the remaining operators. In the remainder of this discussion s,t and u denote elements of $\mathcal{S}'$ and a,b,c denote elements of $\Re$. The following table summarizes the field axioms of $\langle \Re, +, \times \rangle$ and the corresponding axioms of $\langle \mathcal{S}', \oplus, \otimes \rangle$:

Axioms	$\langle \mathcal{S}', \oplus, \otimes \rangle$	$\langle \Re, +, \times \rangle$
Identity :	$s \oplus [0] = s$	$a + 0 = a$
	$s \otimes [+] = s$	$a \times 1 = a$
Inverse :	**none**	$a + (-a) = 0$
	$s \otimes (s^{-1}) = [+]$ if $s \neq [0] or [?]$	$a \times (a^{-1}) = 1$ if $a \neq 0$
Commutativity :	$s \oplus t = t \oplus s$	$a + b = b + a$
	$s \otimes t = t \otimes s$	$a \times b = b \times a$
Associative :	$(s \oplus t) \oplus u = s \oplus (t \oplus u)$	$(a + b) + c = a + (b + c)$
	$(s \otimes t) \otimes u = s \otimes (t \otimes u)$	$(a \times b) \times c = a \times (b \times c)$
Distributivity :	$s \otimes (t \oplus u) = s \otimes t \oplus s \otimes u$	$a \times (b + c) = a \times b + a \times c$

where unary $-$ denotes the additive inverse of $+$; a^{-1} and s^{-1} denote the multiplicative inverses of $\times$ and $\otimes$, respectively.

$\langle \mathcal{S}', \oplus, \otimes \rangle$ satisfies two additional axioms that do not hold for $\langle \Re, +, \times \rangle$:

$$s \otimes s = [+] \text{ if } s \neq [0] or [?]$$
$$s \oplus s = s$$

The relationship between $\langle \mathcal{S}', \oplus, \otimes \rangle$ and $\langle \Re, +, \times \rangle$, are summarized through the relationships between each of the corresponding real and sign operators:

$$\begin{aligned} [a] \otimes [b] &= [a \times b] \\ [a] \oplus [b] &\text{ contains } [a + b] \end{aligned}$$

where "x contains y" if y is a subinterval of x.

Finally, $-$, $/$, $\oplus$ and $\oslash$ are defined in terms of the operators of $\langle \mathcal{S}', \oplus, \otimes \rangle$ and their inverses, inheriting properties through these definitions. Binary $-$, $/$ and $\oslash$ are defined in terms of the additive and multiplicative inverses in the standard way:

$$\begin{aligned} a - b &\equiv a + (-b) \\ a/b &\equiv a \times (b)^{-}1 \\ s \oslash t &\equiv s \otimes (t)^{-}1 \end{aligned}$$

Unary and binary $\ominus$ are defined in terms of $\otimes$:[2]

$$\begin{aligned} \ominus s &\equiv [-] \otimes s \\ s \ominus t &\equiv s \oplus (\ominus t) \end{aligned}$$

The axiomatization of $\langle \mathcal{S}', \oplus, \otimes \rangle$ is sound and complete. The proofs are straightforward from the semantics of the operators, and are not included. The axioms for $\langle \Re, +, \times \rangle$ are sound, although necessarily incomplete (by Godel's incompleteness theorem [10]). Next consider the consequences of these axioms that are relevant to manipulating interactions.

[2] $\ominus$ cannot be defined in terms of the additive inverse of $\oplus$ since one doesn't exist.

6.3 Consequences for Manipulating Interactions

Recall that the basic operations for manipulating interactions involve composition and comparison. These are used by Ibis to verify that a coarse solution produces the desired behavior (Section 3.5.5). Comparison is facilitated by reducing an equation to a canonical form. Composition requires the ability to solve for a particular variable in a single equation. This section focuses on properties of Q1 most important for canonicalization and equation solving.

This analysis is structured in a manner similar to the first principles in the preceding section. We first analyze the relevant properties of $\langle \mathcal{S}', \oplus, \otimes \rangle$ and $\langle \Re, +, \times \rangle$ individually, and then the relationship between them. Most readers are familiar with $\langle \Re, +, \times \rangle$; thus, we present the properties of $\langle \mathcal{S}', \oplus, \otimes \rangle$ and $\langle \Re, +, \times \rangle$ comparatively, breaking the discussion into properties of $\langle \Re, +, \times \rangle$ missing in $\langle \mathcal{S}', \oplus, \otimes \rangle$, shared properties, and properties of $\langle \mathcal{S}', \oplus, \otimes \rangle$ beyond those of $\langle \Re, +, \times \rangle$.

6.3.1 Weaknesses of $\mathcal{S}'$

$\langle \mathcal{S}', \oplus, \otimes \rangle$ contains most of the field axioms. The major weakness is the lack of an additive inverse for any element of $\mathcal{S}'$ except [0] (i.e., if $v \in \{+, -, ?\}$, there is no $w \in \mathcal{S}'$ such that $v \oplus w = [0]$). Thus, whatever $\ominus$ is, it can't be the inverse of $\oplus$. As a result the sign algebra does not meet any of the normal classifications of field, ring, or even group. Thus many lessons of algebra cannot be applied in a straightforward manner. Instead a detailed analysis is required, directly from the first principles.

One major consequence of not having an additive inverse is that there is no cancellation law for $\oplus$:

$$s \oplus u = t \oplus u \quad \not\Rightarrow \quad s = t$$

Without it we cannot in general solve systems of sign equations by subtracting equa-

tions and cancelling terms (as is done in Gaussian Elimination[50]). Furthermore, addends cannot be moved between sides of an equation:

$$s \oplus t = u \not\Rightarrow s = u \ominus t$$

Consequently we cannot always solve for a particular variable in a qualitative expression using standard techniques developed for real expressions.[3] The impact of this property for invention is that there are some variables in interactions that we cannot solve for. Thus some shared variables cannot be used to compose interactions.

Furthermore, real equations are canonicalized by using an additive cancellation rule to move the expressions to one side of the equation:

$$a = b \Rightarrow a - b = 0$$

However, this approach cannot be used for sign equations, because of the missing cancellation law:

$$s = t \not\Rightarrow s \ominus t = [0]$$

Thus, $\langle \mathcal{S}', \oplus, \otimes \rangle$ presents major hurdles to both operations central to the interaction manipulation skills – equation solving and canonicalization.

6.3.2 Commonalities Between $\mathcal{S}'$ and $\Re$

In spite of a missing inverse, $\langle \mathcal{S}', \oplus, \otimes \rangle$ is still quite strong because it shares most of the remaining properties of $\langle \Re, +, \times \rangle$. These properties are central to equation solving and canonicalization.

$\oplus$ has an identity ([0]), is commutative and associative. $\otimes$ has essentially all the

[3]This is why an algebra based only on confluences (i.e., $\langle \mathcal{S}, \oplus, \ominus \rangle$) is so impoverished; its only operators $(\oplus, \ominus)$ are extremely weak.

properties of $\times$ (an identity ([+]), inverse operator ($\oslash$), associativity and commutativity). Like $\times$, $\otimes$ has no inverse for [0], but $\otimes$ also has no inverse for [?] (since [?] contains [0]). This does not present a problem in practice since a subexpression denoting [?] provides no information.

Since $\otimes$ has an inverse, $\otimes$ has a cancellation rule, analogous to $\times$, and multiplicands can be moved between sides of an equation:

$$s \otimes u = t \otimes u \Leftrightarrow s = t \text{ for } t \neq [0], [?] \qquad a \times c = b \times c \Leftrightarrow a = b \text{ for } b \neq 0$$
$$s \otimes t = u \Leftrightarrow s = u \oslash t \text{ for } t \neq [0], [?] \qquad a \times b = c \Leftrightarrow a = c/b \text{ for } b \neq 0$$

The second property above allows us to solve for certain variables or subexpressions of qualitative equations in many situations. Also $\oplus$ distributes over $\otimes$. Thus $\otimes$ can be lifted to the outermost expression of an equation, where the cancellation rule for multiplication can be applied. This extends the set of variables (or subexpressions) that can be solved for. In our limited experience, these properties are sufficient for composing interactions in most situations.

The cancellation rule for $\otimes$ can be used to move the two sides of an equation to one side:

$$s = t \Rightarrow s \oslash t = [+] \text{ for } t \neq [0], [?]$$

This is important for canonicalization, and satisfies the void left by the missing cancellation laws for $\oplus$, highlighted in the last section.[4] Coupling this property with the commutativity and associativity of $\oplus$ and $\otimes$, allows us to represent expressions in a canonical form similar to polynomials on $\Re$.

Earlier we pointed out that $\ominus$ cannot be defined as the inverse of $\oplus$ – there is no

[4]However, cancellation has the undesirable property of introducing an assumption that the denominator is not contain [0] or [?].

inverse. However $\ominus$ is related to $\otimes$ in a manner similar to $-$ and $\times$:

$$\begin{aligned} \ominus u &\equiv [-] \otimes u & -a &\equiv (-1) \times a \\ u \ominus v &\equiv u \oplus (\ominus v) & a - b &\equiv a + (-b) \end{aligned}$$

As a result $\langle \mathcal{S}', \oplus, \otimes \rangle$ and $\langle \Re, +, \times \rangle$ share the following properties important for canonicalization:

$$\begin{aligned} \ominus(\ominus s) &= s & -(-a) &= a \\ \ominus(s \oplus t) &= \ominus s \ominus t & -(a+b) &= -a - b \\ [+] \oslash ([+] \oslash s) &= s \text{ for } s \neq [0], [?] & 1/(1/a) &= a, \text{ for } a \neq 0 \\ [+] \oslash (s \times t) &= ([+]/s) \otimes ([+] \oslash t) & 1/(a \times b) &= (1/a) \times (1/b) \\ & \text{for } s, t \neq [0], [?] & & \text{for } a, b \neq 0 \end{aligned}$$

6.3.3 Properties of $\mathcal{S}'$ not in $\Re$

$\langle \mathcal{S}', \oplus, \otimes \rangle$ has three important properties that allow simplifications not possible in $\langle \Re, +, \times \rangle$, and that are fundamental to the canonicalization and factoring algorithms described in Section 6.4.[5]

First, since $[+] \otimes [+] = [+]$ and $[-] \otimes [-] = [+]$, $\otimes$ *is its own multiplicative inverse*:

$$[s] \oslash [t] = [s] \otimes [t] \text{ for } [t] \neq [0], [?] \tag{6.1}$$

A major consequence is that all occurrences of $\oslash$ in an expression can be replaced with $\otimes$ (as long as the denominator doesn't contain [0]). In addition:

$$\begin{aligned} [s] \otimes [s] &= [+] & &\text{for } s \neq [0], [?] \\ s \otimes t = u &\Leftrightarrow s = u \otimes t & &\text{for } t \neq [0], [?] \end{aligned}$$

[5]Factoring is a subprocedure of equation solving; the necessity for factoring is motivated in Section 6.4.2.

The second property relates to exponentiation. For a qualitative expression s and integer n, let s^n denote $\underbrace{s \otimes s \cdots \otimes s}_{|n|}$ if n is positive or negative, and $[+]$ if $n = 0$. Then the following holds:

$$s^{2i} \otimes s = s \text{ for } i \in \text{integers} \tag{6.2}$$

Thus all expressions raised to a positive or negative odd power are equivalent; likewise, for positive or negative even powers. This allows all exponents i to be reduced to $0 \leq i \leq 2$. Together these two properties allow all sign expressions to be reduced to quadratics. This is used in Section 6.4.1 to develop a compact canonical form and the corresponding canonicalization procedures.

Third, there is a novel cancellation rule for addition:

$$[s] + [s] = [s]$$

As a result of these three properties, common subexpressions are often "absorbed" into a single expression during the simplification process. This results in expressions that are far simpler than their counterpart would be in $\Re$. We return to this issue in Section 6.4.1 during the discussion of the simplification process.

6.3.4 Relating $\mathcal{S}'$ and $\Re$

The remaining task is to examine properties of expressions that use [ ] to combine properties of $\mathcal{S}'$ and $\Re$ (i.e.,$\langle \Re \cup \mathcal{S}', +, \times, \oplus, \otimes, [\,] \rangle$). These are crucial to canonicalization and equation solving for hybrid equations. Many of these operators are related through a set of homomorphisms. For example, recall from the axiomatization of $Q1$

(Section 6.2), that [ ] is a homomorphism of $\Re$ onto $\mathcal{S}$ for multiplication:

$$[a \times b] = [a] \otimes [b]$$

That is, multiplying two real values, and then abstracting their sign, produces the same result as abstracting the signs first, and then using sign multiplication – no information is lost by abstracting earlier (i.e., within subexpressions). The same holds for many of the other operators; [ ] is a homomorphism of $\Re$ onto $\mathcal{S}$ for multiplication, division, minus and exponentiation:

$$\begin{aligned} [a \times b] &= [a] \otimes [b] \\ [a/b] &= [a] \oslash [b] \\ [-a] &= \ominus [a] \\ [a^n] &= [a]^n \end{aligned}$$

The homomorphisms play an important role during canonicalization and simplification. Using the homomorphisms, abstraction ([ ]) are moved as early as possible in an expression, converting real to sign operators. The unique properties of the sign algebra are then used, resulting in a significant compaction of the expression, not possible with the real algebra.

