

NORTHWESTERN UNIVERSITY

An Integrated Architecture
for Engineering Problem Solving

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

By

Yusuf Pisan

EVANSTON, ILLINOIS

December 1998

© Copyright by Yusuf Pisan 1998

All Rights Reserved

ABSTRACT

An Integrated Architecture for Engineering Problem Solving

Yusuf Pisan

Problem solving is an essential function of human cognition. To build intelligent systems that are capable of assisting engineers and tutoring students, we need to develop an information processing model that captures the skills used in engineering problem solving. This thesis describes the *Integrated Problem Solving Architecture (IPSA)* that combines qualitative, quantitative and diagrammatic reasoning skills to produce annotated solutions to engineering problems. We focus on representing expert knowledge, and examine how control knowledge provides the structure for using domain knowledge. To demonstrate our architecture for engineering problem solving, we present a *Thermodynamics Problem Solver (TPS)* that uses the IPSA architecture. TPS solves over 150 thermodynamics problems taken from the first four chapters of a common thermodynamics textbook and produces expert-like solutions.

ACKNOWLEDGMENTS

I would like to thank:

- My advisor, Ken Forbus, for providing guidance and encouragement.
- The members of my committee, Ian Horswill and Larry Birnbaum for their support and feedback.
- The Artificial Intelligence Division of the Office of Naval Research for providing the necessary funding for this research.
- Past and present members of the Qualitative Reasoning Group and especially Ron Ferguson and John Everett for discussions and support.
- All of my friends during this time who provided the necessary emotional support and put up with my complaints.
- My parents, Tamara and Habib Pisan, for making it all possible.
- And finally, Meryl McQueen, my wife, for encouraging me, for putting up with six years of grad school, for accompanying me on adventures, for reminding me that there is more to life than the thesis and for helping me finish in more ways than I ever imagined.

Contents

1.	Introduction.....	1
1.1	Engineering Problem Solving	2
1.2	Motivations	4
1.3	Expert Knowledge	5
1.4	Required Knowledge and Skills.....	6
1.5	Why Thermodynamics?.....	10
1.6	Contributions	11
1.7	Overview	14
2.	Overview of IPSA	16
2.1	IPSA Architecture	17
2.2	An Example	20
3.	Representing Physical Engineering Knowledge	28
3.1	Representing Domain Knowledge	28
3.2	Modeling Assumptions.....	38
3.3	Defining Units and Dimensions	40
3.4	TPS' Model of Common Thermodynamics Objects	41
3.5	Taxonomy of Thermodynamics Problems.....	49
3.6	Chapter Summary	51
4.	Qualitative Reasoning in Problem Solving.....	52
4.1	Defining Qualitative States.....	54
4.2	Finding Constant Parameters.....	59
4.3	Discovering Hidden States	62
4.4	Chapter Summary	67
5.	Plans: Frameworks for Solutions.....	69
5.1	Representing Plans.....	71
5.2	What should be represented in plans?	75
5.3	Common Problem Solving Primitives	75
5.4	Domain Abstractions.....	79
5.5	Choosing and Executing Plans	81
5.6	Chapter Summary	83
6.	Algebraic Knowledge.....	85
6.1	Functional Roles of Equations.....	86
6.2	Directional Preferences	92
6.3	Symmetrical Substitution	93
6.4	Opportunistic Strategies.....	95

6.5	Chapter Summary	98
7.	Diagrams in Problem Solving	99
7.1	A Model of Graph Understanding	100
7.2	Domain Translator	102
7.3	Visual Routines.....	103
7.4	Graph Summary.....	105
7.5	Comparative Diagrammatic Analysis	108
7.6	Diagrams as Input.....	110
7.7	Diagrams as Output	112
7.8	Diagrams for Result Verification	114
7.9	Chapter Summary	115
8.	Evaluation	116
8.2	Comparing to Expert Solutions	122
8.3	Chapter Summary	129
9.	Related Work	130
9.1	Engineering Problem Solving and Planning.....	130
9.2	Qualitative Reasoning	134
9.3	Algebraic Reasoning	136
9.4	Diagrammatic Reasoning	138
10.	Conclusion.....	142
10.1	Contributions to Qualitative Reasoning	143
10.2	Contributions to Planning.....	143
10.3	Contributions to Algebraic Reasoning	144
10.4	Contributions to Diagrammatic Reasoning.....	145
10.5	Limitations.....	145
10.6	Future Work & Possible Extensions	147
10.7	Summary	149
	References.....	150

List of Figures

Figure 1: Architecture for IPSA.....	19
Figure 2: Problem description	22
Figure 3: TPS' answer for problem 4.03a	25
Figure 4: Syntax of defModelFragment form	29
Figure 5: Example of a defModelFragment form.....	31
Figure 6: Types of consequence statements in defModelFragment.....	31
Figure 7: Syntax of defEntity form.....	32
Figure 8: Defining the concept of a sphere	33
Figure 9: Process and action	34
Figure 10: Saturated-vapor relation	35
Figure 11: Equation for density definition	35
Figure 12: Conservation of Energy equation	37
Figure 13: Representing when saturated table lookup can be performed.....	38
Figure 14: Defining phases of a thermodynamic-stuff.....	39
Figure 15: Syntax of defAssumptionClass form.....	40
Figure 16: Syntax of defdimension and defunit forms	40
Figure 17: Examples of dimension and unit definition.....	41
Figure 18: Schematic representations for heater cooler and heat-exchanger.....	43
Figure 19: Schematic representation for Splitter and Mixer.....	43
Figure 20: Schematic representation for turbine and compressor.....	44

Figure 21: Schematic representation for a throttle.....	45
Figure 22: Schematic representation for a pump	45
Figure 23: Schematic representation for a block.....	46
Figure 24: Schematic representation of connected containers	46
Figure 25: Schematic representation for container types.....	47
Figure 26: Schematic representation for a spring.....	48
Figure 27: Distribution of devices	49
Figure 28: Mapping from problem specification to qualitative states	56
Figure 29: Operator definition	58
Figure 30: Combining ideal gas equation.....	61
Figure 31: First law of thermodynamics	62
Figure 32: Simplified version of first law typically used for turbines	62
Figure 33: Constrained attainable envisionment heuristic.....	63
Figure 34: Problem 4.21	65
Figure 35: Work equations for problem 4.21	66
Figure 36: States generated by qualitative analysis	67
Figure 37: Relating intensive and extensive properties.....	80
Figure 38: Additional internal equations.....	81
Figure 39: Equations with intensive properties.....	81
Figure 40: Problem analysis technique	69
Figure 41: defplan syntax.....	72

Figure 42: Examples of defplan forms.....	73
Figure 43: Basic suggestions.....	74
Figure 44: Plan introducing multiple methods	83
Figure 45: Equations that can be used for calculating dryness	88
Figure 46: Bridge equations.....	89
Figure 47: First law of thermodynamics	90
Figure 48: Two forms of the ideal gas law	92
Figure 49: An equation with preferences for direction of solving.....	93
Figure 50: Use of symmetrical substitution	95
Figure 51: Equations that should not be combined	97
Figure 52: Architecture for graph understanding.....	101
Figure 53: Compression of carbon dioxide.....	107
Figure 54: TPS' description of carbon dioxide compression graph	107
Figure 55: TPS' explanation	109
Figure 56: Effect of superheating on Rankine cycle.....	109
Figure 57: Problem description with graph as input.....	110
Figure 58: TPS' representation for problem 4.73	111
Figure 59: Statements added through graph analysis	111
Figure 60: TPS' solution for problem 4.73.....	111
Figure 61: Problem requiring graph output	112
Figure 62: TPS' description for problem 4.03	113

Figure 63: Graph produced by TPS for problem 4.03.....	113
Figure 64: Problem where graph is used for confirmation.....	114
Figure 65: TPS' description for problem 3.09	114
Figure 66: Graph produced by TPS as part of explanation	115
Figure 67: Distribution of problems by chapter	117
Figure 68: Excluded Problems	120
Figure 69: Problems attempted and solved by TPS.....	120
Figure 70: Natural language and TPS' representation of a problem	123
Figure 71: Expert Solution to part a	124
Figure 72: TPS' solution to part a.....	124
Figure 73: Expert solution to part b	125
Figure 74: TPS' solution to part b	126
Figure 75: TPS steps versus expert steps	128
Figure 76: Work equations	129

1. Introduction

Problem solving is an essential function of human cognition. From early humans building fires to stay warm to engineers designing deep space probes, problem solving plays a central role in everyday life. To build intelligent systems that can effectively solve problems and communicate their reasoning, we need to understand how people solve problems and what skills are used in problem solving.

Understanding how people solve problems has been one of the long-term goals of Cognitive Science research. Although it is possible to build intelligent systems without understanding how people achieve the same task (such as chess programs that conduct extensive search), these systems cannot explain their reasoning in terms that are accessible to people. To build intelligent teaching systems, we need to capture and organize how experts in the field conceptualize that task and explain their solutions to the learner.

Domain knowledge lies at the center of problem solving activities. We cannot expect a layman to build rockets or represent a client in a lawsuit without the necessary domain knowledge, which is usually acquired through years of formal education. However, domain knowledge is not sufficient in itself, because there are often many different ways of solving a problem. Lacking infinite resources, we need control knowledge to guide us through the steps of problem solving. We refer to the combination of domain knowledge and control knowledge as *expert knowledge*.

Engineering provides a suitable field to study the interactions between domain knowledge and control knowledge. The problem solving centered approach to teaching engineering has created standard classes of problems with well-defined solutions. These solutions reflect the ideal control knowledge to be transferred to the learner. Unlike domain knowledge, which can be explained more easily, when and how control knowledge should be applied is not well-formulated. Learners glean and fine-tune their control knowledge through solving problems and studying solved examples.

The goal of this work is to develop an information processing model for solving engineering problems. We describe an *Integrated Problem Solving Architecture (IPSA)* that combines qualitative, quantitative and diagrammatic reasoning skills to produce annotated solutions to engineering problems. We focus on representing expert knowledge, and examine how control knowledge provides the structure for using domain knowledge. To demonstrate our architecture for engineering problem solving, we have applied it to thermodynamics and built *Thermodynamics Problem Solver (TPS)* using the IPSA architecture.

1.1 Engineering Problem Solving

Many efforts in problem solving has been directed at knowledge-poor domains, exemplified by the Blocks World. In knowledge-poor domains, the rules are simple, strategies are limited and there is usually exactly one right answer. Strategies that have worked successfully in these domains have failed to scale up to knowledge-rich

domains. The uniformity found in knowledge-poor domains has made it possible to experiment with a number of learning algorithms to develop a good set of strategies. Unfortunately, these algorithms have been shown to be overly dependent on representational choices, unable to characterize classes of problems, and over-specialized on the initial training problem set (DeJong & Mooney, 1986; Tadepalli & Natarajan, 1996). For knowledge-rich domains, problem solving strategies are more complex. Explicit representations of problem solving strategies and domain abstractions play a more important role for knowledge-rich domains.

Engineering problem solving is a knowledge-rich domain where effective control knowledge is needed to create efficient solutions. Compared to knowledge-poor domains, in engineering problem solving the problem space is larger, making the problem solver more dependent on control knowledge. Furthermore, knowledge-rich domains require analysis at multiple levels. For example, deciding what assumptions can be made about a situation is a qualitatively different type of decision than choosing between two equations that are applicable to the current situation.

Textbook problems represent a partial model of how engineers solve problems. For this work, we ignore the social, collaborative and creative aspects of problem solving and focus on traditional problem solving. Strategies used in idealized textbook problems represent the basis that engineers can fall back on when analyzing complex problems that require new innovations. To provide this necessary basis, engineering

education has typically focused on textbook problem solving. In this thesis we examine the necessary knowledge and strategies that are required to be competent in solving textbook problems. This work can be seen as a step towards building a virtual engineer.

1.2 Motivations

We would like to build intelligent tutoring systems that augment classroom education. An intelligent tutoring system must be flexible enough to solve a variety of problems and effectively communicate its solutions to learners. Our architecture provides the foundation for building a teaching system by capturing expert knowledge and making it explicit. Our emphasis in designing the IPSA architecture was on integrating qualitative, quantitative and diagrammatic skills to produce expert-like solutions. A teaching system built on top of IPSA will have to consider the pedagogical issues.

A secondary goal of this work is to understand the nature of human cognition with respect to problem solving by developing an information processing model. An information processing model provides one possible explanation about how people could be solving problems. Once the informational processing requirements for solving a problem has been established further psychological tests can be used to tease out how people actually solve problems. We have used current models of problem solving and experts' intuition about how a problem is solved as the basis for our architecture. The set of skills we identify and implement are essential to produce expert-like solutions

and thus these skills have to be a part of all theories that attempt to explain expert problem solving behavior.

Finally, our research provides the foundation for more powerful and realistic engineering software to assist engineers with design and analysis tasks.

1.3 Expert Knowledge

One of the differences between an expert and a novice is the expert's ability to eliminate paths that are unlikely to lead to a solution. While a novice flounders in the problem space, an expert's control knowledge allows them to produce an elegant solution. Expressing when a piece of domain knowledge such as an equation is potentially relevant is easily expressed. Control knowledge indicating whether a particular equation or a method is the best way to achieve the current goal is harder to articulate. Although most good textbooks provide rudimentary plans describing how to analyze problems, these plans are often too abstract to be directly applied. Furthermore, plans specific to the domain are often combined with general engineering problem solving strategies, which make it more difficult for students to transfer these skills to other engineering domains. The IPSA architecture, which separates domain knowledge from control knowledge, allows us to identify and highlight often elusive control decisions. Because each piece of control knowledge is represented separately in our architecture, identifying control knowledge that is applicable to all engineering problem solving is easier.

We have represented the necessary domain knowledge and control knowledge for introductory thermodynamics and built a program, the *Thermodynamics Problem Solver*, to explore the IPSA architecture. Comparisons between TPS' performance on multiple problem classes with experts indicate that IPSA architecture has the necessary skills and provides a sufficiently rich vocabulary to encode domain knowledge and control knowledge. The distribution of skills used in solving problems we provide in Chapter 8 provide further evidence that all the skills implemented in IPSA are necessary for engineering problem solving.

The performance of a problem solver ultimately depends on the completeness of the domain knowledge and the extent to which control knowledge takes advantage of the skills in the underlying architecture. In the worst case scenario, where no control knowledge is available, the problem solver would be performing blind search to find the answer. With perfect control knowledge, problems would be solved without considering any alternatives, which is an idealization not achieved even by experts.

1.4 Required Knowledge and Skills

Thermodynamics is the study of heat and work, and the conversion of heat into work and vice versa. Power plants (solar, fuel, geothermal, etc.); engines (steam, gas, rockets, etc.); air-conditioning and heating systems are examples of some of the devices that rely on thermodynamics. The cooling cup of hot coffee, the egg that explodes when microwaved and the bicycle pump that heats up while inflating a tire are examples

of thermodynamics processes and effects that are encountered in daily life.

All thermodynamic processes can be broken down into combinations of heating, cooling, expansion, compression, and mass-flow. In most systems, multiple processes operate simultaneously. For example, heating the gas inside a piston increases the pressure of the gas, which in turn forces the piston up, expanding the volume of the container. This system combines both heating and expansion processes.

Of course, an understanding of the nature of thermodynamics is crucial to building competency in thermodynamics problem solving. However, domain knowledge is only part of the information needed to successfully solve thermodynamics problems. There are four areas of expertise that, combined with basic domain knowledge, form the basis for problem solving:

- Qualitative reasoning
- Plan formulation
- Algebraic reasoning
- Diagrammatic reasoning

These four modules provide the framework for understanding how experts combine skills and knowledge to solve problems. We describe these skills in the following sections.

1.4.1 Qualitative Reasoning

Engineering is stereotypically characterized by endless series of equations; however, an essential element of engineering education is the ability to discern when and how equations should be applied. Qualitative reasoning provides the framework for representing the modeling assumptions under which equations and other domain inferences hold true.

All engineering domains involve some level of reasoning about modeling assumptions. Even for domains like physics where a default set of modeling assumptions (such as point objects and frictionless surfaces) can be used extensively, extending the domain model requires making these assumptions explicit. Thermodynamics, with interconnected components and no default assumptions, is an ideal domain for studying and understanding how to reason about and manipulate modeling assumptions.

Most problems require analysis of a system undergoing change. By expressing the causal relations among parameters, qualitative reasoning makes it possible to deduce whether and in which direction the values of parameters are changing. This makes it possible to address the frame problem as originally discussed by (McCarthy & Hayes, 1969) and permits the analysis of multi-state problems.

1.4.2 Plans

Plans represent typical ways of using domain knowledge. As part of problem solving, plans are the well-established procedures that an expert follows. Usually, plans are a sequence of actions that have been learned and have been proven to work well.