Notice that [ ] is not homomorphic for addition or subtraction. For example, expressing height difference as $[H_v] \ominus [H_b]$ is weaker than $[H_v - H_b]$ (e.g., consider $H_v = 8, H_b = 7$). The relationship for addition and subtraction are:

$$\begin{aligned} [a] \oplus [b] &\text{ contains } [a + b] \\ [a] \ominus [b] &\text{ contains } [a - b] \end{aligned}$$

Recall that "x contains y" means that the interval denoted by y is a subinterval of x. Since these properties do not preserve equivalence they are not used during

simplification.[6]

This property of addition and subtraction sheds light on a crucial problem with the standard approaches to qualitative reasoning – they over-abstract. The mistake is that a qualitative equation is traditionally produced from a real equation by replacing each operator with its sign equivalent and each variable v with $[v]$. Thus, in the punch bowl example we would be forced to represent height difference as $[H_v] \ominus [H_b]$. But this expression is useless – since height is never negative and rarely zero, the value will almost always be [?]. We have solved this problem by using a hybridization of real and sign operators within expressions, allowing an equation to use the infinite resolution of the reals for restricted subexpressions where required.

6.4 MINIMA – Algebraic Manipulation Skills

The next step is to incorporate the above properties into a symbolic algebra system that embodies operations for combining and comparing interactions. MINIMA is a symbolic algebra system for Q1 that supports the required operations. MINIMA is a qualitative analog of the symbolic algebra system Macsyma[32], and in fact uses Macsyma to manipulate subexpressions in $\langle \Re, +, \times \rangle$.

We discuss the two most important operations performed by MINIMA for comparing and combining interactions: simplification and equation solving (i.e., solving for a variable or subexpression). Recall that the purpose of simplification is to reduce superfluous syntactic structure in the interactions, making them easier to manipulate and compare. By making the simplifier sufficiently powerful (i.e, reducing expressions to a unique canonical form), comparison operations such as identifying equivalent equations is reduced to determining syntactic equivalence. Equation solving is

[6]However these properties are required to determine if one equation or expression subsumes another.

a major component of composition. Composition is performed using substitution of equals for equals on a shared variable. This in turn requires procedures for equation solving in terms of the shared variable.

Recall from the thesis introduction (Section 1.9) that the purpose of the procedural skills is to improve Ibis' performance, by encapsulating those operations that can be performed efficiently. Thus the overriding concern of this development is to ensure the efficiency of the algebraic manipulation procedures. The following is an overview of the basic motivations for the procedures developed in the next few sections.

Like Macsyma, MINIMA provides two approaches to simplification and equation solving where each approach makes a different trade off about performance versus robustness. At this point we do not have enough experience with the approaches to determine which are the most appropriate for interaction-based invention.

The first approach uses the properties analyzed above to perform complete simplification and equation solving, through techniques for qualitative canonicalization and factorization. Canonicalization and factorization are prohibitively expensive in traditional symbolic algebraic systems on $\Re$. However, this is not the case for a sign algebra – MINIMA exploits the properties of Q1 described in Section 6.3.3 to make canonicalization and factorization very efficient in practice.

Nevertheless, using the first approach there is the potential, in the worst case, for a computational explosion. Furthermore, we can pinpoint the cause of this computational explosion to the automatic application of certain principles, such as distributivity. Inventors do not automatically apply these "costly" principles in practice; they only use them through conscious deliberation.

Using this insight, the second approach restricts simplification and equation solving to "obvious" inferences that an inventor appears to apply automatically. Completeness is traded for faster, more intuitive deductions. This second approach is

sufficient for the examples considered in this thesis. These two approaches to simplification and equation solving are the topics of the next two sections.

6.4.1 Simplification and Canonicalization

The purpose of simplification is to eliminate irrelevant structure in the equations. This facilitates comparing and combining equations in two ways. First, it reduces the number of constituents of equations that must be explicitly manipulated, thus speeding up the composition and comparison operations. Second, it restricts the forms the equations can take on, and thus the number of cases that the composition and comparison operators must handle.

The simplifier eliminates structure through a combination of cancellation (e.g., $a/a \Rightarrow 1$), evaluation ($[+] \otimes [-] \Rightarrow [-]$), substitution of known constants, and reduction of subexpressions to a standard form (e.g., $(b \times a) \times c \quad \Rightarrow \quad a \times b \times c$). The operator definitions and properties of Q1, described previously in Sections 5.4, 6.2 and 6.3, provide the tools to perform simplification.

Canonicalization pushes the elimination of irrelevant structure to the point where syntactic differences reflect semantic differences in certain important respects. Specifically, two expressions or equations are syntactically different if and only if they are semantically different.[7] This makes tests for expression and equation equivalence trivial. Furthermore, the use of a canonical form significantly reduces the number of cases that must be considered for other comparison operations required by Ibis. These include determining if an equation is a tautology or inconsistent, and whether one equation is an abstraction of another.

[7]Two expressions are semantically equivalent if they denote the same function. That is, for all assignments of values to variables in the expressions, the two expressions produce the same result. Two equations are semantically equivalent if they denote the same relation. That is both equations are satisfied by the same sets of variable-value assignments.

The following example demonstrates the result of simplification for each of the two approaches. How these simplifications are performed, and technical differences between the two approaches, are discussed below. Consider the simplification of the equation:

$$([a^3/a] \oplus [c_1]) \otimes [-a/(b-4) \times (-c_2)]$$

given the values for two constants:

$$\begin{aligned} c_1 &= 5 \\ c_2 &= [+] \end{aligned}$$

Performing obvious simplifications, the equation reduces to:

$$([a^2] \oplus [+]) \otimes [a] \otimes [b-4] \quad \text{where } [b-4] \neq [0], [?]$$

where the simplifications involve 1) substituting for constants with known values, 2) applying the homomorphism for $\times$, 3) evaluating [ ] on known values, 4) cancelling identities and double negations, and 5) using associativity and commutativity to canonicalize the order of operator arguments.

Performing a complete canonicalization, the above equation is further simplified:

$$[a] \otimes [b-4] \text{ where } [b-4] \neq [0], [?]$$

Consider these two approaches, and the simplification steps performed in this example, in more detail. Both approaches to simplifying equations (or expressions) involve three steps. First, the real subexpressions of an equation (i.e., expressions contained within [ ]) are simplified using the properties described for $\langle \Re, +, \times \rangle$. Next, real operators are transformed into sign equivalents, whenever possible, using the homomorphisms of Section 6.3.4. Finally, the surrounding sign expressions are simplified

using the properties described for $\langle \mathcal{S}', \oplus, \otimes \rangle$.

Mapping from real to sign operators has two advantages: First, the sign of a quantity is often known when the real value isn't. For example, we know density and gravity are positive, independent of substance and planet, thus $[P] = [d \times a \times g \times H]$ simplifies to $[P] = [H]$. Second, the properties of Section 6.3.3 allow significant simplifications in $\mathcal{S}'$ not possible in $\Re$.

Walking through the three simplification steps on the above example, Minima begins with:

$$([a^3/a] \oplus [c_1]) \otimes [-a/(b-4) \times (-c_2)]$$

First, the real subexpressions are simplified. 5 is substituted for c_1, the common numerator and denominator a are cancelled, and the double negation on the right is cancelled, producing:

$$([a^2] \oplus [5]) \otimes [a/(b-4) \times c_2]$$

Next the homomorphisms are applied and real numbers are abstracted to signs, producing:

$$([a]^2 \oplus [+]) \otimes [a] \oslash [b-4] \otimes [c_2]$$

Finally the sign expressions are simplified. In this case, first $[+]$ is substituted for c_2, and $\oslash$ is replaced by $\otimes$:

$$([a]^2 \oplus [+]) \otimes [a] \otimes [b-4] \otimes [+] \text{ where } [b-4] \neq [0], [?]$$

Next the multiplicative identity, $[+]$, is cancelled:

$$([a]^2 \oplus [+]) \otimes [a] \otimes [b-4]$$

Next, to perform a complete canonicalization, distribution is applied:

$$([a]^2 \otimes [a] \otimes [b-4]) \oplus ([+] \otimes [a] \otimes [b-4])$$

The property $s^3 = s$ is used to reduce the cubic term, $([a]^2 \otimes [a])$, to degree 1, and the multiplicative identity, [+], is cancelled:

$$([a] \otimes [b-4]) \oplus ([a] \otimes [b-4]) \text{ for } [b-4] \neq [0], [?]$$

And finally the property $s \oplus s = s$ is used to collapse the common summands:

$$([a] \otimes [b-4]) \text{ for } [b-4] \neq [0], [?]$$

Now consider the specific properties of and differences between the two simplification approaches.

Canonicalization

The first simplification approach (canonicalization) reduces an expression or equation to a pseudo-canonical form[8] analogous to a multivariate polynomial. [9] The canonical form for Q1 equations is broken into a unique form for real and sign subexpressions. Consider the form for each.

In traditional symbolic algebra systems, real expressions are canonicalized by reducing them to one of two unique forms. First, expressions can be reduced to a unique rational form – a fraction consisting of two multivariate polynomials with common factors removed. Second, the numerator and denominator of the fraction

[8] We use the modifier "pseudo" only because the canonical form for sign expressions has not yet been proven unique for multivariate polynomials. However it has been proven unique for univariate polynomials and polynomials all of whose coefficients are multivariate monomials.

[9] A multivariate polynomial is a polynomial in variable v whose coefficients are polynomials not in v.

can be represented as a prime factorization,[10] rather than a polynomial.

We select the second form (prime factorization) on the following basis. The simplification process should use the homomorphisms to map as many real operators to sign operators as possible. This allows the simplifier to exploit the unique properties of the sign algebra to further reduce the expression. Recall that [ ] is homomorphic for multiplication and division, but not addition or subtraction. Furthermore, to apply the homomorphism for multiplication, division, and unary minus, they appear at the top level of a real expression contained with [ ]. For example, the homomorphism can be applied to $[a \times (b + c)]$, but not to $[a + (b \times c)]$. A prime factorization moves $\times$, /, and unary $-$ to the top level, while polynomial form moves $\times$ and / to the inner-most expression; thus, prime factorization is preferred for the real subexpressions.

Next consider sign expressions. The unique properties of $\langle \mathcal{S}', \oplus, \otimes \rangle$ allow all sign expressions to be reduced to *quadratic*, multivariate polynomials, a significant compaction over that possible with real expressions. This form is derived as follows from the properties discussed in Section 6.3: First, we use the property that an expression is its own multiplicative inverse:

$$s \oslash t = s \otimes t$$

to eliminate $\oslash$, replacing it with $\otimes$. This leaves $\oplus, \ominus, \otimes$. Second, operators in $\mathcal{S}'$ are distributive, commutative, and associative; thus, sign expressions involving the remaining three operators can be reduced to polynomials:

$$c_0 \oplus (c_1 \otimes s) \oplus (c_2 \otimes s^2) \ldots \oplus (c_n \otimes s^n) \text{ where } c_i \in \mathcal{S}'$$

[10] A prime factorization of an expression is a factorization such that each factor cannot be factored.

Third, all exponents can be reduced to degree 1 or 2 using the following property:

$$s^{2i} \otimes s = s \text{ for } i \in \text{integers}$$

Thus the polynomials are at most quadratics:

$$c_0 \oplus (c_1 \otimes s) \oplus (c_2 \otimes s^2) \text{ where } c_i \in \mathcal{S}'$$

The coefficients are elements of $\mathcal{S}'$, and $\mathcal{S}'$ contains 4 distinct values, thus there are 64 distinct quadratic forms. Several of these forms are semantically equivalent; for example:

$$[x]^2 \oplus [x] \oplus [+] \;=\; [x] \oplus [+]$$

Thus the final stage eliminates irrelevant terms in the quadratic equation (e.g., $[x]^2 \oplus [x] \oplus [+] \;\Rightarrow\; [x] \oplus [+]$). A complete list of simplification rules is shown in Figure 6.1. The result, for one variable, is 14 semantically distinct expressions.

$$s \oplus [?] \;\Rightarrow\; [?]$$

$$\begin{aligned}
(a \otimes [x]^2) \oplus a &\;\Rightarrow\; a \\
(a \otimes [x]^2) \oplus (a \otimes [x]) \oplus a &\;\Rightarrow\; (a \otimes [x]) \oplus a \\
(a \otimes [x]^2) \oplus (a \otimes [x]) \ominus a &\;\Rightarrow\; (a \otimes [x]^2) \ominus a \\
(a \otimes [x]^2) \ominus (a \otimes [x]) \oplus a &\;\Rightarrow\; \ominus(a \otimes [x]) \oplus a \\
(a \otimes [x]^2) \ominus (a \otimes [x]) \ominus a &\;\Rightarrow\; (a \otimes [x]^2) \ominus a
\end{aligned}$$

Figure 6.1: Simplification rules used to reduce quadratic sign polynomials to a canonical form. In the first rule, s denotes any sign expression. In the remaining rules, x denotes a real expression, and a denotes a coefficient (a sign expression not in [x]).