One of the advantages of plans is that they reduce the size of the problem space. In addition to conserving limited resources, plans enable the expert to make predictions of the outcome of a specific action. Each step in a plan becomes an immediate subgoal, which may lead to further exploration of a particular approach to solving the problem.

1.4.3 Algebraic Reasoning

Engineering laws are frequently expressed in terms of equations. Unlike novices, who treat equations as collections of parameters, experts attach functional roles to equations using domain knowledge. This enables experts to divide equations into classes and build further abstractions using these classes. This taxonomy of equations promotes efficient problem solving. For example, avoiding the numerical computation of parameters that may be cancelled out while working through equations is typical of how people solve problems. This “lazy” approach conserves resources by minimizing cumbersome calculations. TPS mimics this expert behavior.

1.4.4 Diagrammatic Reasoning

Teachers and textbooks often use diagrams to explain problems or to illustrate theories.

People's ability for efficient visual processing makes diagrams an ideal cognitive tool. In problem solving, diagrammatic reasoning serves two purposes. Diagrams are used as part of both the problem description (to specify spatial configurations and numerical relationships), and the problem solution (to verify numerical results). We demonstrate how TPS incorporates diagrammatic reasoning into its integrated problem solving approach.

1.5 Why Thermodynamics?

Thermodynamics is the study of how energy can be transferred from one form to another. Engineering thermodynamics provides the theoretical underpinnings for power plants, engines of all types, refrigerators, and heat pumps. What makes thermodynamics difficult for students is not the complexity of equations, which are no more complex than those found in, say, dynamics. The challenge lies in the fact that the properties being modeled are subtle, meaning that there are both more equations and that the range of their applicability is more limited.

Engineering thermodynamics requires explicit reasoning about modeling assumptions in order to derive a consistent set of equations. The domain knowledge and the control knowledge necessary for engineering problem solving are important parts of thermodynamics experts' professional knowledge. In short, introductory thermodynamics involves basic engineering skills common to all engineering domains.

Like most engineering domains, the core of formal thermodynamics education is problem solving. Students build the necessary intuition and control knowledge through practice with multiple classes of problems. Discussions with domain experts (P. B. Whalley, personal communication) suggest that textbook problems are reasonable approximations for engineering analyses performed in industry.

Although textbook problems tend to be closely bounded, the idealizations evoked about the system are the same for textbook and real world problems. The kinds of performance bounds thermodynamics problems provide are critical in evaluating designs and suggesting improvements to existing systems. In fact, the well defined solutions for textbook problems provide the baseline for evaluating our problem solver's performance.

1.6 Contributions

The work presented here is a general architecture for engineering problem solving. This is the first architecture we are aware of that combines qualitative, diagrammatic and algebraic reasoning with plans. Each of these fields has been explored with respect to how they can be used for problem solving and we have borrowed from previous work as much as possible. What makes the IPSA architecture unique is the combination of multiple reasoning techniques in a unified framework to solve large sets of problems. We have successfully applied the IPSA architecture to thermodynamics to build a thermodynamics problem solver. Since thermodynamics is considered one of

the most difficult engineering fields, we believe IPSA can be readily applied to other domains with similar success.

IPSA uses an extensive modeling and control language to produce expert-like solutions, which makes it possible to generate explanations describing each step of the solution. This detailed description of the reasoning that resulted in the solution is essential if we care for more than just the final answer. In the case of intelligent tutoring systems, the process of arriving at the answer is a valuable part of the solution. The modeling and control language provide coherent set of annotated solution steps similar to experts' problem solving.

This work contributes to the understanding and design of engineering problem solvers and links problem solving to other areas of research in the following ways:

- For planning, a planning vocabulary that enables us to model expert strategies
- For algebraic knowledge, abstractions and equation classifications applicable to all engineering domains
- For diagrammatic reasoning, a computational model for interpreting graphs and using the results in the context of problem solving
- For qualitative reasoning, using qualitative simulation to derive hidden states and identify relevant assumptions

- To the field of engineering, formalization of necessary skills and demonstrating through application that this formalization is sufficient for solving large problem sets, so these skills can be more explicitly taught and studied

To substantiate our claims about the IPSA architecture, we have implemented it and applied it to thermodynamics in our Thermodynamics Problem Solver (TPS). TPS makes it possible to test the limits and demonstrate the effectiveness of the architecture. The effectiveness of IPSA architecture can be measured on two dimensions: coverage, and similarity to expert solutions.

- Coverage

TPS achieves broad coverage of its domain. TPS has solved 75% of the problems in the first four chapters of (Howell & Buckius, 1992), as well as problems from (Moran & Shapiro, 1995; Whalley, 1992; Wylen & Sonntag, 1985). TPS' success in solving over 150 problems from multiple textbooks demonstrates its competency in the field of thermodynamics.

- Similarity to expert solutions

TPS generates annotated solutions to problems. When comparing TPS' annotated solutions to expert solutions, we ignore syntactic differences (such as skipping steps, combining multiple steps into one, simplifying equations or

solving one subgoal before another) and focus on the main equations applied in the solution.

TPS' problem solving method is a plausible model of how experts solve problems. Traditional computer problem solving relies on processing speed to reach the answer. Modeling human behavior, TPS trades knowledge for fast search, delays numerical calculations, imposes resource limits, and uses domain abstractions to produce expert-like solutions. We use empirical data to support our claim that these skills as described in IPSA and implemented in TPS form the necessary and sufficient set of skills for the task of engineering problem solving.

1.7 Overview

The structure of the thesis is as follows:

- Chapter 2 presents an overview of IPSA.
- Chapter 3 discusses how the system represents engineering knowledge.
- Chapter 4 demonstrates the need for qualitative reasoning in problem solving and shows how qualitative reasoning is used in TPS.
- Chapter 5 describes the plan vocabulary and outlines plans that form the problem solving framework.
- Chapter 6 describes the representation of algebraic knowledge and the classification of equations.

- Chapter 7 demonstrates diagrammatic reasoning and describes how it is integrated into the IPSA architecture.
- Chapter 8 provides an evaluation of TPS.
- Chapter 9 discusses related work in problem solving, planning, qualitative reasoning and diagrammatic reasoning.
- Finally, Chapter 10 concludes with a summary of key points and directions for future work.

2. Overview of IPSA

In order to be successful, a problem solving architecture must satisfy several criteria. It must

- Achieve broad coverage of problems in a knowledge-rich domain.
- Support general engineering skills in order to be applicable to a variety of engineering domains
- Support an extendable and declarative language so that the domain knowledge can be built by experts
- Generate solution paths that are similar to expert solutions.

The TPS system that implements the IPSA architecture fulfills the above criteria.

TPS solves over 150 problems from multiple classes (the different classes of problems are described in Section 3.5) taken from several different textbooks (Howell & Buckius, 1992; Moran & Shapiro, 1995; Wylen & Sonntag, 1985). IPSA integrates a set of four common engineering skills that can be applied to domains other than thermodynamics: qualitative reasoning, plans, algebraic reasoning, and diagrammatic reasoning. One of the main advantages of IPSA is that it allows the knowledge engineer to express control knowledge and domain knowledge in a uniform manner.

TPS' solutions are also qualitatively similar to expert solutions, leading us to believe that it will make an excellent foundation for an intelligent tutoring system.

2.1 IPSA Architecture

As seen in Figure 1, the main modules of IPSA are the Plan Executor, the Qualitative Reasoner, the Algebraic Reasoner, and the Graph Analyzer. The expert knowledge contains both domain knowledge and control knowledge necessary for problem solving. The Logic Based Truth Maintenance System serves as a blackboard for storing all information and inferences about how things are derived. The controller chooses among suggestions provided by these modules and takes the necessary action, such as making an assumption or choosing an equation. The output of the system is the set of steps taken to solve the problem.

A minimal problem description consists of one or more entities, and a goal. In addition, the problem description may include numerical values, modeling assumptions, relations between objects, processes or actions that act on entities, supplementary problem-specific equations, or graphs and tables. Although most problems have calculating a numerical value as their goal, other goals are to prove a modeling assumption, derive an inequality or draw a graph. TPS can solve numerical problems, problems that ask for an assumption or an inequality to be derived, and problems that ask for a graph to be drawn.

Complex problems often involve multiple interacting objects. Processes and actions given in the problem statement provide the necessary chronological information. To distinguish between the value of a parameter at the beginning and the end of the problem, we need to represent time explicitly. We use the representation of time introduced by Hayes (Hayes, 1990). A *history* is defined as spatially restricted and temporally extended piece of space-time. A *state* is defined as an instantaneous "slice" of a history at a certain time-instant. For most problem statements, the history is composed of two distinguished states called begin and end. The set of statements that is true at a given time is referred to as the *partial problem specification*.

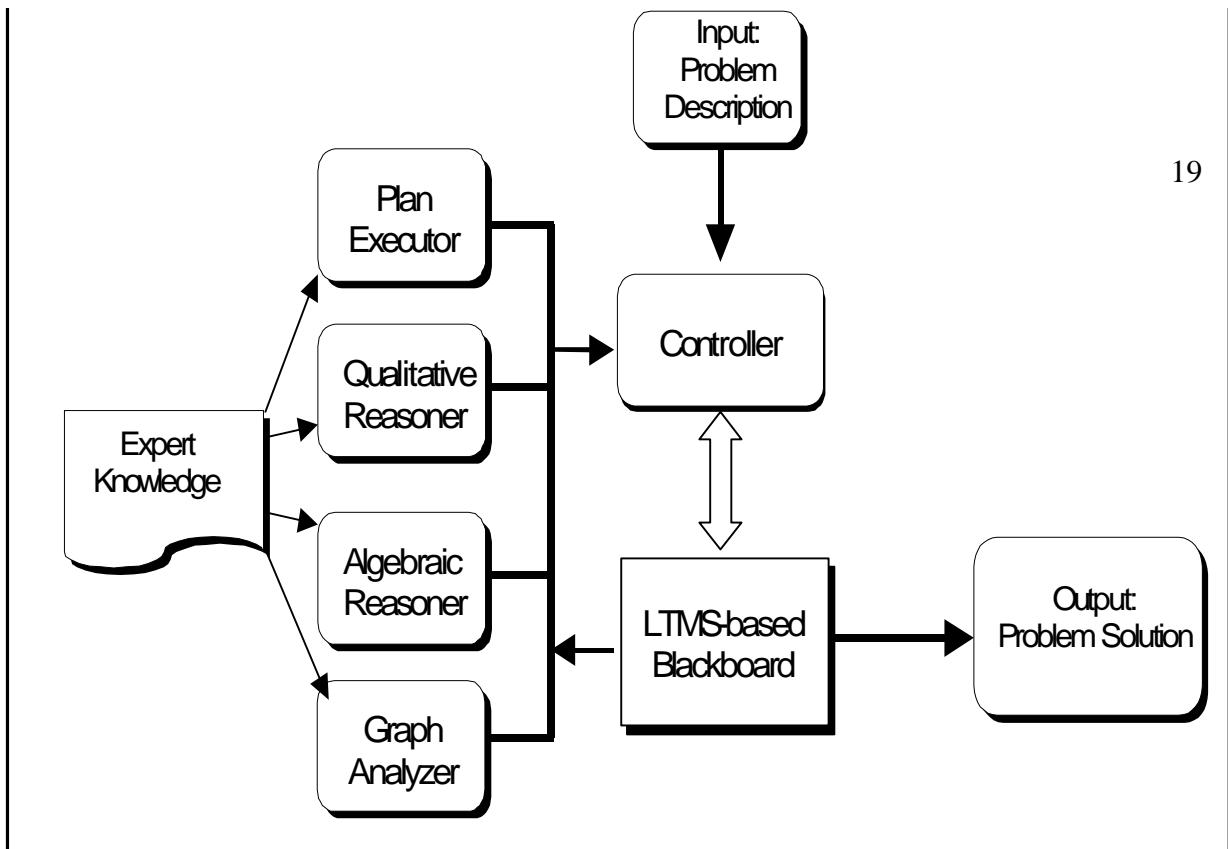


Figure 1: Architecture for IPSA

Once a problem is entered into the system, different reasoning modules can make suggestions to achieve the current goal. Qualitative reasoning suggestions provide information about which parameters remain constant throughout a process and identify the set of modeling assumptions that may apply to the current problem. Plans represent the expert knowledge about how to achieve a goal or what actions to take in a partially-defined context. In IPSA, plans organize other reasoning modules and provide the basic framework for the solution. Plan steps can explicitly use other modules to achieve goals. Diagrammatic reasoning suggestions convert graph information into numerical values and inequalities that can be used by the problem solver to derive parameters, check results, or verify modeling assumptions. Algebraic reasoning suggestions offer equations that can be used to solve for variables, table

lookups, and ordinal information. Most engineering problems are calculation-intensive; therefore, algebraic knowledge is critical to engineering problem solving.

Throughout problem solving, TPS records all its actions in a *solution trace*. The solution extracted from the solution trace is much more detailed than expert solutions, because it includes trivial steps that experts do not bother to record. Output filters eliminate superfluous information to create a more concise, coherent solution.

TPS uses a logic-based truth maintenance system (LTMS) that serves as a blackboard for storing domain knowledge, the solution trace, and the problem description. The domain knowledge base is compiled into pattern-directed rules. The initial domain also includes background knowledge in the form of default assumptions.

2.2 An Example

In the next sections, we give a detailed example of how a simple thermodynamics problem is solved to clarify how the modules of IPSA as implemented in TPS interact with one another. The set of problems TPS solves is included in Appendix A

2.2.1 Input

The input is a problem statement along with any accompanying tables and graphs that are needed to solve the problem. We have remained faithful to the information provided in textbook problems in translating the natural language problem statement into predicate calculus statements.

The *add-problem* procedure is used to specify the problem statement including the goal of the problem. The problem statement presents a specific *scenario* consisting of entities, modeling assumptions and numerical relations. For problems with multiple goals we use *add-problem-description* to represent the problem statement and represent each subpart using *add-problem-goal* procedure. All these forms take the name of the problem as the first argument. The *:statement* is used to provide the natural language text provided in the original text. The *:givens* clause describes the initial scenario including any numerical values and additional equations. The *:goal* clause, which can be used in add-problem and add-problem-goal functions, expresses the objective for the problem. The answer can optionally be provided using the *answer* keyword. This allows the problem solver to compare its result with the given result and determine if the problem was solved successfully. A representative problem description taken from (Howell & Buckius, 1992) is shown in Figure 2.

```

(add-problem-description :hb4.03
  :statement "A piston-cylinder arrangement initially contains water at  $P = 2 \text{ MPa}$  and  $T = 300^\circ\text{C}$ . The piston is moved to a final position where the volume is twice the initial volume. During the piston movement, there is heat transfer to the water in such a way that the pressure in the cylinder remains constant."
  :givens '((piston (at can :begin))
    (static-thermodynamic-stuff (at S :begin))
    (inside (at S :begin) (at can :begin))
    (direction (at can :begin) :up)
    (substance-of (at s :begin) water)
    (nvalue (P (at s :begin)) 2 MPa)
    (nvalue (T (at s :begin)) 300 C)

    (expanding expn (at s :begin) (at s :end))
    (isobaric expn)
    ;; volume is doubled
    (given-equation volume-doubled
      (volume-doubled
        (:= (V (at s :end)) (* 2 (V (at s :begin)))))))

(add-problem-goal :hb4.03a
  :statement "Sketch the process on a  $P$ - $v$  diagram, indicating the relative position of the saturation curve."
  :description :hb4.03
  :goal '(plot-graph (P spec-v) expn))

(add-problem-goal :hb4.03b
  :statement "Find the work done per kilogram of water during the expansion"
  :description :hb4.03
  :goal '(find (nvalue (spec-work expn)))
  :answer '(250.8 kj/kg))

(add-problem-goal :hb4.03c
  :statement "Find the heat transfer per kilogram of water during the expansion."
  :description :hb4.03
  :goal '(find (nvalue (spec-q expn)))
  :answer '(1169.9 kj/kg))

(add-problem-goal :hb4.03d
  :statement "Find the value of the enthalpy of the water after expansion"
  :description :hb4.03
  :goal '(find (nvalue (spec-h (at S :end))))
  :answer '(4192.6 kj/kg))

```

Figure 2: Problem description

Because engineering thermodynamics is highly specialized, converting the natural language descriptions into a set of predicate calculus statements is a straightforward

task for a domain expert. Although textbook authors differ on what is the appropriate language for problems, students can often get to the underlying thermodynamics relations even when they do not understand the scenarios described. For example, (Batali, 1991) presents a natural language system for decomposing physics problems without understanding the individual objects involved. We show the corresponding natural language text and predicate calculus statements below to demonstrate that representing a problem is a straightforward task for a domain expert and can further be automated if necessary.

A piston-cylinder arrangement

```
(piston (at can :begin))
(direction (at can :begin) :up)
```

initially contains water

```
(static-thermodynamic-stuff (at S :begin))
(inside (at S :begin) (at can :begin))
(substance-of (at s :begin) water)
```

at $P = 2 \text{ MPa}$ and $T = 300^\circ\text{C}$.

```
(nvalue (P (at s :begin)) 2 MPa)
(nvalue (T (at s :begin)) 300 C)
```

The piston is moved to a final position where the volume is twice the initial volume.

```
(expanding expn (at s :begin) (at s :end))
(given-equation volume-doubled
  (volume-doubled
    (:= (V (at s :end)) (* 2 (V (at s :begin)))))))
```

During the piston movement, there is heat transfer to the water in such a way that the pressure in the cylinder remains constant.

(isobaric expn)

2.2.2 Output

The output of the system is an annotated solution. We use the term annotated to indicate that the reasons for choosing an equation or making an assumption are made explicit in the answer. Annotated solutions are better than the solved examples in textbooks because they make the underlying assumptions explicit. A common problem students encounter is that the textbook examples skip steps and the student needs to draw out the underlying inferences. TPS allows the user to explicitly ask questions about each solution step, thereby providing a very detailed explanation when required. These detailed explanations, usually provided by classroom teachers and mimicked by TPS, can be critical elements of the learning process.

Problem 4.03, shown in Figure 2, has 4 different subparts. The first part requires drawing a graph, the rest require finding numerical values. The graph produced by TPS as answer to the first part is shown in Figure 3. It is important to note the textbook authors' intentions to order the problem subparts as they are. Drawing a graph of the process imposed on the saturation curve for water emphasizes three important properties. First, the thermodynamic stuff is not saturated for both initial and final phases. Second, the process is represented by a horizontal line since it is specified as isobaric in the problem statement. The final pressure can be obtained easily from the

initial conditions using this equality. Third, this is a constant mass system, that is, mass stays constant throughout this process. This information is required before the specific-volume at the final state can be calculated.

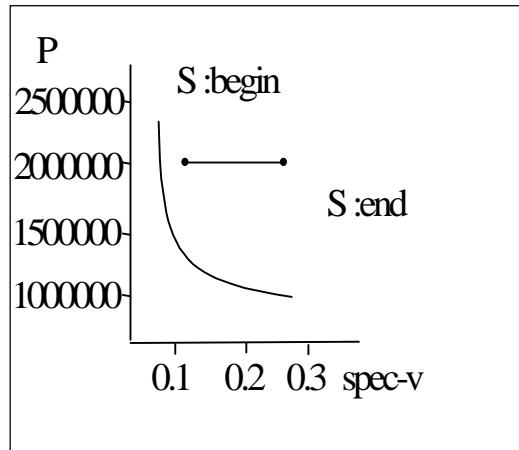


Figure 3: TPS' answer for problem 4.03a

Although problem 4.03 provides students with some guidance by partitioning the problem into different parts, this type of guidance is often not available. Plans allow this expertise, such as finding the state of the thermodynamic stuff as an initial step, to be captured. We show the TPS solution to part c of this problem below. In the solution for the second part of the problem shown below the values for temperature and pressure are used to determine that the initial phase is gas. The solution below is shown in full detail, including 5 equality calculations and 3 table lookups. When comparing TPS' solutions, we ignore syntactic differences such as skipping steps and combining multiple steps into one as these factors are bound to vary among experts and

there is no one optimal presentation. We focus on what equations are used in the solution.

```

Engineering Analysis:
SPEC-CONSERVATION-OF-ENERGY

Found (GAS (AT S :BEGIN))
using
  (TABLE-LOOKUP-USING (T (AT S :BEGIN)) 573.15
                        (P (AT S :BEGIN)) 2000000)

Found (SPEC-V (AT S :BEGIN)) = 0.12545
using
  SUPERHEATED-P-T

Found (SPEC-V (AT S :END)) = 0.2509
using equality
  (:= (SPEC-V (AT S :END)) (* 2 (SPEC-V (AT S :BEGIN)))))

Found (SPEC-WORK EXPN) = 250900.0
using
  SPEC-WORK=PDV-ISOBARIC

Found (P (AT S :END)) = 2000000
using equality
  (:= (P (AT S :BEGIN)) (P (AT S :END)))

Found (SPEC-U (AT S :END)) = 3691601.96749358
using
  SUPERHEATED-P-SPEC-V

Found (SPEC-ENERGY (AT S :END)) = 3691601.96749358
using equality
  (:= (SPEC-ENERGY (AT S :END)) (SPEC-U (AT S :END)))

Found (SPEC-ENERGY-END EXPN) = 3691601.96749358
using equality
  (:= (SPEC-ENERGY-END EXPN) (SPEC-ENERGY (AT S :END)))

Found (SPEC-U (AT S :BEGIN)) = 2771800.0
using
  SUPERHEATED-P-T

Found (SPEC-ENERGY (AT S :BEGIN)) = 2771800.0
using equality
  (:= (SPEC-ENERGY (AT S :BEGIN)) (SPEC-U (AT S :BEGIN)))

Found (SPEC-ENERGY-BEGIN EXPN) = 2771800.0
using equality
  (:= (SPEC-ENERGY-BEGIN EXPN) (SPEC-ENERGY (AT S :BEGIN)))

```

```
Found (SPEC-Q EXPN) = 1170701.96749358
using
  SPEC-CONSERVATION-OF-ENERGY
  ( := ( + (SPEC-Q EXPN)
           (SPEC-ENERGY-BEGIN EXPN)
           (SPEC-ENERGY-INLET EXPN))
        ( + (SPEC-ENERGY-END EXPN)
             (SPEC-ENERGY-OUTLET EXPN)
             (SPEC-WORK EXPN)) )
```

The spec-conservation-energy identified at the beginning of the problem is used as the final equation of the solution. Engineering analysis determines the *frame equation* that will be used for analyzing the system. Frame equation and other equation types are discussed in detail in Chapter 6.

3. Representing Physical Engineering Knowledge

Reasoning begins with representation. To understand how to solve engineering problems, we need to represent the physical knowledge of the domain. In the following section, we describe how domain knowledge is organized and represented. Section 3.2 describes how modelling assumptions are represented, Section 3.3 describes units and dimensions. Section 3.5 describes the taxonomy of thermodynamics problems and section 3.4 describes the models for some of the common objects found in thermodynamics.

3.1 Representing Domain Knowledge

We use the *compositional modeling methodology* (Falkenhainer & Forbus, 1989; Falkenhainer & Forbus, 1991) to organize the problem solving knowledge. The basic premise of compositional modeling is that the *domain theory* includes *model fragments* where each fragment describes a particular aspect of the domain. Our representation language is similar to the CML language described in (Falkenhainer et al., 1994).

```
(defModelFragment name
  [ :documentation <string> ]
  [ :participants ((participant1 :type participant-type1
    [ :constraints constraints ])...)]
  [ :conditions conditions ]
  [ :quantities ((quantity :arguments arguments
    :dimension quantity-dimension) ...)]
  [ :consequences consequences ] )
  [ :substitutions ((symbol1 form1) )]
```

Figure 4: Syntax of defModelFragment form

Each model fragment has a unique name that identifies it.

:documentation The documentation is a string describing the model fragment in natural language.

:participants Each participant in the participants is an object belonging to the class participant-type. A model fragment instance is *instantiated* when all the given participants of a model fragment are satisfied. We use the :constraints clause to specify the relations that must hold between the participants in order for instantiation to occur. This enables TPS to avoid instantiating a large number of model fragments that could never be active.

:conditions The conditions clause specifies the conditions when the model fragment instance becomes *active*. When a model fragment is active the quantities and the consequences defined in the model fragment are believed.

:quantities The quantities clause defines a set of quantities that exist when the model fragment is active. The quantity is defined using the *quantity* name with arguments specified by *arguments*. The *quantity-dimension* specifies the units. A quantity with the same name and arguments but with a different dimension can be specified by another model fragment as long as the model fragments are mutually exclusive. The arguments specified for the quantity is a list of terms usually including one or more participants.

:consequences The consequences clause is a set of statements that are believed when the model fragment is active. The consequences assert numerical relations, logical constraints and implications that define the behavior of its participants. At this point we diverge from the CML specification described in (Falkenhainer et al., 1994) and do not allow equations to be in the consequences of a model fragment. Equations are defined using special `defEquation` forms described below. By using `defEquation` forms we can name individual equations, define equation groups and use these classifications in performing algebraic reasoning. We will discuss how equation grouping facilitates reasoning in Chapter 6 in detail. We give examples of different kinds of consequence statements in Figure 6.

:substitutions Substitutions define a set of purely syntactic substitutions to facilitate building models. Each substitution consists of a *symbol* and an arbitrary *form*.

For all fields in the original definition of the model fragment (except the participants and the substitutions field) the *form* is substituted for the *symbol* wherever it appears.

Separating participants and conditions allows differentiation between model fragments that are not relevant to the current scenario and model fragments that are relevant, but are not currently active. When a scenario is under-specified, examining instantiated but not active model fragments is necessary to determine the default assumptions which can be used to complete the model.

```
(defModelFragment ke-thermodynamic-stuff-static
  :participants ((stuff :type ke-thermodynamic-stuff)
                 (stuff :type simple-mass-object))
  :quantities ((kinetic-energy :arguments (stuff)
                                :dimension energy-dimension)))

(defModelFragment closed-container-geometric
  :participants ((can :type container)
                 (stuff :type thermodynamic-stuff))
  :conditions ((closed can)
                (inside stuff can)))
  :consequences
  ((= (V stuff) (V can))))
```

Figure 5: Examples of defModelFragment form

If-and-only-if statements	(:<=> (gas :self) (> (spec-u :self) (spec-ug :self)))
Logical Implications	(closed can)
Ordinal relation	(> (dryness stuff) 1)
Composed Set Membership	(C+ (pressures-on dir (top can)) (P stuff))
Default Relations	(default (consider (geometric-properties stuff)))
Qualitative Relations	(qprop (P stuff) (spec-u stuff))

Figure 6: Types of consequence statements in defModelFragment

3.1.1 Entities

Entities are the basic building blocks of a domain theory. They define the different kinds of objects that are modeled. Entities can be simple, with only a few parameters, such as the definition of a sphere shown in Figure 8, or complex, such as thermodynamic-stuff with many parameters, modeling assumptions, equations and numerical constraints on the parameters. We describe the set of entities defined in our thermodynamics domain model in section 3.4.

```
(defEntity name
  [ :documentation <string> ]
  [ :subclass-of superclasses ]
  [ :quantities ((quantity :dimension quantity-dimension) ...)]
  [ :consequences consequences ] )
  [ :substitutions ((symbol1 form1) )
```

Figure 7: Syntax of defEntity form

The syntax of a defentity form is shown in Figure 7. The syntax for defEntity is similar to defModelFragment forms except three differences. First, the defEntity form does not have a participants clause. This clause is implicitly bound to the object being defined and additional participants cannot be defined. Second, the *:subclass-of* clause allows a hierarchy of entities to be defined. Our language differs from CML in that we only allow an entity hierarchy and do not allow a model fragment hierarchy. Each superclass has to be the name of an entity definition. Third, for the *quantities* clause, the arguments for the quantity are bound to the object being defined. Entities inherit all

the quantities and relations from all of their superclasses and all the consequences that apply to their superclasses are defined for the entity as well.

```
(defEntity sphere
  :subclass-of (abstract-3d-object)
  :quantities ((radius :dimension length-dimension)
               (diameter :dimension length-dimension))
  :consequences ((default (closed :self)))
```

Figure 8: Defining the concept of a sphere

Background facts of the domain, which are used in most problems, are expressed using *default* statements. For example, spheres are assumed to be closed unless stated otherwise by the problem statement. The default knowledge is used only when it does not contradict assumptions given in the problem.

3.1.2 Representing Change: Actions and Processes

To reason about changes in the world we need to represent the effects of continuous and instantaneous changes. *Processes* represent continuous changes that occur over time. An example of continuous change is the temperature of water rising as a result of heating process. *Actions* are used to represent discrete instantaneous changes.

Opening or closing of a valve which in turn allows stuff to move in or out is an example of an action. Whether to use actions or processes to represent changes in a domain is determined by the types of reasoning we want to perform. In a feedback system where valves are modeled in more detail, closing or opening a valve can be modeled using processes that affect the percentage of flow the valve allows instead of having a binary

distinction.

In a dynamic world where things are changing due to processes or actions, we need to represent time and scope in our modeling assumptions and entities with respect to when they are believed to be true. We use *histories*, introduced by (Hayes, 1990) to address this problem. Histories are descriptions of objects spatially bound and temporally extended. Histories are divided into *episodes* where each episode describes what is happening to a specific object. Time is composed of an ordered set of intervals and instants. An example of a simple process and an action is provided in Figure 9.

(process heating (at stuff :begin) (at stuff :end)) (action open (at box :time1) (at box :time2))
--

Figure 9: Process and action

For most thermodynamics problems, the duration of intervals is ignored and only the ordering of time-instants corresponding to their end points is used in the analysis. In other engineering domains where time is used extensively in equations, such as physics, durations between time-intervals needs to be explicitly specified. Each statement is represented using an explicit temporal frame, so we can have equations that express relationships between time instants. For example, the first law of thermodynamics is essentially an energy equation specifying that the energy at time-instant :begin must equal to total energy at time-instant :end.

3.1.3 Mathematical Relations

There are 3 kinds of mathematical relations we need to represent. First, a parameter can have a specific value at a given time. This is represented using *nvalue* statements. For example, if a thermodynamic stuff is saturated gas, the dryness is defined as 1 as shown in Figure 10.

```
(defrelation saturated-vapor (?stuff)
  :=> (:AND (saturated ?stuff)
              (:NOT (saturated-liquid ?stuff)))
  :<=> (NVALUE (dryness ?stuff) 1))
```

Figure 10: Saturated-vapor relation

Second, the ordinal relation between two parameters can be known or used to derive other constraints. For example, the dryness is defined to be between 0 and 1 when the stuff is saturated.

And finally, we need to represent equations that relate parameters to each other. For most equations, the variables involved in the equations are predetermined and can be explicitly represented as in Figure 11.

We discuss qualitative relations in Chapter 4 in the context of qualitative reasoning.

```
(defEquation density-definition
  :participants ((stuff :type thermodynamic-stuff))
  :consequences
  ((density-definition
    (:= (density stuff) (/ (mass stuff) (v stuff))))))
```

Figure 11: Equation for density definition

The consequences field of a defEquation form can include one or more equations. When multiple versions of an equation are frequently used in the domain, these different versions can be represented under a single defEquation form. Equations that are defined through the same defEquation form cannot be combined with each other as this would result in tautologies after similar variables are canceled.

For some quantities the participants are not defined until the situation is analyzed and a *closed world assumption* is made. For example, the total energy of a thermodynamic stuff might or might not include kinetic-energy and potential energy depending on our current set of assumptions. Similarly, when expressing equilibrium of forces, the number of forces that participate in the current situation is not known at the time the model domain is written. We use set membership to define such *compositional quantities*. For the energy equation, we use C+ and C- to represent the need to be summed to find the energy of the substance (e.g. (**C+** (energy :self) (kinetic-energy :self))). Compositional modelling also allows C* and C/ to indicate that the quantity is composed of the multiple of its defined members. The set memberships can be in the consequences clause of defEntity or defModelFragment forms. Once compositional quantities are defined, we can express concepts such as the conservation of energy in a concise form as shown in Figure 12. The participant shown in Figure 12 is a process identified by its name. The process has a beginning and an end time or alternatively an inlet and outlet condition. This representation allows us to express

both processes that are time dependent, such as a piston rising slowly, and processes that are used to represent steady-state devices where the time spent inside the device is ignored.

```
(defEquation conservation-of-energy
  :participants ((name :type process))
  :consequences
  ((conservation-of-energy
    (:= (+ (Q name)
            (energy-begin name)
            (energy-inlet name))
       (+ (energy-end name)
           (energy-outlet name)
           (work name))))
```

Figure 12: Conservation of Energy equation

In addition to *nvalue* statements that specify the numerical value of a parameter, we use *can-find* and *known* statements as conditions to indicate that the numerical value can be found if desired and the numerical value is calculated respectively. We discuss lazy calculation of numerical values in chapter 6 in more detail.

Ordinal relations serve multiple purposes. First, ordinals might provide support for or contradict modeling assumptions. Secondly, ordinal information can be used in deriving expressions as a part of the problem answer. Finally, ordinal information provides upper and lower limits, allowing us to verify that the numerical results are in line with expectations.