To canonicalize a hybrid expression, using the forms discussed, Minima converts the real subexpressions to a prime factorization, applies the homomorphisms, and then converts the sign expressions to quadratics as above. To canonicalize a hybrid *equation*, Minima first uses the cancellation law for multiplication:[11]

$$s \otimes t = u \Leftrightarrow s = u \otimes t \text{ for } t \neq [0], [?]$$

to bring both expressions to one side of the equation:[12]

$$s = t \Leftrightarrow s \otimes t = [\dashv\vdash] \text{ for } t \neq [0], [?]$$

and then canonicalizes the combined expression.

Next consider the performance of canonicalization in terms of the three steps. The major cost when canonicalizing real subexpressions is computing factorization, which relies heavily on computing greatest common denominator (GCD). This cost is incurred independent of which canonical form is used, since rational form requires determining common factors between the numerator and denominator. Reducing the numerator and denominator is also potentially costly. In the worst case reduction to polynomial form can produce an exponential number of summands. This is the analogous problem to disjunctive normal form for the propositional calculus.

Experience with systems like Macsyma have shown in practice that constructing polynomials tends to be fast, but performing factorization is prohibitive for large expressions. AI circuit analysis systems using Macsyma have found this cost unacceptable for practical problems such as power supply analysis and circuit parameter selection[45,13]. The symptom is that during analysis the real equations start small,

[11]For real equations the canonical form is a rational equated to 0; one side of the equation is brought over to the other side using the cancellation law for addition. Sign expressions do not have an additive cancellation law; thus, the law for multiplication must be used. This has the unfortunate consequence of introducing an assumption about the denominator being non-zero.

[12]Two sides of an equation are not combined if one expression simplifies to 0.

but grow in size quickly as equations are combined. This growth is reflected in a substantial slow down, due to factorization.

Q1 is a superset of the real algebra, and thus inherits its worst case complexity. However, in practice canonicalization of Q1 equations is much faster than that for real equations, and thus far appears acceptable for interaction-based invention, based on our limited experience.

The reason is that canonicalizing real subexpressions in Q1 equations doesn't incur the same cost as canonicalizing equivalent equations that are purely real. First, real subexpressions within Q1 equations start out smaller, before simplification, than corresponding real equations, since some real operators have been replaced with sign operators. Second, while real equations explode in size as they are combined, as Q1 equations are combined *the real expressions tend to stay small.* The reason is that most of the operators in the real subexpressions are mapped to sign operators using the homomorphisms. The only operators not mapped are addition and subtraction; thus, after canonicalization the remaining real subexpressions are the prime factors, which are usually quite simple – linear or quadratic.

The second component of canonicalization – applying the homomorphism – is a linear time operation. The third component of canonicalization – canonicalizing sign expressions – is much less costly in practice than that for real expressions. First, sign expressions reduce to a polynomial form, rather than rational form. Thus, the costly operations of factoring and GCD, used to cancel common factors, is unnecessary. Furthermore, all polynomials reduce to quadratics, thus expressions tend to stay smaller. The net result is that the homomorphisms reduce the size of real expressions by mapping real operators to sign operators, and then this additional structure is collapsed because of the compact form for sign expressions.

In summary canonicalization facilitates the performance of the interaction manipulation skills by reducing expressions to a unique form. Furthermore, canonicalization

of Q1 equations thus far appears to have adequate performance in practice, unlike canonicalization of real equations. Nevertheless, there is the worst case possibility of a computational explosion.

Partial Simplification

Next consider the second approach to simplification. Comparisons between equations, such as tests for equality, that Q1 can make through canonicalization are beyond those that an inventor would typically make. Furthermore, although canonicalization for Q1 does not appear to be too costly in practice, there is nevertheless the potential for a computational explosion. Thus, canonicalization produces superhuman results with the possibility of disastrous computational cost. An alternative solution is to restrict the simplification process to "obvious" operations -- simplifications that an engineer might perform automatically without much thought.

The conjecture behind the second approach, making "obvious" simplifications, is that the automatic inferences made by an inventor involves making only *local* changes to equation structure. In this approach most of the properties of Sections 6.2 and 6.3, and analogous properties for real expressions, are applied as simple rewrite rules during simplification. Commutativity and associativity are used together to convert binary expressions into n-ary expressions whose arguments are sorted lexicographically (as in $(b \times a) \times c \Rightarrow a \times b \times c$).

The main property *not* used is distributivity, since expanding expressions using distribution can radically change the structure an equation. For example, using distributivity the expression:

$$(a + b)(c + d)(e + f)$$

reduces to:

$$ace + bce + ade + bde + acf + bcf + adf + bdf$$

Distributivity is the main source of the exponential explosion that can occur when reducing expressions to polynomial form. Eliminating distributivity removes that possibility of degrading Ibis' performance, but at the cost of robustness.

There are two rough arguments for why this approach might approximate simplifications typically performed by inventors. First, these simplifications are analogous to those which Macsyma performs automatically as it reads in equations and expressions. Macsyma is routinely used by scientists and engineers. During the development of this automatic simplification component it was essential that the automatic simplifications were limited to those that the users find obvious. Second, in [23] Levesque proposes logics of explicit and implicit belief. The logic of implicit beliefs models logical inferences (e.g., inferring equivalences between logical expressions) that one makes automatically without conscious thought. While the logic of explicit belief models deductions that a person normally guides through conscious deliberation. Many of the properties used by Minima's simplifier are analogous to those proposed in the logic of implicit belief, which was intended to capture obvious logical inferences (e.g., this logic included associativity and commutativity but not distributivity). Given the similarities between logics and algebras, it is plausible that logical inferences performed automatically by people carry over to analogous inferences performed automatically during algebraic manipulation.

Finally, consider the effect of the simplification approach on the performance and robustness of the procedural skills. In this approach completeness has been sacrificed for the assurance that simplification can be performed efficiently. This is a necessary trade off; procedural skills are given autonomy over particular operations under the assumption that they are not exorbitantly expensive. To achieve robustness while maintaining performance, the control of expensive operations, such as distribution,

must be put under the control of the higher level problem solving strategy. This has not been necessary in this thesis, since the "obvious" simplifications approach has been adequate for the examples described in this document. Nevertheless, engineers use distributivity, but guide its application carefully. Understanding the process of this guidance is a topic of future work.

6.4.2 Equation Solving and Factoring

The process of composing interactions was demonstrated in Section 3.5.5 when testing whether a proposed interaction topology produces the desired behavior. Recall that combining interactions involves: identifying a shared variable, solving for that variable in one equation (equation solving), substitution, and simplification. Identifying shared variables is performed by Iota, and is discussed in Chapter 8. The task is determining whether two syntactically distinct expressions denote the same variable. Substitution is straightforward, once we have defined our algebra so that "=" is an equivalence relation. Simplification was discussed in the last section. The remaining step, equation solving is the topic of this section.

Given an equation in $\langle \Re, +, \times \rangle$ it is possible to solve for any variable. However, this is not the case for $\langle \mathcal{S}', \oplus, \otimes \rangle$: Since there is no cancellation for addition, addends cannot be moved between the left and right sides of an equation.

$$s = t \;\not\Rightarrow\; s \ominus t = [0]$$

However, it is often possible to solve for certain variables and subexpressions. Cancellation can be performed for multiplication (Section 6.3.2); thus, we can solve for any variable or subexpression that is an argument of a top-level multiplication (or division).

$$s = t \;\Rightarrow\; s \otimes t = [+] \text{ for } t \neq [0], [?]$$

For example, in the equation $(s \oplus t) \otimes u = v$ we can solve for $s \oplus t$:

$$(s \oplus t) \otimes u = v \;\Rightarrow\; (s \oplus t) = v \otimes u \text{ for } u \neq [0], [?]$$

In addition we can solve for u and v; however, we cannot solve for s or t. To identify "obvious" variables and subexpressions that can be solved for, local simplifications to an equation are made using the technique described above, and then the multiplicands of top-level expressions are identified if they exist.

A second, more powerful approach, maximizes the resolution of the subexpressions that can be solved for. This in turn increases those interactions that can be composed, thus maximizing robustness. Of course the cost is in the complexity of the equation solving approach (discussed below). The increased complexity degrades performance.

This more powerful approach involves computing the prime factors of the sign expressions in a qualitative equation, and then solving for any of the factors. Given that the sign expressions are already canonicalized, this requires converting the quadratics to a prime factorization. Traditionally factorization requires computing GCD, which is very expensive. However, factorization using standard GCD algorithms cannot be used for sign expressions, since GCD algorithms rely on cancellation. Instead we use a much simpler approach. Since sign expressions can be reduced to quadratics, it is relatively inexpensive to determine the factors by generate and test.

The factorization of a quadratic is of the form:

$$k_0 \oplus (k_1 \otimes s) \oplus (k_2 \otimes s^2) = ((a \otimes s) \oplus c) \;\otimes\; ((b \otimes s) \oplus d)$$

where k_2, k_1, k_0, a, b, and c are qualitative expressions and $k_2 = a \otimes b$, $k_1 = (a \otimes c) \oplus (b \otimes d)$ and $k_0 = c \otimes d$. Thus to factor a quadratic we generate a,b,c and d by factoring the coefficients k_2 and k_0, and then distribute k_2's factors between a and b, and k_0's factors between c and d. To test we compute the polynomial corresponding

to $(a \otimes c) \oplus (b \otimes d)$ and compare it with k_1. Quadratics are sufficiently infrequent that this strategy is quite acceptable in practice. Thus both performance and robustness are achieved.

6.5 Example: Composing Interactions

Minima's procedures for simplification and equation solving, described above, are demonstrated for composition by the following example. Recall for the punch bowl problem that the desired behavior is:

$$[H_v - H_b] = [dH_b/dt]$$

where H_v and H_b are the fluid heights in the vat and bowl. Furthermore, we know for a container that pressure at the container bottom is linearly related to fluid height as follows:

$$P = d \times g \times H$$

and that density, d, and gravity, g, are always positive. Suppose we decide to combine the interactions by substituting for H_v and H_b in the left hand expression of the desired behavior. The interactions are composed by first solving for H:

$$H = P/(d \times g)$$

substituting for H_v and H_b:

$$[P_v/(d \times g) - P_b/(d \times g)] = [dH_b/dt]$$

and simplifying – first simplifying real subexpressions, then applying homomorphisms, and finally simplifying sign expressions:

$$[(P_v - P_b)/(d \times g)] = [dH_b/dt]$$

$$[P_v - P_b] \oslash ([d] \otimes [g]) = [dH_b/dt]$$

$$[P_v - P_b] = [dH_b/dt]$$

This represents one composition step for the punch bowl example. The complete sequence of interaction compositions for this example was given in Section 3.5.5.

We conclude by briefly considering how some of the concerns for algebraic reasoning carry over to assertional reasoning.

6.6 Summary

Ibis' facility at manipulating interactions is a key component of its performance. By embodying the first principles of interactions into a set of efficient procedural skills, Ibis's design strategies are able to focus on issues more central to identifying innovative devices. In this chapter we explored the qualitative algebraic component of Ibis' skills for manipulating interactions.

Developing this algebraic component represents both a central component of our theory of interaction-based invention, and a key advance in the qualitative reasoning community. We observed that, even after a decade of research in qualitative algebras, very little is known about their algebraic properties, or how qualitative equations can be manipulated symbolically. More importantly, there has been a growing belief that qualitative algebras can never be made powerful enough for tasks like interaction-based invention.

This belief was refuted here by developing a theory of qualitative algebraic reasoning, powerful enough for interaction-based invention. This theory consists of:

- a novel hybrid qualitative/quantitative algebra *(Q1)*,
- a compact axiomatization of Q1, constituting the first principles of interactions, and
- a symbolic algebra system for Q1 *(Minima)*, which captures the qualitative algebraic component of Ibis' skills for composing and comparing interactions.