Tables represent another source of numerical relations based on empirical data. We treat table lookups as atomic operations and do not model how interpolation on table data is done.

```
(defModelFragment saturated-table-lookup-using-Pressure
  :participants ((thermodynamic-stuff ?stuff)
                 (substance-of ?stuff ?sub)
                 (saturation-table-for ?sub ?table))
  :conditions ((saturated ?stuff))
  :consequences
  ((saturated-table-P
    (:Table-Lookup
      (Saturated-table-Lookup ?table
        ((P ?stuff))
        ((T ?stuff)
          (spec-hf ?stuff)
          (spec-hg ?stuff)
          (spec-vf ?stuff)
          (spec-vg ?stuff)
          (spec-uf ?stuff)
          (spec-ug ?stuff)
          (spec-sf ?stuff)
          (spec-sg ?stuff)))))))
```

Figure 13: Representing when saturated table lookup can be performed

3.1.4 Causal Knowledge

Equations relate parameters to each other, but do not provide information about the causal ordering of how parameters affect each other. Causal representations are discussed in detail in Chapter 4 in the context of qualitative reasoning.

3.2 Modeling Assumptions

Modeling assumptions help determine the applicability of model fragments. Modeling assumptions can imply, exclude or introduce other modeling assumptions. The logical dependencies between the modeling assumptions reflect the structure of the domain

and the relationships between model fragments. Examples of modeling assumptions include a container being open or closed, a spring being relaxed or extended, and a thermodynamic stuff being in one of three phases. Modeling assumptions can be inferred from ordinal relations, numerical values, or taxonomies, and can change as result of processes and actions. An example of modeling assumption is shown in Figure 14.

```
(defAssumptionClass thermodynamic-stuff
  :participants ((thermodynamic-stuff ?stuff))
  :consequences
    ((phase-taxonomy
      (:taxonomy (liquid ?stuff)
                  (saturated ?stuff)
                  (gas ?stuff)))))
```

Figure 14: Defining phases of a thermodynamic-stuff

The syntax of the defAssumptionClass form is similar to defModelFragment form. The most important difference is that *consequences* field of the defAssumptionClass form has to provide assumption sets with names. It is possible to define multiple assumption classes that share the same participants and conditions using multiple assumption sets in the consequences field. Being able to put multiple taxonomies in one defAssumptionClass form is useful in groping multiple assumptions together. For example, a process can be adiabatic, isothermal, isentropic, isobaric, isochoric or polytropic which can all be represented in one defAssumptionClass form. Although defining each assumption class separately would achieve the same effect, we feel more compact representations make managing the domain model easier.

```
(defAssumptionClass name
  [ :participants (participant1 ... participantn) ]
  [ :conditions (condition1 ... conditionsn) ]
  [ :consequences (assumptionset1 ... assumptionsetn) ]
```

Figure 15: Syntax of defAssumptionClass form

3.3 Defining Units and Dimensions

When dealing with numerical values, it is necessary to convert the values provided in the problem statement to a consistent set of internal units. We use *defdimension* to define dimensions and *defunit* to define the units that are valid for that dimension. The syntax for these forms is shown in Figure 16, and an example of their usage is shown in Figure 17. Units that do not have conversion factors defined for them are treated as the base units. TPS performs all internal calculations using SI units and performs the necessary conversions for input and output.

```
(defdimension name
  [ := quantity expression ] )

(defunit name
  [ :dimension dimension-name ]
  [ := quantity expression ] )
```

Figure 16: Syntax of defdimension and defunit forms

```
(defdimension length-dimension)

(defunit meter :dimension length-dimension)
(defunit mm := (* 1/1000 m))

(defDimension specific-volume-dimension
  := (/ volume-dimension mass-dimension))

(defunit m^3/kg := (/ m^3 kg))
```

Figure 17: Examples of dimension and unit definitions

3.4 TPS' Model of Common Thermodynamics Objects

We define *thermodynamic stuff* as a homogeneous collection of a substance that can be treated as a single entity. Since thermodynamics is the study of heat and work, all thermodynamics problems include at least one thermodynamic stuff. Depending on the modeling assumptions, a thermodynamic stuff can have up to 21 parameters. The most common parameters encountered in problems are temperature, pressure, mass, volume and specific volume. A thermodynamic stuff can be in liquid, saturated or gas phases. The phase of the stuff is particularly important when solving problems because the phase determines which equations and constraints hold.

3.4.1 Common Entities and Devices

A thermodynamic stuff can be examined in many different contexts. One of these contexts is as a part of a cycle. In closed cycles, a thermodynamic stuff goes through a series of cycle devices to cool or heat the environment or to produce work. The household refrigerator and heating system are everyday examples of closed cycles. In

open cycles, the thermodynamic stuff is taken from a reservoir, which is assumed to be infinite. For example, the combustion cycle of automobile engines takes in and uses air during the process.

An important idealization of cycle devices as analyzed in textbooks is that they do not accumulate any thermodynamic stuff inside them, and have a constant mass flow of thermodynamic stuff through them. A description of the cycle devices that are modeled in TPS follows.

Cooler, Heater and Heat-Exchanger: A cooler ejects heat energy from the working fluid to the environment and a heater injects heat energy from the environment to the working fluid. A heat exchanger is constructed using a cooler and a heater in combination such that the energy is transferred from one fluid to another. Typical assumptions about a heater or a cooler would be to assume that inlet and outlet pressures are equal (i.e. isobaric). Schematic representations for these devices are shown in Figure 18.

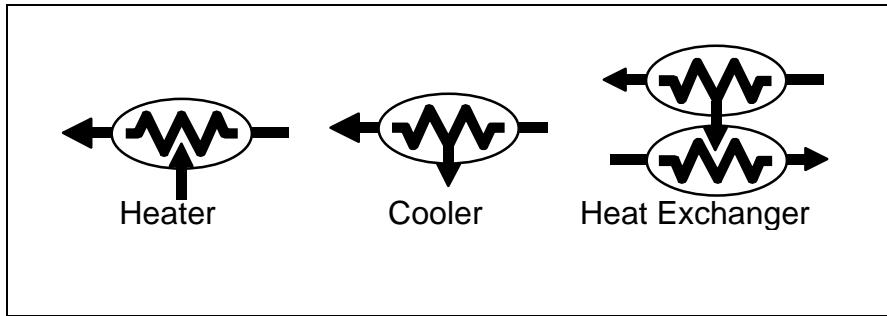


Figure 18: Schematic representations for heater cooler and heat-exchanger

Mixer and Splitter: Mixer and splitter are structural components responsible for the joining and separation of fluid flows. Schematic representations for these devices is shown in Figure 19.

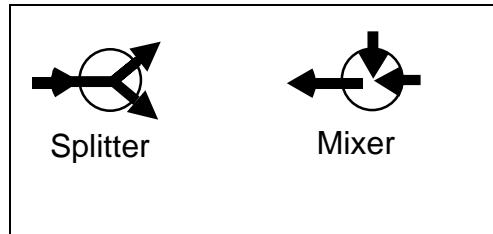


Figure 19: Schematic representation for Splitter and Mixer

Turbine and Compressor: A turbine allows the high-energy stuff to expand causing the blades of the turbine to spin and create work. A compressor uses mechanical energy to compress thermodynamic stuff. Because saturated stuff can harm the turbine blades, the input of the turbine is often high-energy gas and the output is thermodynamic stuff very close to saturation. For a compressor, since compressing liquids requires a large amount of energy, the input and the output phases of the stuff

are gas. The common assumptions for turbines and compressors are isentropic, isothermal, adiabatic, polytropic or non-polytropic. The schematic representation for these devices is shown in Figure 20.

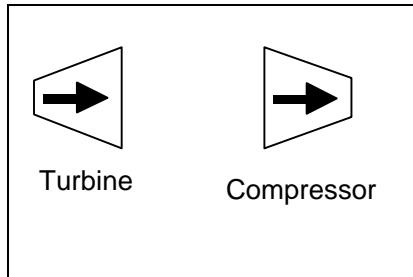


Figure 20: Schematic representation for turbine and compressor

Throttle: A throttle allows the thermodynamic stuff to expand without producing any work. This effect is achieved using a pipe that has a larger outlet than an inlet.

Pipe: A pipe is a conduit that connects containers. Pipes are fitted with valves that regulate flow and are modeled as not to contain any thermodynamic stuff. To reason about pipes with thermodynamic stuff inside, a series of pipes and containers can be chained together.

Nozzle: A nozzle uses geometry to compress the thermodynamic stuff without producing any work by taking advantage of geometry.

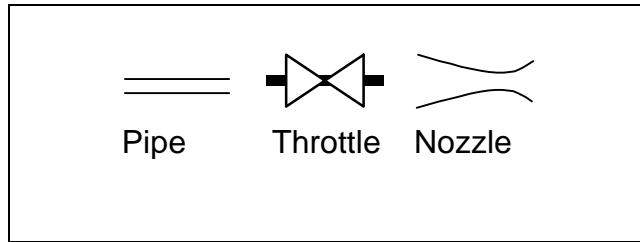


Figure 21: Schematic representation for a throttle

Pump: The primary purpose of a pump is to create the necessary mass flow through the system. Because of the geometric constraints imposed on a pump, the inlet of a pump cannot be gas. Because the amount of energy needed to operate the pump is a small fraction of the work produced by the turbine, the work input to the pump can be ignored in many problems.

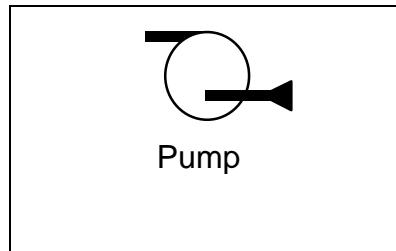


Figure 22: Schematic representation for a pump

We have been described some of the commonly found steady state cycle devices above. An important commonality of these devices is that they are treated as black boxes and the internal workings are never examined in introductory thermodynamics. Next we describe every day objects that are used in thermodynamics problems.

Block: A block is a solid object often used as the top of a container to create a piston. Since the sum of the forces and pressures on a block must be equal when the block is

stationary, a block is ideal for setting up equilibrium conditions.

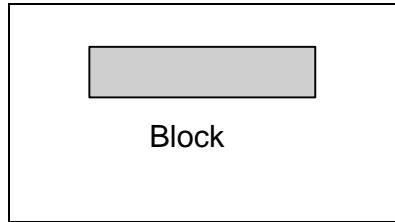


Figure 23: Schematic representation for a block

Container: A container is a receptacle capable of holding thermodynamic stuff.

Containers can be open or closed, rigid or flexible and in various shapes. Containers can be connected to each other through pipes with valves. Different types of containers are shown in Figure 25

Valve: Valves provide connections between containers, allowing thermodynamic stuff to flow between them. A valve can be open or closed.

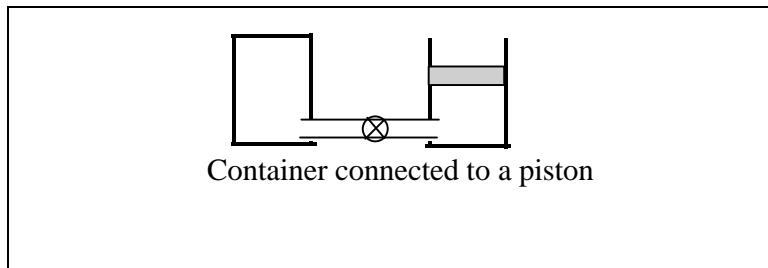


Figure 24: Schematic representation of connected containers

Piston: A piston is a complex entity composed of a container and a movable block. A piston does not have any parameters of its own, but the relations describing a piston are

used to link the parameters for the container, thermodynamic stuff inside, and the block serving as the lid.

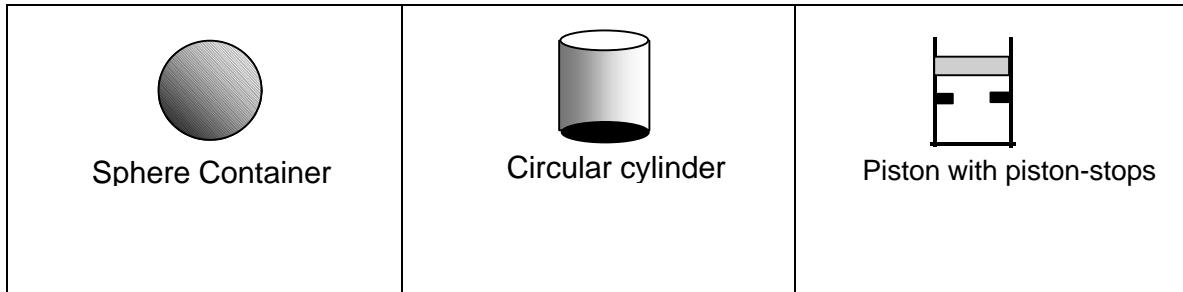


Figure 25: Schematic representation for container types

Piston-Stop: Piston stops provide geometric constraints on the movement of the block. Piston stops provide minimum or maximum height for the piston top, which in turn determine the range of volume the thermodynamic stuff inside can occupy.

Spring: A spring exerts a force that is linearly proportional to the distance from its relaxed length. A spring can be placed inside or outside a container. The spring can be attached to the piston block or freely hanging such that the block touches and pushes against the spring only at a certain height.

External Force: An external force can be applied to a piston, compressing or expanding the thermodynamic stuff inside it. The magnitude of the force can be constant or be determined by a specific equation provided in the problem.

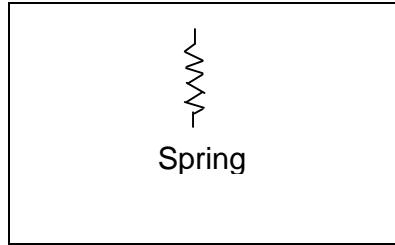


Figure 26: Schematic representation for a spring

3.4.2 Entity and Device Distribution in Problems

Figure 27 shows the distribution of some of the more common devices found in the first four chapters of (Howell & Buckius, 1992). Only problems that are solved by TPS are considered for this analysis. We discuss which problems are excluded from our test set in the evaluation section in Chapter 8. The percentages are given to show the relative frequency of occurrence of different objects. It is important to note that different textbooks take different approaches in how thermodynamics should be taught and would result in different distributions. For example, some textbooks (such as (Whalley, 1992)) emphasize cycles from the first chapters. One of the reasons that we chose (Howell & Buckius, 1992) is that it provided the most number of problems per chapter compared to any other textbook.

The most common entities are, as expected, static and steady state thermodynamic stuff. Static thermodynamic stuffs appear in containers, pistons or when the object is being analyzed without any reference to a container. Steady state stuffs are in continuous flow and appear in cycle devices. Although the static and steady state stuff

share many common parameters, because the units for these parameters are different they have to be treated separately. Piston and container problems present simple situations and help build students understanding. Cycle devices appear in the later chapters and more advanced courses in thermodynamics often deal exclusively with cycles.

Objects	Number of Problems	Percentage of Problems
Static Thermodynamic Stuff	119	82%
Steady-State Thermodynamic Stuff	28	19%
Piston	37	26%
Container	31	21%
Turbine/Compressor	13	9%
Pipe/Nozzle/Throttle	9	6%
Splitter/Mixer	5	3%
Heater/Cooler/Heat Exchanger	5	3%

Figure 27: Distribution of devices

3.5 Taxonomy of Thermodynamics Problems

In determining how to approach a particular problem, it is useful to have generalizations about the type of problem at hand. This is one of the primary functions of creating a taxonomy of problems. Classification allows for efficient decision making in terms of resource allocation and acts as a guide for choosing a control volume. In addition, a clear taxonomy of problems is useful in extending the problem solver to even broader coverage of thermodynamics problems.

Thermodynamics problems can be classified along several dimensions. One classification is *static* versus *dynamic* problems. Problems that do not have any actions or processes in them are static problems. Static problems are typically easier than dynamic problems and are often used in introductory chapters to clarify thermodynamics concepts. For example, all the textbooks we have studied include a set of problems where the student is asked to specify whether the thermodynamic stuff can be considered an ideal gas for the given parameter values. Static problems emphasize basic principles of thermodynamics. On the other hand, dynamic problems involve systems that change through actions or processes. These types of problems can involve multiple sequences of events.

Another dimension of categorizing thermodynamics problems is by examining how control volumes are used. Steady-state steady-flow (SSSF) problems involve a working fluid going through a sequence of devices. Each device in turn is used as a control volume for analysis. Uniform-state uniform-flow (USUF) problems involve accumulation of thermodynamic stuff from a constant source. Problems about inflating a balloon or a container getting filled fall into this category. Contained stuff problems involve a process acting on a simple thermodynamic stuff. A piston with stuff inside being heated falls into this category. The initial classification of the problem allows us to choose a control volume, which in turn specifies and simplifies the equations that can be applied.

3.6 Chapter Summary

In this chapter we have

- Outlined the primitives for our domain model
- Provided examples illustrating how entities, processes, actions, modeling assumptions and units are represented in TPS
- Described the models for different objects and the different types of problems found in textbooks.

In the next chapter, we describe how qualitative reasoning is used in our problem solver.

4. Qualitative Reasoning in Problem Solving

Problem solving is a combination of quantitative and qualitative analysis. In this chapter, we examine how qualitative reasoning helps in understanding the consequences of actions and processes.