The power of Minima is derived from four distinctive properties of Q1:

- Q1 is a hybrid qualitative-quantitative algebra, that merges a sign algebra and the traditional algebra on the reals. Using a combination of real and sign operators allows an inventor to select the appropriate abstraction level to describe a interactions along a qualitative-quantitative spectrum, thus avoiding problems of over and under abstraction.
- Q1 incorporates a complete sign algebra, including multiplication and division. This allows any of the real operators to be selectively abstracted. In addition, much of Minima's power is derived from exploiting unique properties of the complete sign algebra previously overlooked by other researchers.
- Q1 treats "=" as a true equivalence relation, unlike other qualitative algebras[12, 63,51,15,21]. This makes it possible to substitute equals for equals, the basis for combining interactions.
- The unique properties of Q1 are used to develop efficient symbolic algebra procedures for canonicalization and equation solving. These constitute the basic algebraic skills necessary to compose and compare interactions. Crucial to the efficiency of the procedures is the discovery that the canonical form for sign expressions is extremely compact – it is a quadratic.

This completes our description of Ibis' qualitative algebraic skills for manipulating interactions (Minima). In the next chapter we present the first principles describing what interactions are producible. Then Chapter 8 completes the picture by exploring Ibis' requisite skills for constructing interaction topologies (Iota).

Chapter 7

First Principles of Physics

This chapter describes the first principles used to relate intensional interactions to physical structure. The first principles used are a straight forward application of standard physics, and consist of the component models and connection laws of circuit theory[4] and physical system dynamics[41]. We have already used many of these principles informally in past chapters. This chapter provides the precise statement of the models and laws used by Ibis, introducing the necessary notation and terminology.

7.1 The Appropriate Abstraction for Principles

Traditionally engineers describe the component models and connection laws at a quantitative level. On the other hand, much of the reasoning performed by Ibis is qualitative. Thus, before defining these models and laws, one issue to be addressed is: at what level of abstraction should we describe the intensional interactions that

they contain?

Recall that intensional interactions are described by equations in the hybrid qualitative-quantitative algebra, Q1. In Chapter 5 we showed that Q1 allows interactions to be described at many levels of abstraction along a qualitative-quantitative spectrum. Our decision is to express the first principles at a purely quantitative level whenever possible, otherwise at the most detailed level available, and then allow the problem statement and design strategies to determine the abstraction level appropriate to the task.[1] In the remainder of this section we consider the rationale for this decision.

Circuit theory and system dynamics present their models and laws at a purely quantitative level. This level is appropriate for a quantitative analysis of device behavior, which is the primary role that circuit theory and system dynamics were developed for. On the other hand, much of an inventor's reasoning, when searching for fundamentally different solutions, is at a more qualitative level. For example, we saw in Chapter 2, when analyzing the account of the punch bowl problem, that the desired behavior and many of the intermediate interactions were described qualitatively. Furthermore, we have already observed that reasoning at a qualitative level has the significant advantage of suppressing irrelevant detail, allowing the inventor to focus on fundamental differences during the process of invention. Finally, we are interested in being able to reconstruct historically significant devices like Philon's lamp. It is doubtful that these early inventors had precise quantitative models.

These points argue that intensional interactions must be reasoned about qualitatively. Thus, it is plausible that the initial specification of the first principles is

[1]More precisely, when verifying a solution, Ibis propagates through the interaction topology a qualitative expression which is part of the desired interaction (i.e., the problem statement). At each step of this process, Ibis combines an interaction with this qualitative expression, using the qualitative algebra system Minima (Chapter 6). When Minima performs the composition, it removes any of the quantitative details that do not affect this qualitative expression's value. The net effect is that the qualitative interactions have been raised to the appropriate level of abstraction for the task.

also qualitative. However, there are two caveats to initially specifying the interactions qualitatively. First, there is no single "qualitative level" that all interactions can adequately be specified at. For example, the motivation for constructing the hybrid qualitative-quantitative algebra, Q1, is that the inventor must reason about interactions at multiple levels of abstraction (Section 2.2). Second, even for a single interaction within a model or law, no one level of abstraction can be specified *a priori* which is appropriate for all design tasks. The level required depends on the abstraction level of the behavior desired for a particular design problem.

Thus, the solution we have taken is to specify the first principles at the most precise level available (i.e., quantitatively), and then use the abstraction level of the desired behavior to determine those quantitative details that are unnecessary and can be eliminated. An additional advantage of this solution is that it avoids an opportunity to build the answer to a design problem into the first principles by tailoring their levels to only the example problems. Note however that our theory of invention does not depend in any way on the fact that these first principles are specified quantitatively. Our approach constructs devices equally well using more qualitative principles, given that these principles are sufficiently detailed to determine that the device produces the behavior desired.

Next we describe the representation and notation used to describe the first principles, as well as other statements and principles used during the remainder of the thesis. This representation combines first order logic and the Q1 algebra using an infix notation. We then use this representation to describe models and laws for hydraulics.

7.2 Notation for Principles

First principles are described by first order wffs (more specifically horn clauses), where some of the propositions are Q1 equations, used to describe intensional interactions.

More precisely, currently each first principle reduces to a horn clause whose consequent is a q1 equation and whose antecedents are propositions. Each predicate in these propositions correspond to a class defined in Iota (Chapter 8).

We use a few special notational conventions that make these principles easier to read and simpler to manipulate. The following model for a pipe provides a demonstration of this notation:

```
(let* (pi := any(Pipe);
       nt1 := node_of(t1(pi));
       nt2 := node_of(t2(pi)))
  [R(pi)] = [+];
  (let (t := any(time))
    Q(t1(pi))(t) + Q(t2(pi))(t) = 0);
  (let (t := any(Time))
    Pd(nt1,nt2)(t) = R(pi) x Q(t1(pi))(t)));
```

The model begins by defining three variables *pi*, *nt*1 and *nt*2. *pi* is a universally quantified variable ranging over the set of all pipes. *nt*1 and *nt*2 denote nodes which the two ends of the pipe, *pi*, are connected to. The body of the model specifies three interactions produced by *pi*. The second two interactions hold for every point in time (i.e., the variable *t* ranges over all elements of time). Next consider the notational conventions used in this and similar models, and their underlying semantics.

First, the interactions and wffs are described predominantly using infix notation. This makes it easier to read the Q1 equations. The end of a wff is denoted by a semicolon ";" and a unary proposition "P(a)" is sometimes written "a instance P". For example, we describe the statement, "if P1 is a pipe then its fluid resistance, R, is positive" as follows:

```
P1 instance Pipe implies [R(P1)] = [+];
```

Second, the description of interactions and formulas can get quite large. These are simplified by using forms analogous to commonlisp "LET" and "LET*" (pages 110-112 of [46]) to name common subexpressions. For example, the following expression describes two of the interactions produced by a pipe P1:

```
(let (nt1 := node_of(t1(P1));
      nt2 := node_of(t2(P1)))
  [R(P1)] = [+];
  Pd(nt1,nt2) = R(P1) x Q(t1(P1)));
```

And is logically equivalent to:

```
[R(P1)] = [+] and
 Pd(node_of(t1(P1)),node_of(t2(P1))) = R(P1) x Q(t1(P1));
```

Notice, that the syntax is slightly changed from commonlisp to accommodate infix notation. First, for each binding the variable and value are separated by ":=" (e.g., "var := value"). Second, successive bindings and expressions in the body are separated by ";".

The semantics of "LET" and "LET*" is only slightly changed from lisp. For example, the scoping and binding rules are the same, and "LET" performs parallel assignment, while "LET*" performs sequential assignment. One difference is that the sequence of expressions in the body of a LET or LET* denotes a conjunction. The second difference is that, in cases where commonlisp evaluates an expression (e.g., a binding value or expression in the body), instead the expression is mapped from the syntactic form to a unique object it denotes (e.g., term, formula or interaction). This matters for expressions *any*(P) and *some*(P), discussed below, which generate unique objects for each distinct occurrence of the same expression.

Finally, type restricted quantification is specified using a syntactic variant of

skolemization, which exploits the features of LET for naming quantifiers. More specifically, every occurrence of the expressions *any*(P) and *some*(P) generate unique objects denoting quantifiers. *any*(P) denotes a universal that ranges over P, and *some*(P) generates an existential that is an instance of P. These objects may be named using LET. This naming is necessary in order to use the quantifier in several places. For example:

```
(let (x := any(P))
  R(x,x);
  S(some(Q), some(Q)))
```

is logically equivalent to:

```
(forall (x)
  P(x) implies
    (R(x,x) and
      (thereis (y,z) Q(y), Q(z) and S(y,z))))
```

7.3 First Principles of Hydraulics

Next we summarize the set of first principles of hydraulic systems that were used for the punch bowl problem in Chapter 3. The principles presented here are a straightforward application of circuit theory and physical systems dynamics (see [4,41] for further details). The objective of this section is simply to review the important concepts, to introduce necessary terminology, and to provide the precise form of the models and laws used.

Recall that in circuit theory and physical systems dynamics, a device's physical structure is represented by a network composed of a set of components connected to

common nodes. More specifically, each component, C, has a set of terminals (e.g., $t1(C), t2(C), \ldots$). Each terminal is connected to exactly one node, and a node may be connected to any number of terminals. A connection between a terminal $t1(C)$ and node N is denoted $connect(t1(C), N)$, and the node that $t1(C)$ is connected to is denoted $node\_of(t1(C))$:

```
(let (t := any(terminal);
  n := any(node))
  connect(t,n) iff node_of(t) = n);
```

A component interacts with the rest of a device exclusively through its terminals, and it only interacts directly with components which share one of its nodes. Interactions between components are through state variables associated with each terminal and node, where each variable is time-varying. For hydraulics the state variable associated with each node is pressure, P, and with each terminal is fluid flow, Q. Q is positive when fluid flows through the terminal into the component. A third state variable, pressure difference (denoted Pd) is measured between two nodes. Each component may also have one or more state variables associated with it (e.g., volume of fluid in a container), describing part of the internal state of the component, and one or more constants, describing fixed physical parameters, such as pipe length and diameter.

For example, a pipe, pi, is modeled as a component with two terminals, $t1(pi)$ and $t2(pi)$, corresponding to the two ends of the pipe. Flows associated with each of the two terminals, $Q(t1(pi))$ and $Q(t2(pi))$, correspond to flows into the two ends of the pipe. $P(node\_of(t1(pi)))$ and $P(node\_of(t2(pi)))$ denote the pressure at the points where each end of the pipe is connected, and $Pd(node\_of(t1(pi)), node\_of(t2(pi)))$ denotes the pressure difference between the two ends of the pipe. A pipe has no internal state variable, but one constant parameter, fluid resistance, which is denoted

$R(pi)$.

Relationships between these basic objects, such as "every terminal has exactly one node", are described using the terminological language of Iota, which is roughly analogous to traditional knowledge representation languages[9,34,6,55]. The relevant definitions for hydraulics, the underlying representation language, and the reasoning system that uses these definitions (Iota), is described in detail in Chapter 8.

The behavior of a device is governed by a set of models and laws, which describe the behavior of each component and connection, respectively. A model describes the internal behavior of a particular component type. This behavior is described by a set of equations on variables directly accessible to the component: its internal state variables, constant parameters, and external state variables. The external state variables are those state variables associated with a component's terminals, and nodes that these terminals are connected to. For example, the model for a pipe, repeated from above, is:

```
(let* (pi := any(Pipe);
       nt1 := node_of(t1(pi));
       nt2 := node_of(t2(pi)))
  [R(pi)] = [+];
  (let (t := any(time))
    Q(t1(pi))(t) + Q(t2(pi))(t) = 0);
  (let (t := any(Time))
    Pd(nt1,nt2)(t) = R(pi) x Q(t1(pi))(t)));
```

The other component type required for the punch bowl problem is a container, used to describe the punch bowl and vat. A container, c, has two terminals corresponding to the top and bottom of the container (i.e., fluid is only allowed to flow through the container's top or bottom). A container has two internal state variables:

volume of fluid in the container, $V_{fluid}(c)$, and height of fluid, $H_{fluid}(c)$. It has three constant parameters: cross sectional area ($A(c)$), density of the fluid in the container ($density_{fluid}$) and gravitational force downward on the fluid ($gravity$). The last two parameters are the same for all containers, and thus are global constants. The model for the container is:

```
(let* (c := any(container);
       nt := node_of(top(c));
       nb := node_of(bottom(c)))
  [A(c)] = [+];
  (let (t := any(time))
    V_{fluid}(c)(t) = H_{fluid}(c)(t) x A(c));
  (let (t := any(time))
    Pd(nb,nt)(t) = density_{fluid} x gravity x H_{fluid}(c)(t));
  (let (t := any(time))
    deriv(V_{fluid}(c)(t),t) = Q(top(c))(t) + Q(bottom(c))(t)));
```

There are two laws describing the interactions between component external state variables (Q, P and Pd). The first law, called *fluid compatibility*, describes the interaction between pressure and pressure difference:[2]

```
(let (n1 := any(node);
      n2 := any(node);
      t := any(time))
  P(n1)(t)-P(n2)(t) = Pd(n1,n2)(t)));
```

The second law, called *fluid continuity*, describes the interaction between fluid flows. Specifically, assuming that fluid is incompressible and a node has no volume,

[2]This may appear obvious, but is not universally true. It is only an approximation that holds when the state variables are changing slowly relative to the speed of light.

then the net flow of fluid into a node is always zero (i.e., flow through a point is conserved). That is, the sum of the fluid flows through terminals connected to a node is zero:

```
(let (n := any(node);
      t := any(time))
  Sum(Q(ti)(t), ti in terminal_of(n)) = 0)
```

The fluid continuity law, described above, involves a summation over arbitrary sets of terminals. Representing and reasoning about this summation requires a representation more powerful than first order logic, and a type of reasoning we prefer not to deal with in this thesis. For simplicity of presentation we restrict a node to having exactly two terminals connected to it. A temporary solution taken in the implementation of Ibis is to fix the number of terminals connected to a node to three and to assign specific names to the terminals connected ($t1\_of, t2\_of, t3\_of$).[3] A larger or smaller number of connections is produced by connecting several nodes together or "capping off" node terminals.[4] In this presentation we further restrict the nodes to have two connections, since that is all that is required for the examples. Given these simplifications, fluid continuity becomes:

```
(let (n := any(node);
      t := any(time))
  Q(t1_of(n))(t) + Q(t2_of(n))(t) = 0)
```

[3] $t1\_of, t2\_of$ and $t3\_of$ satisfy the following property:

```
(let (t := any(terminal);
      n := any(node))
  (node_of(t) = n iff t1_of(n) = t, t2_of(n) = t,
                      or t3_of(n) = t)).
```

[4] A cap is a one terminal device such that:

```
(let (c := any(cap);
      t := any(time))
  Q(end(c))(t) = 0)).
```

This completes the models and laws used in the punch bowl design. The next chapter demonstrates the reasoning used to connect interactions together into a topology, like the interactions mentioned in these models and laws.

Chapter 8

Iota: Constructive Reasoning Skills

Design is a constructive task – it involves building complex artifacts by connection together simpler elements. For interaction-based invention, this involves building interaction topologies, by creating individual interactions, and connecting them through shared variables. Interaction topologies do not have an independent existence. Instead, interactions are produced by creating instances of physical parts, and interactions are interconnected by connecting these parts together. As we saw in Section 3.5.3, creating interactions is straight forward – models and laws are used to identify the appropriate physical parts to be used. Connecting interactions together into a topology requires a more subtle process, which we call *constructive reasoning*, and is the topic of this chapter.

Two interactions are connected together by forcing them to share a variable; that is, by making two of their variables identical. To accomplish this we must be able to answer three questions:

- Are two variables identical?
- Can they be made identical?
- And if so how?

Most traditional problem solvers use syntactic unification to determine whether two variables or terms are, or can be, made the same. However, a purely syntactic approach is inadequate here, since the same variable or term may be denoted by two syntactically distinct expressions. For example, suppose the bottom of a bowl, b, and t1 of a spring, $s1$, are connected to the same node. Then the two syntactically distinct expressions $node\_of(bottom(b))$ and $node\_of(t1(s1))$ refer to the same node.

Instead of a syntactic approach, Ibis determines what variables are, or can be made the same, by reasoning from the relationships imposed by the definitions of the variables and objects involved. For example, Ibis determines that flow into the bottom of a cup, $Q(bottom(cup))$, cannot be shared with flow into the bottom of the vat $Q(bottom(vat))$, by using the definitions of flow, bottom, container and the fact that cup and vat are distinct containers. Specifically, by definition, flow (Q) is a one to one (1-1) function on terminals; thus, for the two variables to be equal, $bottom(cup)$ and $bottom(vat)$ must be equivalent. Likewise, *bottom* is a 1-1 function from containers to terminals; thus, *vat* and *cup* must be equivalent. Finally, cup and vat are unique containers – they are not equivalent. By reasoning from the definitions we conclude that $Q(bottom(cup))$ and $Q(bottom(vat))$ are not and cannot be made equivalent. Reasoning from definitions is called *terminological reasoning* and is the central focus of the knowledge representation community[7], notably[66,5,9,6,34,55]. Since the constructive reasoning performed by Ibis focuses on definitions, it is a special case of terminological reasoning.

The terminological language and reasoning component presented here represents a notable advance in knowledge representation. Over the last decade researchers in

the knowledge representation community have concluded that computing subsumption and classification of concepts is computationally intractable for most languages proposed thus far (e.g., [9,34,55]). More specifically, in [8] Brachman and Levesque observe that there exists a "computational cliff" that most knowledge representation systems fall off – subsumption for even extremely impoverished knowledge representation languages is NP-hard. In this chapter we show that an extremely simple knowledge representation language, and corresponding reasoning component, can be developed that is complete, extremely efficient, and powerful enough for a sophisticated scientific reasoning task – interaction-based invention.

This language and reasoning component, called *Iota*[1], derives its power from the following three sources: First, we identify that for interaction-based invention many definitional constructs used in traditional languages are unnecessary (e.g., conjunctive concepts, decompositions and role number restrictions); eliminating these constructs removes their corresponding computational complexity. Second, instead we identify several definitional constructs as being crucial that, although they have been recognized as useful in the past (e.g., disjointness, and 1-1 functions), have been left out of most logically complete knowledge representation systems[6,34,8] because of complexity concerns. The resulting language, while powerful enough for interaction-based invention, avoids the NP computational cliff. Iota represents a special case of 2-SAT (a polynomial time problem) that can be reasoned with efficiently using standard algorithms.

The presentation of Iota, developed in this chapter, is broken into three parts. First Iota is discussed as a "black box" in terms of the types of questions it answers and the language of relationships between individuals and concepts used to answer them (Section 8.1). We demonstrate how Iota's operations are used by Ibis, when constructing interaction topologies, to connect together pairs of interactions and to

[1]which stands for "Iota think up a name for this system pretty quick."

instantiate links. Finally, we highlight important features of Iota's algorithms and their impact on robustness and performance.

8.1 Iota – Abstract Specification

This section describes Iota as a "black box", in terms of the types of objects it manipulates, the questions it answers, and the information used to answer the questions. The definitional language used inside Iota overlaps with classic knowledge representation languages (e.g., kl-one [9]); thus, it is useful to present Iota within the context of these languages.

Classic knowledge representation systems divide the world into *classes* (e.g., terminology and concepts) and *individuals* (e.g., objects that exist in the world). A language is provided for interrelating classes, relating individuals to classes, and interrelating individuals. The system then answers questions about classes and individuals based on these relationships; for example, is one class a specialization of another class, or is some individual an instance of a class?

The division into classes and individuals corresponds to the terminological-assertional distinction[6] – an abstract notion of truth based on definition, versus truth based on the manifestation of certain conditions in "the world." Roughly speaking, terminological reasoning focuses on the consequences of relations between concepts, while assertional reasoning focuses, (although not exclusively) on the consequences of relations between individuals.

While "pure" knowledge representation systems (as opposed to those referred to as "hybrid systems"[6,55,37]) support some assertional reasoning, it is quite minimal (e.g., they do not support deductive reasoning such as first order propositional resolution, forward or backward chaining). The focus of these systems is on terminological

reasoning – answering questions about classes or individuals by reasoning from definitions (i.e., the relationships between classes). More extensive assertional reasoning, such as resolution, production systems, or propositional inference are incorporated as separate components, resulting in a hybrid system[6,55,37]. Iota strikes a balance between the two. Although a purely terminological component is insufficient, the demands of our *constructive task* is much less than the general deductive capabilities provided by earlier hybrid systems. The primary focus of asserional reasoning is on determining whether two individuals are equivalent or distinct.

8.1.1 Objects Manipulated

Iota uses the same division, as earlier systems, of the world into classes and individuals. Individuals are specific variables (e.g., $Q(bottom(vat))$), and physical parts (e.g., particular components, such as *vat*, and terminals, such as $bottom(vat)$). Classes result from partial specifications of objects through the use of universals (i.e., "$any(c)$"), such as partially specified variables (e.g., $Q(t1(any(pipe)))$), which appear in the potential interaction topology, and physical parts (e.g., $any(pipe)$), which appear in component models and connection laws. Next consider the types of questions Iota answers about these objects.

8.1.2 Queries

Recall that Iota is used to determine whether two variables are or can be made the same. Variables are functions of other types of objects, such as nodes, terminals and components, and variables being the same typically depends on whether these other objects are equivalent.[2] Thus Iota must answer questions about the equivalence of

[2]One should not confuse making variables equivalent, discussed here, with making *variable values* equivalent (e.g., what an equation might express).

objects in general.

We already observed that objects may be completely or partially specified; thus, a question can be about individuals and/or classes. Two variables can be made equivalent unless they are necessarily distinct; thus, Iota answers the following three questions:

- Are two individuals not equivalent ($i1 \neq i2$)?
- Are two classes disjoint ($c1$ disjoint $c2$)?
- Is an individual not an instance of some class (not (i instance c))?

These questions are different from those answered by early knowledge representation systems[9,55,34]. The purpose of these early systems is to construct a classification hierarchy. To accomplish this they answer questions about one class subsuming another and about an individual being an instance of a class. The above questions, answered by Iota, are the dual (i.e., disjointness rather than containment). The use of disjointness is similar to that in more recent representation systems, such as the terminological component of Krypton[6].

We are also interested in determining whether two variables are already equivalent, without requiring additional constraints. For example, this is necessary in order to determine which interactions should be connected together in the topology of existing interactions. This is supported by questions about equivalence and subsumption, similar to those answered by early knowledge representation systems:

- Are two individuals equivalent ($i1 = i2$)?
- Is one class a specialization of another ($c1$ are $c2$)?
- Are two classes the same ($c1$ equal $c2$)?

- Is an individual an instance of some class (i instance c)?

Iota is also used to instantiate links in the topology of potential interactions. This involves determining what equivalences between physical parts result from equating two variables. Finally, instantiating a set of links may lead to an inconsistency. To handle this Iota must be able to identify sets of equivalences and other relations that are inconsistent.

This completes the specification of the queries that Iota answers. Next consider what information is used to answer those questions, and how this information is specified.

8.1.3 Definitions and Assertions

Similar to classic knowledge representation systems, the reasoning used by Iota to answer the above questions focuses on definitions – the assertional capabilities are modest. They concentrate on reasoning about equivalences, with minimal support for reasoning about propositions. The motivation for this is two fold: First, the set of definitions, together with a minimal set of assertional knowledge, is sufficient for building interaction topologies of the type we are interested in. Second, this class of knowledge can be reasoned about efficiently, as is argued in Section 8.3, while the automatic use of more powerful assertional reasoning without guidance leads to a computational explosion. Thus we have chosen a split that is robust for our task, while preserving performance.

Although the focus on definitions is shared with classic knowledge representation systems, the reasoning process and relationships used to answer Iota's questions are a bit different. Traditional knowledge representation systems support an extensive vocabulary for relating classes, and for defining classes completely in terms of other classes (i.e., *defined classes*). But they provide almost no language for relating in-

dividuals. Iota's vocabulary strikes more of a balance – it requires a much smaller vocabulary for relating classes (e.g., no defined classes), but uses a more extensive vocabulary for relating individuals (e.g., $=$, $\neq$). Furthermore, the two vocabularies mirror each other; terms for relating individuals are analogs of terms for relating classes (e.g., "disjoint" for classes and "$\neq$" for individuals).

Consider the specific vocabulary supported by Iota. Iota's language represents those relationships between individuals and classes that satisfy two conditions: First, they are prevalent in the terminology of mathematics and physics used in invention. Second, they facilitate answering questions about objects being distinct.

The objects used for invention have two predominant features critical to our task – they are broken into distinct classes, and there is a strong correspondence between objects (e.g., through 1-1 functions). First, the classes of objects are clearly delineated – structural constituents and variables are broken into disjoint sets. These include terminals, nodes, components, flow, pressure, and pressure difference. Second, the objects in many of these classes are referenced parametrically in terms of members of other objects. For example, fluid flow $Q(t)$ is referenced relative to a terminal t, by function Q, and pressure $P(n)$ is referenced relative to a node n, through function P. These correspondences are also broken into distinct classes. For example, the range and domains of functions from components to terminals are disjoint (e.g., the functions $t1, t2$ and *bottom*). Furthermore, the images of most functions, such as Q, P, *bottom* and $t1$, are distinct – they are one to one. These strong divisions of classes of objects and functions hold by their very definition. Furthermore, this strong division facilitates determining whether two objects can be made equivalent. Thus this definitional information is crucial to determining shared variables.