One area in which qualitative reasoning plays a crucial role in problem solving is in addressing the frame problem(McCarthy & Hayes, 1969). In this domain, the frame problem includes deciding what parameters are changing from one state to the next. One method for dealing with the frame problem is the STRIPS representation (Fikes & Nillson, 1971). Each state is composed of a set of facts that are known to be true about the world. Operators enable transitions from one state to the next by adding and deleting facts. For example, the *pickup* operator in the Blocks World domain would add (*Holding Block-A*) statement and delete the (*Hand-Empty*) statement to achieve a new state.

Although STRIPS operators have been used in planning and problem solving, the representational requirement of having complete knowledge of the world makes this approach intractable in large domains. There are two drawbacks to using the STRIPS model. First, creating a new state as a result of each operator results in a large number of states where state information that is not affected has to be stored many times. Second, STRIPS model does not provide sufficient support for reasoning about continuous change, which is essential in engineering domains.

Qualitative reasoning addresses the frame problem by using histories which are extended through time and spatially bounded (Hayes, 1985). This approach allows us to create world-models that do not require complete knowledge and also solves the state explosion problem by specifying that only objects in the same location can interact with each other. Qualitative Process Theory (Forbus, 1984a) introduces processes as the agents of change and describes a framework for reasoning with multiple processes. Heat transfer between objects, acceleration due to gravity and the effects of friction-force are examples of processes that have been modeled in qualitative physics.

Qualitative reasoning provides a causal ordering for parameters which can be used both for generating explanations and for understanding cause and effect relationships among parameters. A drawback to using purely qualitative reasoning is its inherent ambiguity. For example, several possible states can follow a given state, if knowledge about magnitudes of differences is unavailable. Similarly, when a parameter is both positively and negatively indirectly influenced, the sign of the derivative for the parameter cannot be determined using purely qualitative reasoning alone. Although it is possible to create more detailed qualitative domain models to avoid such ambiguities (such as by specifying the ordinal relation between the two opposing rates), in the context of engineering problem solving a more simple and an elegant solution is to use the numerical values to constrain the qualitative reasoner.

Qualitative reasoning is used in TPS for the following tasks:

- To derive what assumptions are not affected by the process or action
- To derive parameters which remain constant
- To determine if there are qualitative states that are not mentioned in the problem statement

In the next section, we describe the definition of qualitative state and our extension to Gizmo (Forbus & de Kleer, 1993).

4.1 Defining Qualitative States

A *qualitative state* is a set of propositions that characterize a qualitatively distinct behavior of a system. A qualitative state can abstractly represent an infinite number of quantitative states. For example, the temperature of water in a pot being heated increases continuously, so the quantitative state is changing. However, the qualitative state remains unchanged until a *limit point* is reached.

Qualitative states partition the behavior of the system into natural units for analysis. A qualitative state can have transitions to several next states. A collection of qualitative states and transitions that describe the behavior of the system is called an *envisionment* (de Kleer, 1977). There are two types of envisionments commonly used for analyzing systems. An *attainable envisionment* describes all states that are reachable from a given initial state (Forbus, 1987). A *total envisionment* can be

considered a collection of attainable envisionments where all initial states for the system are considered. In engineering problem solving, we only need a subset of an attainable envisionment that characterizes the problem under analysis.

The problem statement provides a partially specified history. Solving the problem requires filling out some more of this specification using the domain knowledge. A partially specified history includes numerical values as well as facts known to be true for that state. Unlike a qualitative state, a partial problem specification is not complete, because not all ordinal relations and assumptions are specified. One or many partial problem specifications can map into a single qualitative states, and a single partial problem specification can map into multiple qualitative states. For example, water at 20 C and at 30 C under standard atmospheric pressure would be described using a single qualitative state since it is liquid in both cases. If the pressure was not specified, because water at 20 C can be either in saturated phase or liquid phase, both possible qualitative models will have to be considered as a possible starting point in analyzing the problem.

Transitions from one partial problem specification to another can map to zero, one or multiple qualitative state transitions (see Figure 28). For example, in the case of water being heated from 20 C to 30 C under standard atmospheric pressure, the transition from the initial partial problem specification to the final partial problem specification maps into zero qualitative transitions. If the final state was specified as

30 C and as being saturated rather than liquid than the problem specification transition would map to one qualitative state transition. If the final state was described only using temperature then both saturated, liquid and gas phases would have to be considered. In this case, the problem specification transition would map to multiple qualitative state transitions.

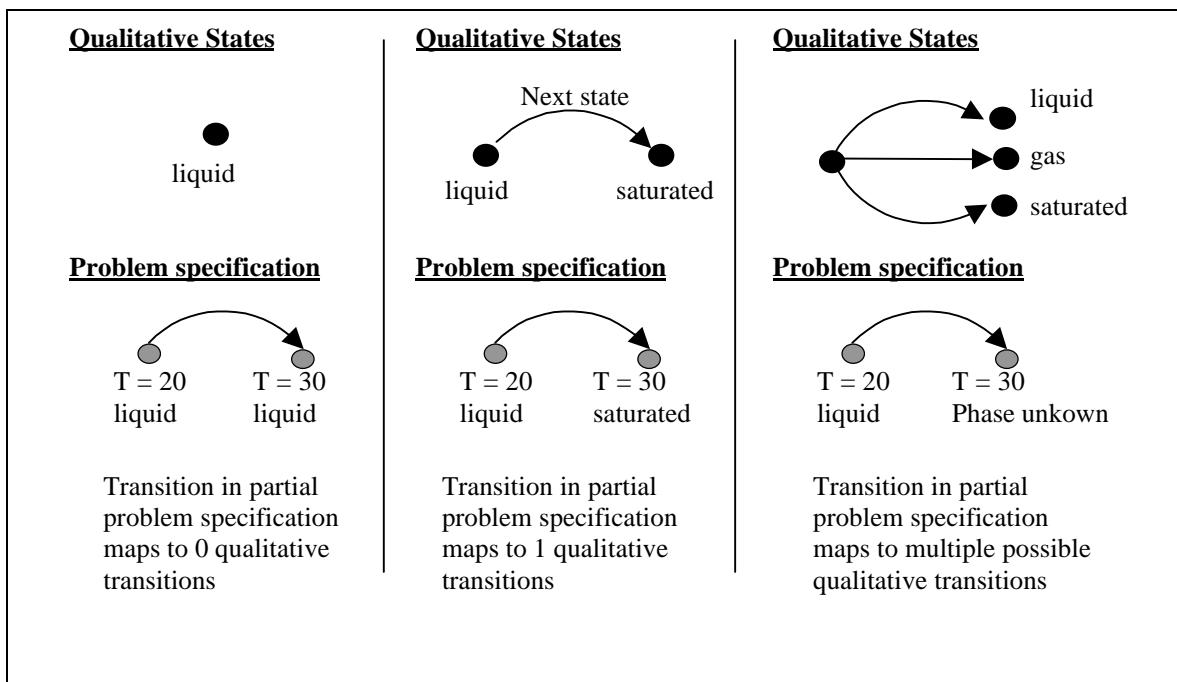


Figure 28: Mapping from problem specification to qualitative states

We use Gizmo (Forbus, 1984b), developed by Ken Forbus, to perform the necessary qualitative analysis. The initial partial problem specification becomes the scenario for Gizmo. The domain knowledge relevant to qualitative reasoning is provided to Gizmo as the domain model. Gizmo creates a set of qualitative states that match the initial partial problem specification.

Performing limit analysis on the initial qualitative states provides possible matches for the partial problem specifications that may follow. Two types of changes precipitate state transitions. First, processes such as heating or cooling affect the numerical values of parameters, eventually resulting in changes in the ordinal relations between parameters. These are known as continuous changes. Secondly, discrete actions such as opening a valve result in discontinuous changes.

4.1.1 Extensions to Gizmo

We have extended Gizmo to reason about discrete actions. Discrete actions, such as opening a valve, cause changes that result in qualitative state transitions. We took the work described in (Forbus, 1989) as a basis for these extensions.

For simplicity we placed two restrictions on actions:

1. *Single action*: Only one action can be taken at a time
2. *No Overlap*: Actions cannot overlap with state transitions caused by limit hypotheses

Since actions can be arbitrarily complex, the *single action* assumption does not make us lose any generality in our representation. On the other hand, the *no overlap* assumption requires that the actions occur quickly compared to physical changes, since no reasoning about physical quantities is done while the action is in progress. For

engineering problem solving, this restriction does not represent any problems because all actions in problems are atomic. To model actions that take a significant amount of time, such as adjusting the hot and cold taps of a shower, we can model the action as a series of atomic actions that are executed consecutively.

```
(defOperator (Open ?valve)
  :preconditions ((Valve ?valve))
  :delete-list (((:= (state ?valve) closed)))
  :add-list    (((:= (state ?valve) open)))
```

Figure 29: Operator definition

An action has a set of add and delete statements. For an action to be active, the preconditions must be satisfied in the current qualitative state. The next set of qualitative states that would be produced after the action is taken is generated as follows:

1. Delete statements in the delete list from the set of assumptions defining the qualitative state.
2. Add statements as assumptions for the new qualitative state.
3. Remove preconditions that are contradictory to the newly added assumptions until the state is no longer contradictory. This ensures that preconditions that are not implied by the current assumptions are kept in the new state.

4. The standard state completion generation algorithm is evoked to complete the states and derive the necessary statements from the base assumptions of the state.

There can be zero, one or many qualitative states produced as a result of the action. If there are zero states, this signifies that our definitions for the action and the domain model are contradictory. For example, if the add-list included both valve-open and no fluid-flow then we would get a contradiction. Deriving multiple next states that can be reached as a result of taking the action shows that there is ambiguity after the action is taken.

Constraining the domain model or defining actions more precisely can reduce this ambiguity. In cases where this is not possible, the multiple final states produced by the qualitative reasoner must be explored. For example, opening a valve that connects two tanks containing stuff with varying temperatures and pressures would result in a fluid flow. When the system reaches equilibrium, the stuff inside can be liquid, saturated or gas depending on the initial numerical values. The intermediary and the final qualitative states must be compared to the partial problem specifications to determine the specific path.

4.2 Finding Constant Parameters

An important factor to consider when dealing with multi-state problems is deciding which assumptions and entities can be carried over from the initial state to the final

state. For solid objects, the decision is relatively simple. If we have a container in the initial state, we presume that the container continues to exist for all of the following qualitative states; however, deciding whether the container that was open in the initial state remains open requires more elaborate reasoning. For fluids the problem is further complicated since fluids can be combined or split, changing the number of entities in the system.

Qualitative analysis provides the necessary knowledge to simplify and combine equations for problems with more than one state. For example, a typical problem in thermodynamics is considering the consequences of heating a thermodynamic stuff. In the simplest case, where an ideal gas is heated, the ideal gas law (shown in Figure 30) applies to both the initial and final states. By performing a simple qualitative analysis of the problem, a problem solver can conclude that the mass (m) and the gas constant (R) remain constant when stuff is heated. This knowledge allows combining the application of the ideal gas law to the initial state with its application to the final state to create a simpler equation. In cases where the mass of the substance is not known, this simplification is crucial to solving the problem.

Ideal Gas Law applied to initial state: $P_{\text{Initial}}V_{\text{Initial}} = m_{\text{Initial}}R_{\text{Initial}}T_{\text{Initial}}$

Ideal Gas Law applied to final state: $P_{\text{Final}}V_{\text{Final}} = m_{\text{Final}}R_{\text{Final}}T_{\text{Final}}$

Qualitative analysis shows m and R to be constant

Ideal Gas Law combined: $P_{\text{Initial}}V_{\text{Initial}}/T_{\text{Initial}} = P_{\text{Final}}V_{\text{Final}}/T_{\text{Final}}$

Figure 30: Combining ideal gas equation

For equations that involve parameters from multiple states, such as the first law of thermodynamics, determining which parameters are constant is essential to simplify the equation. When applying the first law of thermodynamics (shown in Figure 31), assumptions and knowledge of what parameters remain constant between states must be used for simplification. Although there are a total of 19 variables in the first law, there are very few situations when all the parameters are used. For example, when examining steady state devices, such as a turbine, variables with subscripts 1 and 2 are ignored. Qualitative analysis of the turbine can further eliminate kinetic energy and potential energy since their effects are usually negligible. The version of the first law that is typically used for turbines is given in Figure 32.

	$Q_{c.v.} + \sum m_i(h_i + V_i^2/2 + gZ_i) = \sum m_e(h_e + V_e^2/2 + gZ_e) +$	
	$(m_2(h_2 + V_2^2/2 + gZ_2) - m_1(h_1 + V_1^2/2 + gZ_1)) + W_{c.v.}$	
Legend for Variables:		Subscripts Used:
state	m mass	c.v. Control Volume
state	h enthalpy	i parameter at inlet
	V velocity	e parameters at exit
	g gravity	1 parameters inside the cv in the initial
	Z height	2 parameters inside the cv in the final
	Q Heat	
	W Work	

Figure 31: First law of thermodynamics

$$Q_{c.v.} + m_i h_i = m_e h_e + W_{c.v.}$$

Figure 32: Simplified version of first law typically used for turbines

4.3 Discovering Hidden States

We have discussed problems that have two partial problem specifications that map to an initial and a final qualitative state. However, a complex problem can map to a sequence of qualitative states. In these cases, new episodes need to be defined and the problem history needs to be expanded. For example, when a thermodynamic stuff is heated it goes through liquid, saturated and gas phases. Multiple qualitative states corresponding to each of the phases has to be used in analyzing the problem.

Total envisions are useful in diagnosis, in determining failure modes and in effects analysis. However, they produce a large number of states even for small examples. This makes total envisions impractical for most applications. Using an

attainable envisionment, which is the generation of all states from a given complete state, is more feasible in most cases.

Although the number of states produced in an attainable envisionment is significantly smaller than total envisionment, it is still more powerful than what a typical problem solver needs. We generate a constrained attainable envisionment for analyzing a problem. Determining whether the initial and final partial problem specifications given in the problem statement are the only states requires performing a constrained qualitative attainable envisionment. This type of envisionment examines the possible behaviors of the system before it reaches the final state. In most cases, the constrained attainable envisionment produced is a strict subset of the attainable envisionment starting at the initial state. In the worst case, the states produced by constrained attainable envisionment are identical to the states from attainable envisionment. The heuristic for constrained attainable envisionment is shown in Figure 33.

<p>Step 1: Find the initial set of states that match the initial conditions given in the problem statement.</p> <p>Step 2: For each initial state find the set of next states.</p> <p>Step 3: If any of the next states matches the final state, end state generation. If there are no matching states, then go to Step 2, using the newly found states as initial states.</p>
--

Figure 33: Constrained attainable envisionment heuristic

It is possible that the attainable envisionment starting from the initial state can produce multiple qualitative states that qualify as final states. In these cases, our

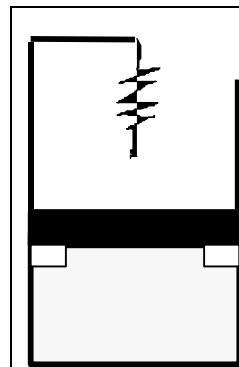
heuristic finds only the first matching qualitative state. For some tasks, such as diagnosis, finding all possible states is necessary; however, problem solving is a more constrained activity where finding the first qualitative matching state is often sufficient for analysis. It is possible that TPS will find a final qualitative state which is consistent with the problem description given, but is not the actual final qualitative state. In this case, further qualitative states that should have been considered by TPS will never be examined possibly leading to inaccurate results in the problem solution. Although we have not encountered such a situation in practice, this possibility exists.

The set of states generated while performing a constrained attainable envisionment are the hidden states not explicitly mentioned in the original problem. For example, when analyzing an oscillating spring where the initial state specifies the compressed length and the final state specifies the partially stretched length of the spring, constrained attainable envisionment is necessary to identify when the spring is relaxed. Similarly, solving a thermodynamics problem where liquid water is heated in a closed container can require performing constrained envisionment to determine the different states the substance might go through.

Hidden states arise from changing ordinal relations between parameters that could be due to shifts in spatial configuration of objects or internal parameters of an object. A well-defined domain theory identifies the necessary comparisons in the domain, which are used in defining state transitions. The changes in ordinal relations between

parameters correspond to changing equations and views of the system. For example, we call a spring compressed, relaxed or stretched depending on the relationship between length of spring and the rest length of spring. Since our qualitative reasoning system uses limit points, analyzing damped oscillation does not create an infinite number of states, as would have been required if the QSIM(Kuipers, 1986) qualitative simulator were used.

In Figure 34, we show a typical problem involving a piston and a spring taken from an introductory thermodynamics textbook (Wylen & Sonntag, 1985).



A frictionless piston having cross-sectional area of 0.06m^2 rests on stops on the cylinder walls such that the contained volume is 30L . The piston mass is such that 300kPa pressure is required to raise the piston against ambient pressure outside. When the piston has moved to the point where the contained volume is 75L , the piston encounters a linear spring that requires 360kN to deflect it 1m . Initially the cylinder contains 4kg of saturated (liquid+vapor) water at 35C . If this system is heated until the final pressure inside is 7MPa , determine the work done during this process.