The division of objects into distinct classes and the strong correspondence between objects are the primary features used to determine whether two variables are necessarily distinct or conversely can be made equivalent. For example, to determine

that $Q(t1(any(pipe)))$ and $Q(bottom(vat))$ cannot be made equivalent we use the fact that Q is a 1-1 function and that the range of $t1$ and *bottom* are disjoint classes of terminals. In general to determine whether two objects are distinct we reason about the classes they are part of (e.g., are they disjoint?), or the objects they are functionally related to and the functions relating them (e.g., as in the above example, are the objects related to them disjoint and the function 1-1?).

Next consider the definitional constructs and their motivation, provided by Iota for describing the salient features of classes, individuals and functions. In this discussion a symbol beginning with i denotes an individual, C a concept, and F a function. Along with each construct we specify its logical import in terms of first order formula and other Iota constructs. Logically an individual is treated as a constant, a concept, C, as a one place predicate(i.e., C(x)), and a function, F, as a two place predicate (i.e., F(x,y)). The formulas corresponding to each construct constitute the first principles of terminological reasoning.

First the basic constructs for relating concepts and individuals are:

C1 disjoint C2	not (thereis (x) C1(x) and C2(x))
C1 are C2	(forall (x) C1(x) implies C2(x))
i instance C	C(i)
i1 $\neq$ i2	not (i1 = i2)
i1 = i2	i1 = i2

These constructs capture disjointness and specialization of classes (disjoint, are), analogously equivalence and disjointness of individuals ($=, \neq$), and instantiation of classes as individuals (instance). These constructs allow Iota to determine that two objects are distinct (i.e., $\neq$ or disjoint), based on the disjointness of their containing

classes. For example, if "a instance S," "b instance R" and "S disjoint R" then "a $\neq$ b". Analogously, if "C are S", then "not (b instance C)."

Successively stronger correspondences between objects (e.g., mappings, functions and 1-1 functions) are defined by the following constructs:

mapping(F)	(forall (x,y) F(x,y) implies domain(F)(x) and range(F)(y))
	(forall (x) domain(F)(x) implies (thereis (y) F(x,y)))
onto(F)	(forall (y) range(F)(y) implies (thereis (x) F(x,y)))
function(F)	mapping(F)
	(forall (x,y) F(x,y) implies F^{-1}(y,x))
	(forall (x1,y1,x2,y2) F(x1,y1) and F(x2,y2) implies
	(x1 = x2 implies y1 = y2) and
	(y1 $\neq$ y2 implies x1 $\neq$ x2))
1-1_function(F)	function(F) and function(F^{-1})

We observed above that many objects manipulated during invention are named parametrically in terms of other objects (e.g., P(node_of($bottom(vat)$))). The properties of functions and mappings allow us to determine that two objects are distinct or equivalent, based on their parameters being distinct or equivalent. For example, F being a 1-1 function tells us that, if two subsets of the domain of F are distinct (equivalent), then their images under F are also distinct (equivalent). The set of relations used during invention are almost exclusively functions or 1-1 functions; mapping is provided primarily as a clean separation of properties.

A function is defined further by constraining its domain, range or images of the domain's subset. This is accomplished by relating them to other classes and individuals using the constructs: are, disjoint, instance, = and $\neq$. These constituents of functions are denoted by the following constructs:

domain(F)	domain of F (a class)

range(F)	range of F (a class)
F(C)	image of a class C under F (a class)
F(i)	image of an individual i under F (an individual)

Earlier we observed that the domain and ranges of many functions are distinct (e.g., $t1$ and *bottom*). We can express this fact by using the above constructs to access classes for the domains or ranges, and relate them using the constructs defined earlier for relating classes (e.g., range($t1$) disjoint range (*bottom*)).

Finally there are additional constructs that do not extend Iota's expressivity, but are easier to use than building the corresponding relations up from the constructs already discussed:

C1 = C2	C1 are C2, C2 are C1
C1 are_unique C2	C1 are C2
	if Cn unique_instance C2, then C1 disjoint Cn
i unique_instance C	i instance C
	if i1 unique_instance C, then i $\neq$ i1

The constructs "are_unique" and "unique_instance" help the user to express constructs compactly, and substantially improve the performance of Iota's algorithms, by avoiding the need to construct cross products from "disjoint" and "$\neq$". For example, they allow a new distinct class of objects (e.g., pipes) and its constituents (e.g., $t1$ and $t2$) to be defined without explicitly naming all the existing objects that they are disjoint with (e.g., we simply write "pipes are_unique components" and "range($t1$) are_unique terminals"). The construct "C1 are_unique C2" states that C1 is a specialization of C2, and is disjoint from all other specialization of C2 that have been specified as unique using are_unique. Analogously, i unique_instance C states that i is an instance of C, and is not equal to any other instance that has been declared unique using unique_instance.

Finally, to make definitions compact we use the following notational convention: Supplying a sequence to a construct, where a single argument is expected, is a short form for applying the construct to each member of the sequence. For example, "1-1_function(f,g)" is equivalent to "1-1_function(f), 1-1_function(g)". Also "domain(f,g) are_unique things" is equivalent to "domain(f) are_unique things, domain(g) are_unique things".

The selection of the above constructs is crucial to Iota's performance. Consider how it avoids the computational cliff that most other complete knowledge systems fall over. The difference is one of focus: Traditional knowledge representation systems focus on *definition*, while Iota focuses on *differentiation.*

More precisely, traditional knowledge representation systems concentrate on *defined classes*, supplying a vocabulary for specifying classes completely in terms of other classes. For example, in Krypton,[6] the statement:

male_elephant CONGEN elephant, male

defines male_elephants to be exactly those things that are elephants and males, and is logically equivalent to:

(forall(x) male_elephant(x) iff elephant(x) and male(x))

These languages provide a wide variety of constructs for specifying defined classes, such as conjunctions, disjoint decompositions, role and number restrictions.

Each of these constructs substantially contributes to the complexity of the reasoning. For example, introducing both conjunctions and disjoint decompositions is sufficient to make computing subsumption NP Hard. The problem is that descriptions of defined classes specify conditions that are, not only necessary, but *sufficient* (i.e., "iff"). Iota avoids this complexity by not including defined concepts. That is,

Iota's constructs specify conditions that are necessary (e.g., "p are q, r") but not ones that are sufficient (e.g., "p iff q and r"). Instead Iota concentrates on constructs that help it to differentiate one class or individual from another. This is facilitated, for example, by using constructs that draw clear divisions between classes or individuals (e.g., disjoint, $\neq$, unique) and reflect the close correspondence between objects and classes (e.g., 1-1_function, =, are, instance).

The use of these constructs are demonstrated in Figure 8.1, which contains the complete set of definitions for the basic constituents of hydraulics (e.g., types of variables, and the physical objects that make up networks of components and connections) and the specific classes of components used in the punch bowl problem (Chapter 2). Notice that the majority of the objects are described by 1-1 functions, and that most of the classes are mutually disjoint. This highlights the claim, made in Section 8.1.3, that the two most predominant features of objects used for interaction-based invention are: they are broken into distinct classes, and there is a strong correspondence between objects. Given the prevalence of this type of information, it is striking that this information is not expressible, and thus unexploitable in most knowledge representation languages that are complete and computationally efficient.

8.2 Using Iota to Connect Interactions

In this section we demonstrate how Ibis uses Iota when constructing interaction topologies. This is motivated by the example of connecting the following two interactions:

```
(let (t := any(Time))
    [Pd(node_of(bottom(vat)),node_of(bottom(bowl)))(t)]
```

```
quantity, physical_constituent are_unique thing;
node, terminal, device are_unique physical_constituents;
physical_constant, state_variable, time are_unique quantity;

function(node_of);
domain(node_of) = terminal;
range(node_of) = node;

1-1_function(t1_of, t2_of, t3_of);
domain(t1_of,t2_of,t3_of) = node;
range(t1_of, t2_of, t3_of) are_unique terminal;

1-1_function(Q, P, Pd);
range(Q, P, Pd) are_unique state_variable;
domain(Q) = terminal;
domain(P) = node;
domain(Pd) = (node X node);

gravity, density_fluid, Patm unique_instance physical_constant;

container are_unique device;
1-1_function(top, bottom, V_fluid, H_fluid, A);
domain(top, bottom, V_fluid, H_fluid, A) = container;
range(top, bottom) are_unique terminal;
range(V_fluid, H_fluid, A) are_unique state_variable;

pipe are_unique device;
1-1_function(t1, t2, R);
domain(t1, t2, R) = pipe;
range(t1,t2) are_unique terminal;
range(R) are_unique state_variable;

cap are_unique device;
1-1_function(end);
domain(end) = cap;
range(end) are_unique terminal;
```

Figure 8.1: Definitions of hydraulics used in selecting a pipe to solve the punch bowl problem.

= [Q(bottom(bowl))(t)]);

and:

```
(let* (pi := any(Pipe);
       nt1 := node_of(t1(pi));
       nt2 := node_of(t2(pi)))
   (let (t := any(Time))
       Pd(nt1,nt2)(t) = R(pi) × Q(t1(pi))(t)));
```

The first interaction describes a relationship between the vat and bowl, and is what we call an existing interaction. The second interaction is part of the pipe model, and is what we call a potential interaction.

To connect the interactions, Iota must determine which pairs of variables from the two interactions are, or can be shared. To accomplish this Ibis supplies Iota three sets of information: definitions, commitments about the design already made, and type restrictions specified as part of the two interactions. First, at the beginning of the design task Iota is supplied the following definitions for the objects of hydraulic systems, which are relevant to this example:

```
pipe, container are_unique device;
function(node_of);
1-1_function(t1, t2, bottom, Q, Pd);
range(t1, t2, bottom) are_unique terminals;
```

Second, constraints are imposed on individuals by the initial problem statement, and as the design evolves, by instantiating interactions and links. This involves introducing new objects, restricting their types, describing their functional relationship to other objects, and specifying that subsets of them are equivalent or distinct. In

our example, the following constraint has already been imposed by the punch bowl problem statement:

vat, bowl unique_instance container;

That is, both vat and bowl are instances of container, and vat $\neq$ bowl.

Finally, interactions involving universal quantifiers (e.g., "any(pipe)") typically include type restrictions on those quantifiers.[3] The type restrictions for quantifiers in the two interactions are supplied to Iota. In our example the description of the two interactions includes three type restrictions – the expression "t := any(Time)" in the first interaction, and the two expressions "pi := any(Pipe)" and "t := any(Time)" in the second interaction. In this example, to avoid confusion between the two lexically scoped variables with the same name, "t", we refer to the variable appearing in the first interaction as "ta" and the variables appearing in the second as "pib" and "tb". These three type restrictions are provided to Iota by the following statements:

t1a are Time;
t1b are Time;
pib are Pipe;

Given this information, Ibis tries to connect the interactions, by asking Iota, for each pair of variables occurring in the two interactions, whether they are equivalent, necessarily distinct, or neither. For example, if Iota is given the pair of variables Q(t1(pi))(t) and Q(bottom(bowl))(t), then it answers that they are necessarily distinct (i.e., disjoint). More specifically, by definition the range of t1 and bottom are distinct subclasses of the class terminal, and Q is a 1-1 function. Thus the images of

[3]Recall that logically a type restriction "pi := any(pipe)" is equivalent to the proposition "pipe(pi)". The logical form of a model, law or interaction reduces to a set of horn clauses whose consequent is a Q1 equation, and whose antecedents are propositions. The type restrictions are antecedents whose predicate has a corresponding class in Iota.

Q applied to the images of t1 and bottom must be distinct. The conclusion is that Q(t1(pi))(t) and Q(bottom(bowl))(t) cannot be consistently shared.

On the other hand, given two variables:

Pd(node_of(bottom(vat)),node_of(bottom(bowl)))(ta)
Pd(node_of(t1(pib)),node_of(t2(pib)))(tb)

Iota infers neither that they are equivalent nor distinct. More specifically, *Pd* is a 1-1 function on pairs of nodes; thus, the two variables are the same if the nodes are equivalent, and distinct if the nodes for either argument are distinct. Each of these nodes are specified as the nodes of particular terminals, using the function node_of. Although each of these terminals are necessarily distinct, node_of is not a 1-1 function. As a result the four nodes may or may not be equivalent.

Thus, Ibis connects the two interactions, with a single link going between the pressure difference (Pd) from node_of(bottom(vat)) to node_of(bottom(bowl)) and pressure difference from node_of(t1(pib)) to node_of(t2(pib)). This demonstrates the first use of Iota, determining connections between interactions. The second use is imposing new connections, by instantiating links. This results in constraints being imposed on physical structure. Consider an example. Suppose that the pipe interaction has been instantiated with pipe *P*1. Iota is told:

P1 unique_instance Pipe;

and the following interaction (along with the other pipe interactions) has been created:

(let (t := any(Time))
Pd(node_of(t1(pi)), node_of(t2(pi)))(t) = R(pi) × Q(t1(pi))(t))

In this expression, we refer to the variable "t" in the let as "tc" to distinguish it from the other occurrences of "t" (i.e., "ta" and "tb").

Next suppose that the above link is instantiated. To accomplish this, Iota is given the equivalence:

Pd(node_of(bottom(vat)),node_of(bottom(bowl)))(ta)
= Pd(node_of(t1(P1)),node_of(t2(P1)))(tc).

Using the fact that Pd is 1-1, Iota infers:

t1a = t1c;
node_of(bottom(vat)) = node_of(t1(P1));
node_of(bottom(bowl)) = node_of(t2(P1));

More extensive examples of instantiating links, as well as recognizing inconsistent links, were provided in Sections 3.5.3 and 3.6.

Thus far Iota has been treated as a black box. The next section outlines the algorithms used to implement the abstract specification for Iota (Section 8.1).

8.3 Algorithms for Answering Queries

Consider how Iota uses the relations specified by the definitional constructs listed in Section 8.1 to answer questions about whether objects are distinct or equivalent. Most of the algorithms are straightforward – the work involved was in identifying the set of definitional constructs that enable robustness and performance. In this section we sketch informally the basic structure of the algorithms. To avoid getting bogged down in technical details, the impact of key algorithmic components on completeness and computational complexity are presented informally.

Recall from Section 8.1 that Iota supports the following inferences. First, Iota determines if two objects are distinct, by answering one of the following three questions:

- Are two individuals not equivalent (i1 $\neq$ i2)?
- Are two classes disjoint (C1 disjoint C2)?
- Is an individual not an instance of some class (not (i instance C))?

Second, tests for equivalence are based on the following questions:

- Are two individuals equivalent ($i1 = i2$)?
- Is one class a specialization of another ($c1$ are $c2$)?
- Are two classes the same ($c1$ equal $c2$)?
- Is an individual an instance of some class (i instance c)?

Finally, Iota determines all consequences of $=,\neq$ based on the definitions, recognizes any inconsistencies, and provides dependency information used to support dependency directed backtracking.

The description of the algorithms used to answer these questions is broken into four parts. First we discuss the treatment of the basic constructs: $\neq$, are, instance, disjoint. Second, equality is incorporated through equivalence classes. Third, *disjoint* and $\neq$ are re-encoded compactly by sets of distinct objects. Finally, the role of functional relationships is discussed.

Kernel Algorithm: Competence and Robustness

First the kernel of Iota's algorithms use constraints involving $\neq$, are, instance and disjoint, to prove that two objects are necessarily distinct in response to a query.

This is inferred in two ways: First, two individuals are distinct if they are explicitly related by $\neq$. Second, two objects are distinct if contained by (or are) classes that are disjoint. By "contained" we mean that, if the object is an individual it is an instance of the class, and if the object is a class, it is a specialization ("are"). To determine that two objects are contained by disjoint classes, Iota traces outward from the two objects, along "are" and "instance" relations. If the two paths reach separate classes that are disjoint then the objects are distinct.

This kernel is efficient and complete. To see this we restate the problem in terms of propositional satisfiability, and then use the complexity result of complete satisfiability algorithm. Specifically we use set of support resolution for 2-SAT.

First consider the logical restatement of Iota's database and queries in clausal form. All the relations – instance, are, disjoint, and equal – map to a set of binary and unit clauses:

C1 disjoint C2	(forall (x) not C1(x) or not C2(x))
C1 are C2	(forall (x) not C1(x) or C2(x))
C1 equal C2	(forall (x) not C1(x) or C2(x))
	(forall (x) C1(x) or not C2(x))
i instance C	C(i)

where each individual i is an atomic constant symbol.

Each of Iota's queries is mapped to a question of satisfiability, by augmenting the set of clauses based on the form of the question. The answer to each question is true if the augmented set of clauses is constant. The three questions about distinct objects are encoded as follows: "not (a instance S)?" is encoded by adding S(a) to the set of clauses. "a $\neq$ b?" is encoded by substituting all occurrences of b for a in the set of clauses. "R disjoint S?" is encoded by introducing a unique individual "a", and adding the unit clauses R(a) and S(a).

The questions about equivalence are encoded as follows: "a instance S?" becomes the clause not(Sa), "S equal R?" becomes the queries "S are R?" and "R are S?," and "S are R?" is encoded by introducing a unique individual "a" and adding the unit clauses S(a) and ¬R(a). Finally we defer the question "a = b?" until "=" is introduced into the vocabulary.

Given this encoding, next consider complexity. Most knowledge representation languages are a superset of 3-SAT, and thus are NP-hard.[8][4] However, for Iota each of the constraints and queries reduces to unit or binary clauses, and thus is a subset of 2-SAT[5] – a polynomial time problem.

Furthermore, each of the questions is posed as a set of unit clauses. We could answer them using set of support resolution – a complete inference strategy. Using this approach, each resolution step would combine a unit and binary clause, producing a unit clause. Thus the problem reduces to unit clause resolution based on set of support, which has complexity O(n) on the number of clauses. In addition, the sequence of resolution steps correspond to the algorithm described above, which traces paths from the two objects being compared, along "instance" and "are" relations, towards opposite sides of a "disjoint" relation. The complexity of the algorithm is further restricted by the depth and "upward" branching of the lattice formed by "instance" and "are" relations. Both of these are quite small (i.e., about 3) for the definitions sufficient to represent objects of circuit theory and physical system dynamics. In summary, Iota's kernel algorithm is sound, complete, and efficient, and has complexity O(n) in the number of relations.

[4]nSAT is the problem of whether a set of clauses is satisfiable, where each clause has at most n literals.

[5]The reduction requires one additional step. 2-SAT is defined on propositional clauses, while some of the clauses in our encoding are first order. However, each first order clause can be converted to a set of propositional clauses, without loss of generality. This is accomplished by instantiating each first order clause for the individuals mentioned in a query. Thus the problem reduces to propositional 2-SAT.

Equivalence Classes

The second step in this development is to incorporate equality between individuals into the kernel. Using the equivalence relations, sets of individuals are collected and maintained into equivalence classes.[6] When a question is asked involving an individual, rather than looking at the individual's "instance" relations only, the kernel algorithm traces through "instance" relations associated with any member of the equivalence class. Furthermore, the question "$i1 = i2$" is answered by consulting the equivalence classes. An inconsistency arises when $i1 = i2$ and $i1 \neq i2$. This can be tested by a simple look-up whenever $i1 \neq i2$ is asserted, or an individual is added to an equivalence class.

Returning to the logical encoding of Iota, this augmentation corresponds to computing the equivalence classes of individuals, renaming each individual in the set of clauses with a representative selected from its equivalence class, and then testing satisfiability on the modified set. The augmentation preserves completeness as long as the equivalence classes are complete with respect to equivalence closure.

Computing equivalence closure is a standard problem with complexity O(n) in the number of equivalence relations. Equivalence classes in Iota are incrementally maintained. There are several motivations for computing equivalence closure immediately as opposed to on demand. First, one of Iota's primary goals is to determine the impact of equating variables on physical structure, where this impact takes the form of equating physical parts (Section 3.5.3). Second, equivalence classes are used to detect inconsistencies (in particular inconsistent physical structures (Section 3.6)). These should be identified immediately in order to stop the invention process from pursuing blind alleys. Third, the same equivalence classes are frequently used in multiple queries. Equivalence classes can be maintained incrementally using standard union find algorithms, and requires at most O(n log(n)) in the total number of

[6]Equality between classes can be similarly encoded.

equivalences added. Furthermore, the examples explored in this thesis produce small equivalence classes, at most four or five members. Thus the addition of equality does not significantly alter performance.

Sets of Distinct Objects

The third augmentation is the explicit introduction of sets of distinct objects, and is used to improve performance. Although equivalence classes tend to be small, the number of mutually distinct objects is quite large, as is evident in the definition for hydraulics (Figure 8.1). Describing mutually distinct objects using $\neq$ and disjoint produces an explosion in the problem size which results in a significant degradation of performance. This in turn can have a significant impact on the competence of the interaction manipulation skills. For example, representing n distinct classes requires n^2 relations using "disjoint"; the situation is analogous for "$\neq$". To avoid this problem we represent collections of distinct objects explicitly as sets, leaving relations involving "$\neq$" and "disjoint" implicit. More specifically, each class has associated with it a set of individuals that are distinct instances, and a set of classes that are distinct specializations. Membership in these two sets are specified by the constructs "unique_instance" and "are_unique", described in Section 8.1. Tests for "$\neq$" and "disjoint" require additional tests for membership in the corresponding sets. [7] Tests for consistency translate to the condition that these distinct sets and equivalence classes overlap by no more than a single individual.

[7] More precisely two objects *i1*,*i2* are distinct if members of their corresponding equivalence classes are in one of these "distinct" sets. An inconsistency arises if at least two members of a single equivalence class are members of a "distinct" set.

Functional Relationships

The final augmentation is the incorporation of functional relationships. As was discussed above, because of the strong correspondence between objects, functional relationships are fundamental to inferring that objects are distinct or equivalent. For example, two individuals are distinct, such as $P(n1)$ and $P(n2)$, if both are the image of the same 1-1 function (P), and the inverse images of the individuals are distinct ($n1 \neq n2$).

Recall that information about functional relationships appears in three forms. First, relations are specified between classes and individuals as follows:

i1 = F(i2)
C1 = (C2)

Second, restrictions are placed on the domain and range of F:

domain(F) are C1
range(F) are C2

Third, the type of F is declared using "mapping","function" and "1-1 function".

To incorporate this information two types of inferences are performed from functional relationships: the first exploits domain/range restrictions and the second exploits restrictions on the images and inverse images of the functional relationships. First, domain and range restrictions are used to infer "instance" and "are" relations. Consider a functional relation, O1 = F(O2), where O1 and O2 are individuals (i1, i2) or classes (C1, C2). Then O1 and O2 must be contained by the range and domain of F:

i1 = F(i2) implies (i1 instance range(F)) and (i2 instance domain(F))
C1 = F(C2) implies (C1 are range(F)) and (C2 are domain(F))

By recording these facts, Iota is able to use constraints on the domain and range of F to answer queries about O1 and O2. For example, this is used during the punch bowl problem to infer that vat fluid height and vat area cannot be shared (i.e., $H_{fluid}(vat) \neq A(vat)$). More specifically, by definition the two classes of state variables H_{fluid} and A are distinct:

range(H_{fluid}) are_unique state_variables;
range(A) are_unique state_variables;

Then given instances of these state variables, $H_{fluid}(vat)$ and $A(vat)$, Iota infers the following from the range restrictions:

$H_{fluid}(vat)$ instance range(H_{fluid});
$A(vat)$ instance range(A);

When asked for the relationship between $H_{fluid}(vat)$ and $A(vat)$, Iota infers that they are instances of distinct classes, and thus $A(vat) \neq H_{fluid}(vat)$.

The second form of inference on functional relations uses relationships between images (inverse images) of a function to relate inverse images (images) of that function. For example, given that two nodes ($n1$,$n2$) are distinct ($n1 \neq n2$), and that pressure (P) is 1-1, then we infer that the images of P – the pressures of the two nodes – are distinct ($P(n1) \neq P(n2)$).

In general, given a mapping F, the following inferences are permitted:

i1 = i2	implies	F(i1) = F(i2)
F(i1) $\neq$ F(i2)	implies	$i1 \neq i2$
i instance C	implies	F(i) instance F(C)
C1 are C2	implies	F(C1) are F(C2)
F(C1) disjoint F(C2)	implies	C1 disjoint C2

where the second rule listed holds only when F is a function.

If F is a 1-1 function, then its inverse F^{-1} is also a function; thus, the inverse of these rules hold:

F(i1) = F(i2)	implies	i1 = i2
$i1 \neq i2$	implies	F(i1) $\neq$ F(i2)
F(i) instance F(C)	implies	i instance C
F(C1) are F(C2)	implies	C1 are C2
C1 disjoint C2	implies	F(C1) disjoint F(C2)

Conceptually, by successively applying these inference rules, relations are "propagated" forward and backward along functional relations. In Iota these inference rules are run opportunistically, in order to catch inconsistencies as soon as possible (e.g., $a \neq a$).

The worst case complexity for propagating *all* relations is $O(n^2)$ in the number of relations. This, however, is extremely conservative. Each chain of functional relations tends to be short, and doesn't branch significantly. Under this assumption the complexity is closer to O(n). In practice the propagation of a single relation (e.g., disjoint, =) typically moves up and down the length of a single chain of functional relations, and thus is O(d) where d is the depth of the chain of functional relations.