Figure 34: Problem 4.21

The total amount of work done by the system is the combination of work done against air pressure and against the spring. The equations for work are given in Figure 35.

Adiabatic Expansion: $W = \text{Pressure}_{\text{final}} (\text{Volume}_{\text{final}} - \text{Volume}_{\text{initial}})$

Work against Spring: $W = (\text{Pressure}_{\text{initial}} + \text{Pressure}_{\text{final}})/2 * (\text{Volume}_{\text{final}} - \text{Volume}_{\text{initial}})$

Figure 35: Work equations for problem 4.21

Qualitative analysis of the problem is needed to determine that two different models of work are required. The first model is applicable to the piston rising, and the second model is applicable to the piston pushing against the spring. This problem needs to be broken down into a minimum of 3 states: piston resting, piston touching spring and piston compressing spring. Doing the qualitative analysis we find there are 5 different qualitative states. The relevant qualitative relations and the corresponding work equation are given in Figure 36.

In this example, the initial state is well-constrained and there is only one possible qualitative state. The final state is given only in terms of pressure. Examining how the value of pressure changes using Figure 36, we see that pressure only increases when the piston is pushing against the spring. Since the initial pressure is less than 7MPa, we know that the piston pushing against the spring is the first qualitative state that can satisfy our criteria for being the final state of the problem.

1. Piston resting on Stops	Pressure(atmosphere) > Pressure(stuff) Pressure(stuff) is constant, Volume(stuff) is constant $W = 0$, no change in volume
2. Piston just touching stops	Pressure(atmosphere) > Pressure(stuff) Pressure(stuff) is constant, Volume(stuff) is constant $W = 0$, no change in volume
3. Piston Rising	Pressure(atmosphere) = Pressure(stuff) Pressure(stuff) is constant, Volume(stuff) is increasing $W = \text{Pressure} (\text{VolumeFinal} - \text{VolumeInitial})$
4. Piston just touching spring	Pressure(stuff) > Pressure(atmosphere) Pressure(stuff) is constant, Volume(stuff) is increasing $W = \text{Pressure} (\text{VolumeFinal} - \text{VolumeInitial})$
5. Piston pushing spring	Pressure(stuff) > Pressure(atmosphere) Pressure(stuff) is increasing, Volume(stuff) is increasing $W = \text{PressureInitial} + \text{PressureFinal} /2 * (\text{VolumeFinal} - \text{VolumeInitial})$

Figure 36: States generated by qualitative analysis

4.4 Chapter Summary

Our contributions to qualitative reasoning can be summarized as follows:

- Finding constant parameters for multi-state problems
- Finding hidden states that are not made explicit in the problem statement

These techniques are necessary for all domains which involve multi-state problems.

In Section 5.4, we discussed the importance of domain abstractions and its importance as control knowledge in thermodynamics.

5. Plans: Frameworks for Solutions

Plans provide the frameworks for solutions. They reflect expert control knowledge, a key ingredient in the skill of problem solving. In this chapter we examine how plans are represented and used in the IPSA architecture.

Plans represent typical ways of solving problems. Good thermodynamics textbooks provide explicit plans for students to follow for complex problems and demonstrate their usage through examples. In knowledge rich domains, where the number of equations one can start from is large, plans provide constraints that narrow down the problem space. An example of an explicit plan taken from (Wylen & Sonntag, 1985) is shown in Figure 37.

- Decide on a control volume
- Determine what is known at the initial state
- Determine what is known at the final state
- What assumptions are known about the process
- What is the model for the behavior (phase changes, ideal-gas etc)
- Analyze system using first law

Figure 37: Problem analysis technique

Some of the earlier research that examined differences between expert and novice problem solving proposed that experts used forward chaining when solving problems(Larkin, McDermott, Simon, & Simon, 1980). The plan shown in Figure 37

can be viewed as supporting this view since the goal of the problem is not mentioned anywhere in the plan. Further research in the area of expert and novice differences seems to indicate that expert behavior in knowledge rich domains is significantly more complex and diversified (Priest, 1986; Priest & Lindsay, 1992).

TPS uses two kinds of plans to interleave backward and forward chaining while problem solving: *goal-centered* plans and *problem-centered* plans. Goal-centered plans provide methods for achieving the current goal. For example, trying different methods to find the phase of a thermodynamic stuff would be represented as a goal-centered plan. Problem-centered plans are suggested based on the structure of the problem and are not tied to a specific goal. Problem-centered plans, such as the plan shown in Figure 37, are not guaranteed to lead directly to the solution, but rather present a uniform way of approaching problems.

Using a combination of goal-centered and problem-centered plans helps create a more expert like solution than that could be achieved by either type of plan alone. We conjecture that expert problem solutions balance being goal directed and using a solving framework. For example, a problem solver with only goal-centered plans would possibly take very different approaches to find two similar parameters because of the equations and relations the different parameters participate, in ignoring the similar structures of the two problems. A problem solver with only problem-centered plans would suffer from constructing solutions that did not necessarily get to the

desired parameter, except by accident. The plan shown in Figure 37 is a problem-centered plan that invokes a series of new goals to be pursued, providing a framework for the problem while still being goal-directed through its subplans.

Plans are used for the following tasks:

- To express typical ways of solving problems
- To interleave forward and backward chaining
- To group similar methods for achieving the same goal
- To represent connections between domain primitives

In the next section, we describe how we represent plans and provide examples.

5.1 Representing Plans

We took the general planning framework introduced by the RAPS system (Firby, 1989) in implementing plans. We extended this general framework and integrated it with the rest of our domain model represented using compositional modelling language. The syntax of a defPlan form is shown in Figure 38 and two example plans are shown in Figure 39.

```
(defPlan <name>
  :index <index>
  :triggers <triggers>
  :succeed <condition>
  :steps [<step-name> <goal> <tests>]
  :methods [<method-name>
    :context <condition>
    :steps <steps>] )
```

Figure 38: defplan syntax

The *name* of the plan is used to identify the plan when debugging and explanation.

The *triggers* are a set of statements that have to jointly hold before the plan is considered as a candidate. The *index* clause provides the means for one plan to refer to another plan. The bindings in triggers field must be compatible with any arguments that are mentioned in the index clause. The *succeed* clause indicates when the plan has achieved its goal and can be terminated without finishing all its steps. A plan has either a set of *steps* or a set of *methods* that can be used in different *contexts*. Methods are useful for gathering plans that achieve the same goal together. Each step in a plan has a *step-name*, a *goal* for the step and possibly a set of conditions that must hold once the goal is achieved. If the conditions of the step are not satisfied, the plan fails and further steps are not executed. Any values obtained through other plans or through model fragments can be accessed using additional bindings in the conditions. For example, if the numerical value is required before the plan can process (*nvalue <param> ?found-value*) can be used binding the value to the variable *?found-value* to be used in following plan steps. Similarly, the plan shown in Figure 39 uses (*problem-goal ?goal*) to obtain the goal before proceeding to step t4.

```
(defplan initial-problem-analysis
  :index (initial-analysis)
  :steps ((t0 (analyze-assumptions-about-entities)
              (entity-assumptions-analyzed))
          (t1 (choose-control-volume)
              (control-volume ?control-volume))
          (t2 (choose-frame-equation)
              (frame-equation ?equation))
          (t3 (solve-equation ?equation)
              (solved ?equation) (problem-goal ?goal)))
          (t4 ?goal (solved ?goal)))))

(defplan fix-state-using-saturation-table
  :index (fix-state ?stuff)
  :triggers ((basic-thermodynamic-stuff ?stuff)
              (substance-of ?stuff ?sub)
              (saturation-table-for ?sub ?table)))
  :succeed ((phase-of ?stuff ?phase))
  :methods
  ((T-spec-v-lookup
    :context ((can-find (T ?stuff)))
    :steps ((t0 (T-spec-v-lookup ?stuff ?table)))))
  .....
  (P-spec-h-lookup
    :context ((can-find (P ?stuff)))
    :steps ((t0 (P-spec-h-lookup ?stuff ?table)))))))
```

Figure 39: Examples of defplan forms

Plans guide problem solving through ordering and introducing goals. There are also two kinds of suggestions that complement plans: *and-subgoals* and *ordered-subgoals*. The syntax for these suggestions is shown in Figure 40. And-subgoals and ordered-subgoals represent a series of goals that need to be solved to achieve the current goal. And-subgoals and ordered-subgoals are generated from the equations, equalities and table-lookup relations provided in the domain model. Unlike plans, which do not guarantee to solve the current goal, and-subgoals and ordered subgoals are guaranteed to achieve the current goal if all the subgoals can be solved. For example, if the current goal is to find the density of thermodynamic stuff, a subgoal to find the value for mass and volume will be generated based on the definition for

density. When the values for mass and volume is found, the value for density can be calculated by substituting them into the density definition equation.

```
(and-subgoals <goal> <subgoals>)

(AND-SUBGOALS
  (SOLVE-SUBGOALS (CAN-SOLVE-RELATION (WEIGHT (AT CAN :BEGIN))
  (WEIGHT-OF-OBJECT 2)))
    ((FIND (VALUE (MASS (AT CAN :BEGIN)))) 
     (FIND (VALUE (ACCELERATION GRAVITY)))))

(ordered-subgoals <goal> <subgoals>)

(ORDERED-SUBGOALS
  (CAN-SOLVE-RELATION (WEIGHT (AT CAN :BEGIN)) (WEIGHT-OF-OBJECT
  2))
    ((SOLVE-SUBGOALS
      (CAN-SOLVE-RELATION (WEIGHT (AT CAN :BEGIN)) (WEIGHT-OF-
OBJECT 2))
        (CALCULATE (CAN-SOLVE-RELATION (WEIGHT (AT CAN :BEGIN)) (WEIGHT-
OF-OBJECT 2)))))))
```

Figure 40: Basic suggestions

While the subgoals in and-subgoals statement can be reordered by the system, the ordered-subgoals represent a priori ordering that cannot be altered. For example, to find the value of a variable using an equation with three variables we need to determine the value of two of the parameters. The order in which these variables are calculated is insignificant. When finding the value of a variable, we can delay our numerical calculations until the very end. This is represented using ordered-subgoals where the problem solver must find a method for calculating the value before it can do the calculations.

5.2 What should be represented in plans?

In representing a domain, what aspects of the domain should be represented in terms of plans is part of the art of writing a domain model. Basic concepts in the domain can be represented using model fragments, but when we need to group concepts or impose an order on how they are used plans are needed. We have followed the following guidelines in writing our plans:

- Any explicit guidelines given in textbooks should be represented as plans.
These guidelines represent the accumulation of many years of expertise.
Unfortunately, none of the textbooks we have examined gave detailed guidelines except for the first plan shown in Figure 39 that seems to be commonly adopted by most textbooks.
- A sequence of equations that occur frequently should be represented as plans.
Expert knowledge is required to group functionally similar equations together before plans can be used to organize their sequence.
- Frame equations, as discussed in 6.1.3, are often central to the analysis of problems. Deciding what frame equations to use, and resolving of assumptions that will allow frame equations to be chosen should be represented in plans.

5.3 Common Problem Solving Primitives

Although the plan examples we provide above are applicable for thermodynamics, the

underlying planning language is applicable to general engineering problem solving. We highlight some of the important predicates that are used in plans below.

(find (nvalue <parameter>)): The numerical value goal is often used as the goal for the problem or when the parameter values have to be calculated. The solution when found is indicated as follows: (solution-for (find (nvalue <parameter>)) <value>). There are four ways a numerical value can be found: through equations, through a table lookup, through an equality relation or through an assumption that explicitly sets the value. For example, our domain model would assert the statement (nvalue (dryness <stuff>) 1) when the thermodynamic stuff is a saturated liquid. The nvalue statement and the find-nvalue statement derive the solution statement shown above.

(nvalue <parameter> <value>): The value or a parameter is represented using this form. Nvalue statements are found in problem descriptions as well as a part of the domain model when the set of assumptions determines the value of the parameter. The sussed abstraction is helpful in marking parameters as can-find and only calculating the numerical value if necessary.

(known <parameter>): A parameter with a known numerical value is marked as known. In cases where we do not need to bind an additional variable to the value of the parameter, known statement is used instead of an nvalue statement. For example,

because numerical values have to be known for table lookups, the model fragment for table lookup uses known statements in its conditions field. It is not possible to use nvalue statements in the conditions field since this will introduce a new variable binding as the value of the parameter.

```
(nvalue <param> <value>) <=> (known <param>)
```

(find (value <param>)): The find-value goal is used to find a method that would make it possible to solve the parameter without necessarily solving it. This allows us to delay numerical calculations until the end of the problem in most cases. In problems where table-lookups are used, the numerical values needed to perform table lockups is calculated as needed.

(can-find <param>): A parameter is marked as can-find if there is a method, such as an equation, that can be solved to find the value of the parameter.

(calculate <param>): The find-numerical-value goal is converted to an ordered set of two goals: find-value and calculate. Can-find-method indicates which method can be used to use to calculate the numerical value, values and assumptions that participate in the method change with different methods and when the (calculate <param>) is expanded the numerical value is calculated and added to our solution.

```
(can-find-method <param> <method>) ^  
(values and assumptions participating in the method) ^  
(calculate <param>) => (nvalue <param> <value>)  
  
(given <statement>), (given-equation <equation>), (problem-goal
```

<goal>: Given, given equation and problem-goal are used to specify statements that were provided in the problem definition. The statements provided in the given statements are asserted, and equations provided are treated in the same way as equations generated at runtime.

(solve-equation <name>), (table-lookup <name>), (can-solve-relation <param> <name>): Both solve-equation and table-lookup goals spawn and-subgoals for finding the necessary parameters. The name in can-solve-relation can either be the name of an equation or the name of a table lookup necessary to find the parameter in the statement. When the necessary parameters are found, solve-equation can mark the parameter as *can-find* and the table-lookup adds a *nvalue* statement.

(subgoal-of <goal> <subgoal>): Subgoal-of is used in the triggers of plans. One of the places this is used is to catch opportunities where multiple goals share a same subgoal and solving that subgoal without making a commitment to either method is advantageous.

(add <statement>), (add-when <condition> <statement>): Add and add-when statements are used to add a new fact. For example, in Figure 39, the phase of the thermodynamic stuff is added after comparing the values obtained from the tables.

(bridge-equation <name>), (frame-equation <name>), (internal-equation <name>): Equation types are used as parts of the triggers for plans. Equation types

are discussed in detail in Chapter 6 along with algebraic reasoning techniques.

5.4 Domain Abstractions

In this section, we discuss how domain abstractions are used as control knowledge in thermodynamics. Control reasoning is an additional layer of domain knowledge that is tightly coupled with the primitives of the domain to describe how to use the primitives effectively. For example, all cycle devices are variations of one input one output devices and many equations can be applied at the more abstract level. Similarly, knowing specific-volume or specific-energy of a thermodynamic stuff is equivalent as both parameters can be used to pinpoint the phase of the thermodynamic stuff. While some domain abstractions, such as the device abstraction, are made explicit in the domain or obvious even to beginners, other abstractions represent a more subtle part of the expert knowledge. This aspect of knowledge-rich domains differentiates them from knowledge poor domains, where abstractions are rarely required.

We call a thermodynamic stuff *sussed*¹ when there is enough information about the thermodynamic stuff to calculate the state parameters. Since a thermodynamic stuff can have up to 21 parameters, and because these parameters are highly connected, this abstraction is especially useful. Once a thermodynamic stuff is marked sussed, its

¹ sussed is a slang used primarily in British English to mean figured out. This word migrated to our vocabulary due to collaboration with Peter Whalley who is a thermodynamics professor at Oxford.

parameters can be marked as known and their calculation can be delayed until the numerical values are needed.

To understand the importance of this abstraction, we need to take a closer look at the structure of equations and sets of equations in thermodynamics. The parameters of a thermodynamic stuff can be divided into two categories: intensive properties independent of mass and extensive properties that vary directly with mass.

extensive parameters	intensive Parameter
volume (V)	specific-volume (spec-v)
enthalpy (H)	specific-enthalpy (spec-h)
entropy (S)	specific-entropy (spec-s)
internal-energy(U)	specific-internal-energy (spec-u)

For each intensive parameter there is an equation involving mass that allows us to calculate the corresponding extensive parameter (Figure 41).

```

volume = spec-v * mass
enthalpy = spec-h * mass
entropy = spec-s * mass
internal-energy = spec-u * mass

```

Figure 41: Relating intensive and extensive properties

Furthermore there are additional equations that relate these parameters to each other (Figure 42) as well as dryness equations (Figure 43).

```

spec-h = spec-u + (pressure * spec-v)
enthalpy = internal-energy + (pressure * volume)
pressure * volume = mass * R * temperature
pressure * spec-v = R * temperature

```

Figure 42: Additional internal equations

```

spec-v = spec-vf + (dryness * (spec-vg - spec-vf))
spec-u = spec-uf + (dryness * (spec-ug - spec-uf))
spec-s = spec-sf + (dryness * (spec-sg - spec-sf))
spec-h = spec-hf + (dryness * (spec-hg - spec-hf))

```

Figure 43: Equations with intensive properties

When two parameters of a thermodynamic stuff is known, the thermodynamic stuff can be marked as sussed. For a sussed stuff, the parameters are marked as known and only calculated at the end if necessary. This abstraction helps the problem solver not get tangled in the large set of equations given above.