Note that during this development we have restrict ourselves to functions on one argument. The language and algorithms are easily extended to handle functions with multiple arguments. Iota is currently being extended to handle functions on multiple arguments. This is necessary during interaction-based invention, for example, to infer that pressures P(n1,n2) and P(n1,n3) are distinct, given that n2 $\neq$ n3. [8]

The four steps described above complete the description of Iota's basic algorithms for determining whether two objects are distinct or equivalent, and for recognizing

[8]Also note that there are a few additional limitations to the current state of Iota's implementation. First, the set of equivalence classes tend to be small; thus, we have not needed to fully integrate an equivalence closure mechanism, although one has been developed. Second, a few of the rules of inference presented in the last augmentation have not yet been incorporated. Both of these limitations are to be fixed in the near future, and pose no foreseeable technical difficulty.

inconsistencies. We have seen in this development that each augmentation is sound, complete, and computationally efficient; thus, the terminological skills provided by Iota satisfy the robustness and performance criteria.

8.4 Summary

The fundamental operation for constructing interaction topologies is connecting two interactions together – what we call constructive reasoning. Ibis connects these interactions by constraining two of their variables to be the same. This is accomplished by equating an appropriate set of physical parts. In this chapter we explored how Iota uses the relationships imposed by the definitions of objects (the *terminology*) to determine where and how variables and other objects can be made the same.

The primary issue in developing a terminological component is to achieve performance without the sacrifice of robustness. Knowledge representation researchers have identified a computational cliff[8], which most knowledge representation languages fall off, that are sufficiently powerful to perform interesting tasks.

In this chapter we showed that an extremely simple knowledge representation language and its corresponding terminological reasoning component, called Iota, can be developed that is complete, extremely efficient, and powerful enough for the demands of interaction-based invention. To avoid the computational cliff, Iota provides only those definitional constructs that: First, provide the best discrimination with respect to identifying distinct and equivalent objects, and second, are the most prevalent in a designer's terminology. For interaction-based invention these are constructs that capture the clear delineation between class boundaries, and the strong (almost always 1-1) correspondence between objects.

Given these key insights, the robustness and efficiency of *Iota* results from three

properties:

- First, Iota excludes terminological constructs, such as defined concepts and value restrictions, that push most traditional languages over the cliff.
- Second, Iota includes terminological constructs which provide significant discrimination, with respect to identifying identical and distinct objects (e.g., disjointness and 1-1 functions), without a severe computational cost. While these constructs are well known in the knowledge representation community as being useful, they are often excluded in languages that try to achieve efficiency and completeness.
- Third, we observe that the core language within Iota is a special case of 2-SAT, which can be solved through efficient polynomial time algorithms. Thus the NP hard cliff that most languages slide over is avoided.

Bibliography

[1] J. Allen and J. Koomen. Planning Using a Temporal World Model. In *Proceedings of the Eighth IJCAI*, pages 741–747, 1983.

[2] D. Barstow. *Automatic Construction of Algorithms and Data Structures.* PhD thesis, Stanford University, September 1977.

[3] D. Bobrow (editor). Special Issue: Qualitative Reasoning about Physical Systems. *Artificial Intelligence*, 13, December 1984.

[4] A. G. Bose and K. N. Stevens. *Introductory Network Theory.* Harper and Row, New York, 1965.

[5] R. J. Brachman. What IS-A Is and Isn't. *Computer*, 16(10):30–36, October 1983.

[6] R. J. Brachman, R. Fikes, and H. J. Levesque. Krypton: A Functional Approach to Knowledge Representation. *IEEE Computer, Special Issue on Knowledge Representation*, 16(10):67–73, October 1983.

[7] R. J. Brachman and H. J. Levesque, editors. *Readings in Knowledge Representation.* Morgan Kaufman Publishers, Inc., Los Altos, CA, 1985.

[8] R. J. Brachman and H. J. Levesque. The Tractability of Subsumption in Frame-Based Description Languages. In *Proceedings of the National Conference on Artificial Intelligence*, August 1984.

[9] R. J. Brachman and J. G. Schmolze. An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, 9:171–216, 1985.

[10] N. J. Cutland. *Computability: An introduction to recursive function theory.* Cambridge University Press, New York, 1980.

[11] J. de Kleer. *Causal and Teleological Reasoning in Circuit Recognition.* AI-TR-529, MIT AI Lab, September 1979.

[12] J. de Kleer and J. S. Brown. A Qualitative Physics Based on Confluences. *Artificial Intelligence*, 24, December 1984.

[13] J. de Kleer and G. J. Sussman. Propagation of Constraints Applied to Circuit Synthesis. *Circuit Theory and Applications*, 8:127–140, 1980.

[14] J. de Kleer and B. Williams. Back to Backtracking: Controlling the ATMS. In *Procceedings of the AAAI*, August 1986.

[15] J. Dormoy and O. Raiman. Assembling a Device. *AI in Engineering*, 1988.

[16] R. Fikes and N. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2(3), 1971.

[17] K. Forbus. Qualitative Process Theory. *Artificial Intelligence*, 24, Dec. 1984.

[18] F. J. Hill and Peterson G. R. *Introduction to Switching Theory and Logical Design.* Wiley, New York, 1974.

[19] D. Johannsen. Bristle Blocks: A Silicon Compiler. In *Proceedings of the 16th Design Automation Conference*, 1981.

[20] L. Joskowicz and S. Addanki. From Kinematics to Shape: An Approach to Innovative Design. In *Proceedings of the 7th National Conference on Artificial Intelligence*, pages 347–52, August 1988.

[21] B. Kuipers. Qualitative Simulation. *Artificial Intelligence*, 29, Sep. 1986.

[22] D. B. Lenat and J. S. Brown. Why AM and Eurisko Appear to Work. In *Proceedings of the National Conference on Artificial Intelligence*, August 1983.

[23] H. J. Levesque. A Logic of Implicit and Explicit Belief. In *Proceedings of the National Conference on Artificial Intelligence*, August 1984.

[24] M. Matson. *Macromodeling and Optimization of Digital MOS VLSI Circuits.* VLSI Memo 85-231, MIT, January 1985.

[25] O. Mayr. *The Origins of Feedback Control.* MIT Press, Cambridge, MA, 1970.

[26] D. McDermott. *Flexibility and Efficiency in a Computer Program for Designing Circuits.* AI-TR-402, MIT AI LAB, June 1977.

[27] C. Mead and L. Conway. *Introduction to VLSI Design.* Addison Wesley, Menlo Park, CA, 1980.

[28] T. Mitchell. Learning and Problem-Solving. In *Proceedings of the Eighth IJCAI*, pages 1139–51, 1983.

[29] T. Mitchell, S. Mahadevan, and L. I. Steinberg. LEAP: A Learning Apprentice for VLSI Design. In *Proc. of the Ninth International Joint Conference on Artificial Intelligence*, pages 573–80, 1985.

[30] T. M. Mitchell, L. I. Steinberg, S. Kedar-Cabelli, V. E. Kelly, J. Shulman, and T. Weinrich. An Intelligent Aid for Circuit Redesign. In *Proceedings of the National Conference on Artificial Intelligence*, August 1983.

[31] S. Mittal, C. M. Dym, and M. Morjaria. PRIDE: An Expert System for the Design of Paper Handling Systems. *Computer*, July 1986.

[32] J. Moses. Algebraic Simplification: A Guide for the Perplexed. *Communications of the ACM*, 14(8), Aug. 1971.

[33] S. S. Murthy and S. Addanki. PROMPT: An Innovative Design Tool. In *Proceedings of the 6th National Conference on Artificial Intelligence*, pages 637–42, July 1987.

[34] P. F. Patel-Schneider. Small can be Beautiful in Knowledge Representation. In *Proc. of the IEEE Workshop on Principles of Knowledge-Based Systems*, pages 11–16, Denver, CO, 1984.

[35] O. Raiman. Order of Magnitude Reasoning. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, August 1986.

[36] A. Ressler. *A Circuit Grammar for Operational Amplifier Design.* AI-TR-807, MIT AI LAB, January 1984.

[37] C. Rich. Knowledge Representation Languages and Predicate Calculus: How to Have Your Cake and Eat it Too. In *Proceedings of the National Conference on Artificial Intelligence*, pages 193–196, 1982.

[38] R. C. Rosenberg and D. C. Karnopp. *Introduction to Physical System Dynamics.* McGraw-Hill Book Co., New York, 1983.

[39] G. Roylance. *A Simple Model of Circuit Design.* AI-TR-703, MIT AI LAB, May 1980.

[40] E. Sacerdoti. The Nonlinear Nature of Plans. In *Proceedings of the Fourth IJCAI*, pages 206–214, 1975.

[41] J. L. Shearer, A. T. Murphy, and H. H. Richardson. *Introduction to System Dynamics.* Addison-Wesley, Reading, Mass., 1971.

[42] H. Shrobe. The Data Path Generator. In *Proceedings of the Conference on Advanced Research in VLSI*, January 1982.

[43] R. Simmons. *Combining Associational and Causal Reasoning to Solve Interpretation and Planning Problems.* AI-TR-1048, MIT AI Lab, September 1988.

[44] J. Siskind, J. Southard, and K. Crouch. Generating Custom High Performance VLSI Designs from Succinct Algorithmic Descriptions. In *Proceedings of the Conference on Advanced Research in VLSI*, January 1982.

[45] R. M. Stallman and G. J. Sussman. Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-aided Circuit Analysis. *Artificial Intelligence*, 9, 1977.

[46] G. L. Steele Jr. *Common LISP: The Language.* Digital Press, 1984.

[47] L. I. Steinberg. Design as Refinement Plus Constraint Propagation: The VEXED Experience. In *Proceedings of the 6th National Conference on Artificial Intelligence*, pages 830–5, July 1987.

[48] M. E. Stickel. A Nonclausal Connection-Graph Resolution Theorem-Proving Program. In *Proceedings of the National Conference on Artificial Intelligence*, August 1982.

[49] M. E. Stickel. Theory Resolution: Building-In Nonequational Theories. In *Proceedings of the National Conference on Artificial Intelligence*, August 1983.

[50] G. Strang. *Linear Algebra and Its Applications.* Academic Press, New York, 1980.

[51] P. Struss. Mathematical Aspects of Qualitative Reasoning. *Artificial Intelligence in Engineering*, To Appear 1988.

[52] G. Sussman. *A Computer Model of Skill Acquisition.* Elsevier, Inc., New York, NY, 1975.

[53] K. T. Ulrich. *Computation and Pre-Parametric Design.* AI-TR-1043, MIT AI Lab, Sept. 1988.

[54] K. T. Ulrich and W. P. Seering. Function Sharing in Mechanical Design. In *Proceedings of the 7th National Conference on Artificial Intelligence*, pages 342–6, August 1988.

[55] M. Vilain. The Restricted Language Architecture of a Hybrid Representation System. In *Proc. of the Ninth International Joint Conference on Artificial Intelligence*, pages 547–51, 1985.

[56] A. Weinberger. Large Scale Integration of MOS Complex Logic: A Layout Method. *IEEE Journal of Solid State Circuits*, SC-2:182–190, 1967.

[57] D. Weld. Comparative Analysis. *Artificial Intelligence*, 36(3):333–74, October 1988.

[58] D. Weld. Exaggeration. In *Proceedings of the 7th National Conference on Artificial Intelligence*, pages 291–5, August 1988.

[59] B. C. Williams. Doing Time: Putting Qualitative Reasoning on Firmer Ground. In *Proceedings of the National Conference on Artificial Intelligence*, August 1986.

[60] B. C. Williams. Exorcising Control of Demon Invocation: A Study in Tractable Non-Monotonicity. unpublished.

[61] B. C. Williams. *Invention from First Principles via Topologies of Interaction.* AI-TR-1127, MIT AI Lab, to appear 1989.

[62] B. C. Williams. MINIMA: A Symbolic Approach to Qualitative Reasoning. In *Proceedings of the National Conference on Artificial Intelligence*, August 1988.

[63] B. C. Williams. Qualitative Analysis of MOS Circuits. *Artificial Intelligence*, 24(1-3):281–346, January 1984.

[64] B. C. Williams. Temporal Qualitative Analysis: Explaining How Physical Systems Work. In *Artificial Intelligence Approaches to Engineering Design*, Addison-Wesley, to appear 1989.

[65] B. C. Williams. The Use of Continuity in a Qualitative Physics. In *Proceedings of the National Conference on Artificial Intelligence*, August 1984.

[66] W. Woods. What's in a Link: Foundations for Semantic Networks. In *Representation and Understanding*, pages 35–82, Academic Press, 1975.