5.5 Choosing and Executing Plans

Plans are instantiated when their trigger conditions are believed. The trigger conditions of a plan can include domain primitives as well as conditions about the state of problem solving, such as what the current goal is and what subgoals are being considered as candidates. Plans whose goal conditions are believed are not considered as candidates since executing those plans will not result in any new facts being derived. Plans that have matching index clauses are considered to be candidates for execution. Our domain knowledge is organized to minimize the number of candidate plans. When there are multiple candidate plans, plans that have not been tried before and shorter

plans are preferred.

A plan can introduce a series of methods for achieving the same goal, such as the plan in Figure 44. Methods with context clauses that are believed in are given preference in these cases. When there are multiple equally likely methods, the order they are listed is used to choose among them.

Each step of a plan has a set of test statements which must be satisfied before proceeding to the next stage. Because the test statements can bind further variables, an instantiated plan can have unbound variables in its steps that have not been executed. TPS uses the suggestions architecture (Forbus & de Kleer, 1993) where different components generate suggestions to solve the current goal and the controller chooses among suggestions. Plans are treated as another suggestion type and given priority over other primitive suggestions such as equation solving and table-lookups. When a plan is chosen to be executed it is treated as a special case of an ordered-subgoal that can still bind new variables at each step.

```
(defplan fix-state-using-saturation-table
  :index (fix-state ?stuff)
  :triggers ((basic-thermodynamic-stuff ?stuff)
              (substance-of ?stuff ?sub)
              (saturation-table-for ?sub ?table))
  :succeed ((phase-of ?stuff ?phase))
  :methods
  ((T-spec-v-lookup
    :context ((can-find (T ?stuff)))
    :steps ((t0 (T-spec-v-lookup ?stuff ?table)))))

  ...
  (P-spec-h-lookup
    :context ((can-find (P ?stuff)))
    :steps ((t0 (P-spec-h-lookup ?stuff ?table)))))))
```

Figure 44: Plan introducing multiple methods

Plan steps serve multiple purposes. They introduce a new goal to be achieved, name a plan to be executed or add new knowledge using domain primitives. Addition of new knowledge enables model-fragments to become active and provide suggestions for solving the current goal.

5.6 Chapter Summary

In this chapter we have

- Demonstrated how plans can be used to provide a layer of abstraction closely tied to the domain primitives
- Provided a predicate vocabulary to support engineering problem solving
- Showed how RAPS can be used to represent engineering problem solving.

In the next chapter, we describe how algebraic reasoning is used in our problem solver.

6. Algebraic Knowledge

There are two kinds of algebraic knowledge: First, mathematical knowledge that allows one to manipulate equations to combine similar terms and isolate unknowns. This knowledge falls into the domain of mathematicians. Engineers need this kind of knowledge as well, but they are more concerned with the second kind of algebraic knowledge: formulating a consistent model describing the phenomenon. Once a model is created mathematical knowledge and domain specific knowledge can be used to efficiently navigating through the network of equations to find the values for parameters.

Bundy (Bundy & Welham, 1979) suggested that what differentiates an expert mathematician from a novice is that the expert has access to an implicit set of methods which they draw on when simplifying equations. Bundy proposed a set of algebraic manipulations for simplifying equations based on the structure of the equation. When solving problems in an engineering domain such as thermodynamics, students have to combine and simplify multiple equations to find an answer. We take the classical problem solver (CPS) described in (Forbus & de Kleer, 1993) which implements Bundy's implicit methods for solving single equations and extend it to multiple equations by classifying equations based on the structure of the domain.

To bridge the gap between expert and computer solutions, we need to identify and organize the algebraic knowledge an expert brings into problem solving. For solving

single equations, Bundy (Bundy & Welham, 1979) automated the selection of methods. We have identified three categories of equations to guide problem solving and developed a method for automatically classifying them. In the next section, we describe the functional roles of equations, show how equations can be automatically classified and provide evidence that our methodology can be extended to other engineering domains. In sections 6.2 and 6.3, we describe two additional knowledge-based strategies: providing a preferred direction for solving equations, and using symmetrical substitution to combine equations. In section 6.4, we describe opportunistic strategies that take further advantage of the structure of equations.

6.1 Functional Roles of Equations

The classification of an equation is a domain dependent task. Unlike Bundy's implicit methods for equation simplification, our equation classification is not dependent on the structure of equations. Instead it depends on the role of the equation in the domain. Experts do not treat all equations the same. The equation classifications developed by experts are internalized to the point that finding the right set of equations to work with has been considered an art form (de Kleer, 1975; de Kleer & Sussman, 1980). Expert intuition about equations is based on how equations are frequently used in the domain. We have identified three classes of equations to be useful in guiding problem solving: *internal equations*, *bridge equations* and *frame equations*. This vocabulary for classifying equations enables us to both talk about what kind of equation is needed for

a particular task as well as build additional structures based on equation classification.

We discuss each of the equation classes in turn and give examples of each.

6.1.1 Internal Equations

Internal equations are equations that are composed of variables pertaining to only one object. The number of internal equations is dependent on how complicated the objects are and the number of parameters used in modeling objects. In thermodynamics, objects tend to be complex, with up to 21 parameters, which results in over 30 internal equations relating these parameters. In physics, objects are more idealized and have fewer parameters, resulting in fewer internal equations. The equations shown in Figure 45 are examples of internal equations relating parameters of a thermodynamic stuff to each other.

The parameters of a thermodynamic stuff are related to each other through equations and empirical tables. For example, the dryness of a saturated stuff is a function of its mass in gas phase compared to the total mass. The equations that can be used for calculating dryness are given in Figure 45. The equations given in Figure 45 are not independent from each other. If equation substitution were allowed within this group, combining these equations with each other would create situations where all variables would cancel out. This redundancy in equations is necessary to avoid deriving all equations from first principles for each problem.

Internal equations make it possible to find the numerical values of objects that are not initially given, which are then used by bridge equations to provide additional information to other objects.

```

dryness = mass-of-gas / mass

spec-h = (spec-hf * (1 - dryness)) + (spec-hg * dryness)
spec-u = (spec-uf * (1 - dryness)) + (spec-ug * dryness)
spec-v = (spec-vf * (1 - dryness)) + (spec-vg * dryness)
spec-s = (spec-sf * (1 - dryness)) + (spec-sg * dryness)

spec-h = spec-hf + (dryness * spec-hfg)
spec-u = spec-uf + (dryness * spec-ufg)
spec-v = spec-vf + (dryness * spec-vfg)
spec-s = spec-sf + (dryness * spec-sfg)

```

Figure 45: Equations that can be used for calculating dryness

6.1.2 Bridge Equations

Bridge equations are equations that involve parameters of at least two and possibly more objects. Bridge equations determine the interaction between objects and provide the means for transferring information between them. Separating bridge equations from internal equations allows us to choose equations depending on what we are trying to achieve. For example, when solving a problem about a closed cycle, internal equations provide additional parameter values and bridge equations define the types of processes connecting stuffs. Examples of bridge equations are given in Figure 46.

Thermodynamic Stuff in a Piston: Pressure	Pressure-of-stuff = Atmospheric-pressure
Steady-State Device (ex. Turbine):	mass-in = mass-out
A device modeled as adiabatic:	Temperature-in = Temperature-out

Figure 46: Bridge equations

Bridge equations are equations that connect objects or states together. An object in the system can be arbitrarily complex. We could choose to model the speed of the turbine blades when we model the turbine or keep it as simple as possible. In all cases, the device interacts with the outside world through a set of contact points or a set of equations. These equations enable the problem solver to extract information from one system that can be used by another system. The importance of bridge equations changes depending on how detailed a model we have. The more detailed our model of processes, the more equations we have about the internal parameters of the device. Although the number of bridge equations can also increase with a more detailed model, this increase is significantly less than the increase in internal equations.

Different domains make different demands on bridge equations. In physics, the objects are relatively simple, and can usually be defined with one or two equations. In these cases the number of internal equations is rather small and either bridge equations or frame equations gain more importance. In thermodynamics, the objects are complex involving many parameters which are related through internal equations. Thermodynamics is very well suited for making use of bridge equations since the number of bridge equations compared to internal equations is small.

6.1.3 Frame Equations

Frame equations are the central equations of the domain and serve as starting points for solving problems. General laws and theorems in the domain, which are applicable in most circumstances, are considered frame equations. By providing a starting point, frame equations offer a framework for solving the problem. The first law of thermodynamics, shown in Figure 47, is one of the frame equations for analyzing systems. Although the first law is applicable to all situations, the general form is very rarely used. Instead, modeling assumptions are used to create a simplified version. Thermodynamics textbooks advise students to write down the version of the first law that is applicable to the current situation as a first step in solving a problem.

$Q_{c.v.} + \sum m_i(h_i + V_i^2/2 + gZ_i) = \sum m_e(h_e + V_e^2/2 + gZ_e) + (m_2(h_2 + V_2^2/2 + gZ_2) - m_1(h_1 + V_1^2/2 + gZ_1)) + W_{c.v.}$			
Legend for Variables:		Subscripts Used:	
	m	mass	c.v.
	h	enthalpy	i
	V	velocity	e
state	g	gravity	1
	Z	height	2
state	Q	Heat	
	W	Work	

Figure 47: First law of thermodynamics

6.1.4 Automating Equation Classification

Although our initial intuition was that equation classification has to be done by experts, it is possible to automate this classification to a large extent. We present a set of heuristics that can be used for automated classification below

- Equations that refer to a single object and connect different parameters of the object to each other are internal equations. For example, an equation relating specific volume to mass and volume of object would be an internal equation. Internal equations can also contain constants such as gravity.
- Equations that relate compositional quantities are frame equations. The equilibrium equation where the set of forces is summed and the energy equation describing the total energy of the system fall in this category.
- Equations relating the parameters of two or more objects are bridge equations. For example, the equation stating that the volume of thermodynamic stuff in a container is equal to the volume of the container would be a bridge equation connecting two objects.

We have found a few cases where these criteria are not sufficient. For example, of the two forms of the ideal gas law shown in Figure 48 the first one would be labeled as an internal equation because it involves a universal constant and the second one a bridge equation because it relates gas constant of a specific substance to the parameters

of the thermodynamic stuff. In these cases, expert knowledge about the use of these equations is required.

$PV = \text{moles } R^* T$	R^* is the universal gas constant
$PV = \text{mass } R T$	R is the gas constant of the substance

Figure 48: Two forms of the ideal gas law

6.2 Directional Preferences

Directional preferences reflect how an equation is typically used in problem solving.

Although an equation can be solved for any one of its parameters in theory, in practice equations tend to be solved for one variable more often than others. The equation given in Figure 49 is the definition for constant-pressure specific heat (cp). This equation is often used for finding specific heat and almost never used for finding the gamma value of the substance. (Gamma is either known from tables or derived from other equations that contain only a single occurrence of the term.)

Control conditions for equations, in terms of preference statements, provide directionality to equations. An equation can still be used to solve for any of its variables, but marking an equation's directionality of typical use explicitly helps guide problem solving. The *defEquation* form shown in Figure 49 is similar to the *defModelFragment* form described in section 3.1. It takes two additional keywords: *use for* and *not-use-for*. Both keywords take a list of parameters that are used as preferences when the equation is being considered. If an equation is active for the

current goal, and it is the only equation that has directionality for the current variable then the equation will be the first one to be tried.

```
(defEquation cp=R/gamma*gamma-1
  :participants ((substance ?sub))
  :conditions ((ideal-gas ?sub))
  :not-use-for ((gamma ?sub))
  :use-for ((cp ?sub))
  :form (:= (cp ?sub) (/ (* (R ?sub) (gamma ?sub))
                           (- (gamma ?sub) 1))))
```

Figure 49: An equation with preferences for direction of solving

It is not possible to create the perfect set of preferences that will prevent the problem solver from ever making the wrong choices. Even experts do not solve every problem with a perfect plan, without backtracking, every time. Directional preferences provide guidance, but do not eliminate possibilities, so TPS can still solve a problem that does not fit its plans through backtracking.

6.3 Symmetrical Substitution

Another form of equation simplification is substituting another equation in place of a variable. Since solving one equation to find the unknown variable that is used in another equation is the same as substituting, the problem solver uses equation substitution when multiple substitutions can be made into one equation. For example, the first law has initial and final kinetic energy on both sides of the equation. Substituting $mv^2/2$ into both sides of the equation simplifies the problem. This substitution can let us eliminate “m” from both sides of the equation in some circumstances. Symmetric substitution is performed when the same parameter is found

more than once within an equation.

Having a vocabulary for classifying equations allows us to develop heuristics, which exploit equation classifications to achieve more expert-like performance. Symmetry has been proposed as one of the essential elements of human reasoning. Symmetry has been used to explain new phenomena as well as develop both mental and physical models (Ishida, 1995; Leyton, 1992; Weyl, 1952). The symmetrical substitution heuristic states that substituting different instantiations of the same equation for the initial and final state of the thermodynamics stuff is advantageous when simplifying frame equations. For example, in Figure 50 choosing to substitute $KE = mv^2/2$ for both KE_i and KE_e makes it possible to eliminate mass in the subsequent steps. Although it is possible to find an equation other than $KE = mv^2/2$, using the same equation maintains the symmetry of the equation through simplifications.

Simplified version of Frame Equation The First Law of Thermodynamics

$$Q_{c.v.} + H_i + K_{e_i} + P_{e_i} = H_e + K_{e_e} + P_{e_e} + W_{c.v.}$$

Symmetrical Subst. using Internal Equation $KE = mV^2/2$ for K_{e_i} and K_{e_e}

$$Q_{c.v.} + H_i + m_i V_i^2 / 2 + PE = H_e + m_e V_e^2 / 2 + PE + W_{c.v.}$$

Symmetrical Subst. using Internal Equation $PE = mgZ$ for P_{e_i} and P_{e_e}

$$Q_{c.v.} + H_i + m_i V_i^2 / 2 + m_i g Z_i = H_e + m_e V_e^2 / 2 + m_e g Z_e + W_{c.v.}$$

Symmetrical Subst. using Internal Equation $H = mh$ for H_i and H_e

$$Q_{c.v.} + m_i h_i + m_i V_i^2 / 2 + m_i g Z_i = m_e h_e + m_e V_e^2 / 2 + m_e g Z_e + W_{c.v.}$$

Simplifying using Bridge Equation $m_i = m_e$ and $Q = qm$

$$q_{c.v.} + h_i + V_i^2 / 2 + g Z_i = h_e + V_e^2 / 2 + g Z_e + w_{c.v.}$$

Figure 50: Use of symmetrical substitution

For any variable, there are many equations that can be substituted in place of that variable. The abundance of choices is the reason why novices flounder, and why computer programs often need to try many choices before finding the right one. Figure 50 demonstrates how a frame equation can be used as a starting point and symmetrical substitution (using bridge equations) can be used to simplify the equation. The final equation derived in Figure 50 is commonly used as the per unit mass form of the first law.

Symmetrical substitution method is not domain specific and is applicable to all domains that use equations.

6.4 Opportunistic Strategies

Opportunistic strategies are domain independent strategies that take advantage of the

structure of equations. All equations represent numerical relations among parameters that are applicable under a given set of assumptions. Opportunistic strategies do not take advantage of domain knowledge; therefore, they are used exclusively when no other applicable knowledge is available.

6.4.1 Common Unknown Variables

When there are multiple equations that can be used to solve for the current unknown variable, the decision to choose one equation from the set can be delayed by choosing an unknown variable that is common to a group of equations. This delay in commitment to a specific equation leaves choices open, while at the same time allowing us to work on a subgoal that will be eventually used as a part of the answer. For example, the equation family given in Figure 45 has the common parameter, dryness. As a result, finding the value of dryness first proves to be the most efficient way to proceed.

6.4.2 Common Assumptions

Just as equations share common parameters, families of equations often share common assumptions that need to be satisfied before an equation can be used. For example, equations in Figure 45 can only be used when the thermodynamic stuff is in the saturated phase. If there are active equations, TPS tries to solve the problem using those equations. If this attempt fails then TPS tries to introduce new equations into the analysis by proving additional assumptions about the system. To figure out what

assumptions to look for, TPS considers the instantiated but not yet applicable set of equations. In this way, TPS locates an appropriate assumption that maximizes the number of new equations.

6.4.3 Combining Equations

Although linearly solving one equation at a time is sufficient for solving most engineering problems, sometimes it is necessary to combine equations with similar unknowns. This substitution allows canceling of intermediate variables whose values cannot be determined. When combining equations, extra care must be taken to prevent substituting an equation into one that is derived from that equation itself. This error would result in a new equation in which all variables cancel out. To avoid this problem TPS keeps track of the set of equations that have been used in deriving the new equation, and does not allow equations in the same family to be combined. Figure 51 shows enthalpy and the specific enthalpy equation. Since the second equation can be derived from the first equation, the two equations should not be combined.

$H = U + P V$ $\text{spec-h} = u + P \text{ spec-v}$ <p>The second equation can be derived from the first using</p> $\text{spec-h} = H / \text{mass}$ $\text{spec-u} = U / \text{mass}$ $\text{spec-v} = V / \text{mass}$
--

Figure 51: Equations that should not be combined

6.5 Chapter Summary

Our contributions to algebraic reasoning can be summarized as follows:

- Identifying functional classes of equations
- Providing a criteria to semi-automatically classify equations
- Introducing equation directionality preferences to provide further guidance in problem solving
- Identifying the symmetrical substitution strategy
- Incorporating multiple opportunistic strategies into problem solving

7. Diagrams in Problem Solving

Understanding diagrams is an important part of human cognition, requiring integration of perceptual information and conceptual knowledge. One aspect of problem solving that is often not addressed is how diagrams are used in solving problems. In this chapter we present a cognitively inspired model of graph understanding and demonstrate how diagrammatic interpretation is used in the IPSA architecture.

A quick glance through any textbook will show that diagrams are widely used to give explanations, summarize information and represent spatial relations. For example, we have examined how diagrams are used in (Howell & Buckius, 1992). There are 9 graphs in the 25 solved examples given in the first four chapters. When we look at the solution book, we find that 20% of the solutions have associated graphs. Although problems do not always ask for a graph to be drawn, the prominence of graphs in the solution book emphasizes their importance. Another area where graphs are used is in the problem description, although these occur less frequently.

Within the IPSA architecture, we use diagrams in three different ways. First, diagrams are taken in as a part of the input to the system. TPS extracts numerical values and qualitative relations from input diagrams. Secondly, diagrams are used as part of the output of system. Problems that require drawing a diagram as part of the solution help in building intuitions. We believe IPSA's ability to use diagrams in a similar manner will facilitate integration of IPSA into a teaching architecture. Finally,

diagrams are also used to verify numerical results throughout the problem solving process. Often, students catch their numerical errors through diagrams that are incongruous with their understanding. IPSA uses diagrams for input and output. Because IPSA does not make numerical errors, graphical verification is not used while solving problems, but only used when generating explanations.

In this chapter, we describe a computational model of how line graphs are used in problem solving. We provide a model at the level of information processing capabilities. We then demonstrate the model on diagrams from multiple domains and illustrate how diagrammatic interpretation is integrated into IPSA. The chapter concludes with some examples that make extensive use of diagrams.

7.1 A Model of Graph Understanding

Diagram comprehension requires being able to identify objects, determine the relevant features for a particular problem and interpret the graphical features in terms of the domain. Object recognition is one of the major problems in understanding a diagram, but developing a general algorithm for recognizing objects is well beyond the scope of this thesis. Our model focuses on understanding line graphs, which is a subset of understanding diagrams. Engineering domains make extensive use of line graphs.

The diagrammatic reasoning module is based on an earlier system called SKETCHY (Pisan 1995; Pisan 1994). SKETCHY can provide natural summaries,

answer questions, perform comparative analyses, and detect contradictions in problem solving assumptions. SKETCHY has been fully tested on 65 graphs from two domains (economics and engineering thermodynamics), which suggests that the model is robust.

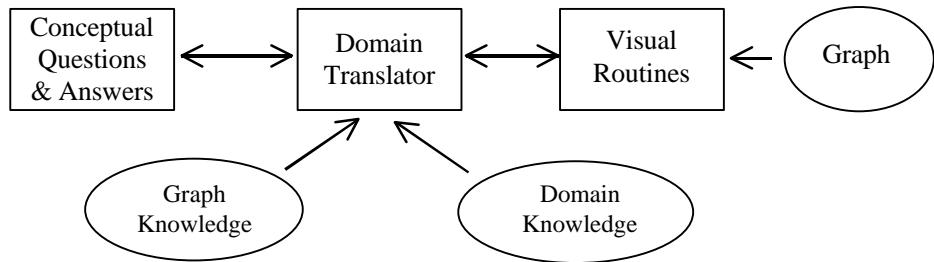


Figure 52: Architecture for graph understanding

Figure 52 shows the architecture for graph understanding. Conceptual questions are constructed using a domain-specific vocabulary. The *domain translator* uses general graph knowledge and domain specific knowledge to convert the questions into graphical relations. *Visual routines* take graphical relationships as their input, inspect the graph to gather the necessary information and return the information to the domain translator. Depending on the results, the domain translator might initiate other visual routines to answer the question. When all the necessary information is obtained from the graph, the domain translator converts the graphical relationships into the vocabulary of the domain and generates an answer to the question. We ignore how visual routines are realized in humans as this problem is being addressed by other researchers (e.g., Ullman, 1984; Chapman, 1991; Horswill, 1995; Ullman, 1984).

7.2 Domain Translator

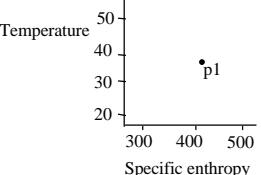
Translating a conceptual question into a form that can be answered using a diagram requires knowing how domain terms are expressed via spatial relations in the graph. For example, to answer the question “When is SUPPLY equal to DEMAND?” the domain translator first needs to identify what objects are being referred to by SUPPLY and DEMAND. Once the appropriate lines are identified, their intersection point can be easily found. Both an isothermal process in thermodynamics and an inelastic supply curve in economics are represented using a horizontal line. The domain knowledge connects the domain concepts with the spatial relations.

Consistent use of graph conventions across domains makes interpreting graphs in a new domain easier. For example, when there are no scales on the axes, lines going up and to the right are interpreted as having a positive slope and signifying that variables on the axes are qualitatively proportional to each other. Steeper lines are interpreted as showing relations where the variable represented on the Y-axis is increasing faster. Two regions with equal areas are interpreted as being equal in magnitude. Although domain specific knowledge can override graph conventions, graphs in most domains follow graph conventions closely. As a result, general graph knowledge can be applied to new domains to produce reasonable graph interpretations even when there is very little or no conceptual domain knowledge.

To extract information from a diagram, first the visual routines identify entities and important relations in the diagram. Once a set of graphical relationships is identified, the domain translator converts them into conceptual terms meaningful for the domain using labels present in the graph. The set of available visual routines is described below.

7.3 Visual Routines

After the conceptual question is translated into graphical terms by the translator, visual routines are invoked to gather the necessary information from the graph. Ullman (1984) suggests how psychologically plausible elemental operations (such as bounded activation and boundary tracing) can be combined to construct visual routines. Visual routines are used to retrieve coordinates of objects, determine spatial orientations, find about interactions, and get information about size and changes in the graph.

<u>Visual Routine</u>	<u>Graph</u>	<u>Conceptual Result</u>
Examine-label		referring to point as p1 as given in problem statement

Coordinate-at-point	<p>A Temperature-Specific Entropy chart with Temperature on the vertical axis (20 to 50) and Specific Entropy on the horizontal axis (300 to 500). Two points are plotted: p_1 at approximately (350, 35) and p_2 at approximately (450, 35).</p>	Temperature of p_1 is 35 and specific entropy of p_1 is 420
Right-of, Left-of, above, below	<p>A Temperature-Specific Entropy chart with Temperature on the vertical axis (20 to 50) and Specific Entropy on the horizontal axis (300 to 500). Two points are plotted: p_1 at approximately (350, 35) and p_2 at approximately (400, 30).</p>	specific entropy of p_1 is greater than specific entropy of p_2 (ordinal relation)
Inside Outside	<p>A Temperature-Specific Entropy chart with Temperature on the vertical axis (20 to 50) and Specific Entropy on the horizontal axis (300 to 500). A saturation dome curve is shown, with a point p_1 plotted inside it. The curve is labeled "sat line".</p>	p_1 is in saturated phase since it lies inside the saturation line when the endpoints are connected. The object represented by p_1 is in saturated phase.
Steeper, flatter	<p>A Temperature-Specific Entropy chart with Temperature on the vertical axis (20 to 50) and Specific Entropy on the horizontal axis (300 to 500). Two curves are shown: l_1 (steeper) and l_2 (flatter), both starting from the same point on the entropy axis.</p>	Temperature of l_1 is increasing faster than l_2 .
Bigger, smaller	<p>A Temperature-Specific Entropy chart with Temperature on the vertical axis (20 to 50) and Specific Entropy on the horizontal axis (300 to 500). Two curves are shown: l_1 (steeper) and l_2 (flatter). The area under l_1 is labeled a_1 and the area under l_2 is labeled a_2. A saturation dome curve is also shown.</p>	a_1 is smaller than a_2 . The amount of work represented by a_1 is less than the work represented by a_2 .

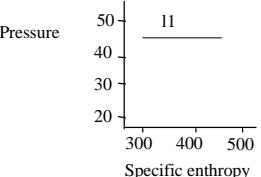
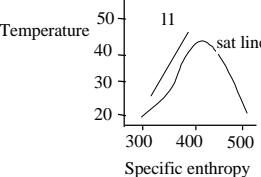
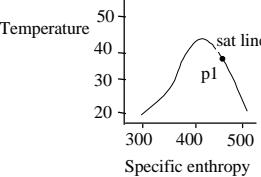
Vertical, horizontal	Pressure 	11 is horizontal. Pressure is constant throughout the process. The process is adiabatic.
Touches, intersects	Temperature 	11 does not touch the saturation line. For the process represented by 11, saturated phase does not have to be considered.
On-line, on- border, forms- border	Temperature 	p1 is on the saturation line. p1 is a saturated gas.

Table 1: Examples of how visual routines are used

7.4 Graph Summary

Figure 53 shows a graph from a thermodynamics textbook. Although this particular graph is not directly used in solving a problem, variations of this graph are often used to demonstrate the phase of the thermodynamic stuff. Grasping the concepts represented in this graph (i.e. the shape of the saturation curve, shapes of the isotherms) is essential for solving many thermodynamics problems. The graph shows three regions (liquid, liquid/vapor, and vapor regions) that correspond to the phase(s) a substance can be in. The temperature lines, which are contours of equal temperature, effectively add a third dimension to the graph. Even in the absence of concrete

problem solving context, TPS can produce graph summaries and allow students to ask questions about the graph. Figure 54 provides an example graph summary.

We have found that using general graph knowledge is sufficient to produce adequate graph summaries. We expect that some domains will require additional domain-specific heuristics. The set of general graph heuristics necessary for graph summaries is given below:

- Only include information for objects with labels.
- Include coordinates of labeled points if the axes have scales.
- If a point is on a line or on the border of an area, include this information.
- Include information about any qualitative changes in line slopes
- Describe each qualitative region separately.
- If lines intersect, include this in the graph description (representing equality)
- Mention changes due to modifications.

Domain-specific vocabulary, such as using isobaric instead of constant pressure and isochoric instead of constant volume further augments the graph summary, bringing it closer to an expert's explanation.

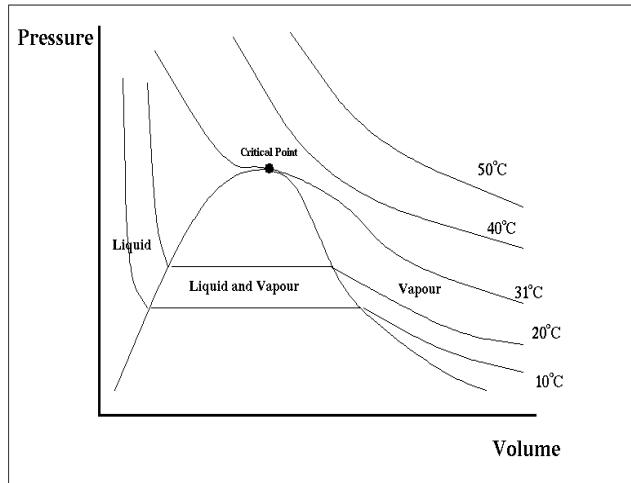


Figure 53: Compression of carbon dioxide

For line 31-C:
VOLUME and PRESSURE are inversely proportional.

For line 20-C:
The slope of 20-C has discontinuities
; associating discontinuities with regions
Inside region LIQUID:
VOLUME INCREASE and PRESSURE DECREASE.
Inside region LIQUID-AND-VAPOUR:
VOLUME INCREASE and PRESSURE CONSTANT.
Inside region VAPOUR:
VOLUME INCREASE and PRESSURE DECREASE.
CRITICAL-POINT is on lines (31-C)
CRITICAL-POINT is on regions (LIQUID LIQUID-AND-VAPOUR
VAPOUR)

For TEMPERATURE contour:
As TEMPERATURE increases
the slopes of TEMPERATURE lines become more LINEAR.
; basis for Boyle's Law

For a constant PRESSURE:
As VOLUME increases TEMPERATURE INCREASE.
VOLUME and TEMPERATURE are directly proportional.

For a constant VOLUME:
As PRESSURE increases TEMPERATURE INCREASE.
PRESSURE and TEMPERATURE are directly proportional.

Figure 54: TPS' description of carbon dioxide compression graph

7.5 Comparative Diagrammatic Analysis

Graphs provide a natural way of performing comparative analysis (Weld, 1990) by combining qualitative and quantitative information. Comparative analysis is the task of predicting how a system will react to perturbations of its parameters. Purely qualitative techniques for comparative analysis, such as the methods used by Weld, are limited in their prediction capacity because the net effect of opposing influences cannot be determined.

Graphs are an ideal representation for comparative analysis because they combine qualitative and quantitative information in a concise format that is ideally suited to our visual processing system. The basic visual processes that are used in graph summaries are also sufficient to generate comparative diagrammatic analyses.

Analyzing engineering cycles is an important task in thermodynamics. The basic cycle for a steam power plant is the Rankine cycle, shown in Figure 56. A common modification to the Rankine cycle is superheating the steam in the boiler to increase the efficiency of the cycle. The net work of the cycle before modification is represented by area 1-2-3-4-1 and after modification by 1-2-3'-4'-1. The area under 1-2-3-3' represents the total heat put into the system.

```

; using graph interpretation rules
For point 3:
The ENTROPY of 3 INCREASE.
The TEMPERATURE of 3 INCREASE.
For point 4:
The ENTROPY of 4 INCREASE.
The TEMPERATURE of 4 CONSTANT.
For region WORK
..The area covered by WORK INCREASE.
For region HEAT:
The area covered by HEAT INCREASE.
; using thermodynamics knowledge
For variable EFFICIENCY:
EFFICIENCY has INCREASE.

```

Figure 55: TPS' explanation

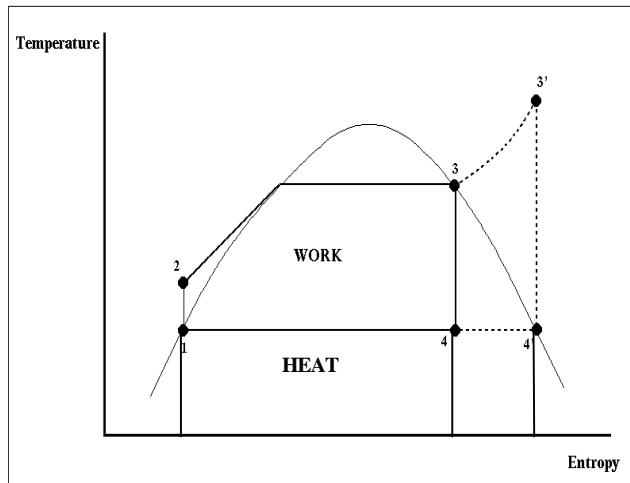


Figure 56: Effect of superheating on Rankine cycle

Qualitative methods alone are sufficient to reach the conclusion that WORK and HEAT have increased as a result of modification. Efficiency, defined as the amount of work divided by the amount of heat, is represented indirectly through work and heat as areas in the graph. Determining whether efficiency has increased or not cannot be

resolved qualitatively. TPS uses visual routines to calculate the changes in areas and determine that the efficiency of the cycle is increased. When numerical values are available, a more accurate computation of the change in efficiency is made possible by equations.

7.6 Diagrams as Input

We show the problem description taken from (Howell & Buckius, 1992) in Figure 57. TPS combines the information from the graph with its representation of the problem description shown in Figure 58. The statements added to the problem description through graph analysis are shown in Figure 59.

4.73 “Oxygen in a system undergoes the process shown on the P - v diagram. Find the work done. Is it work done on or by the system?”

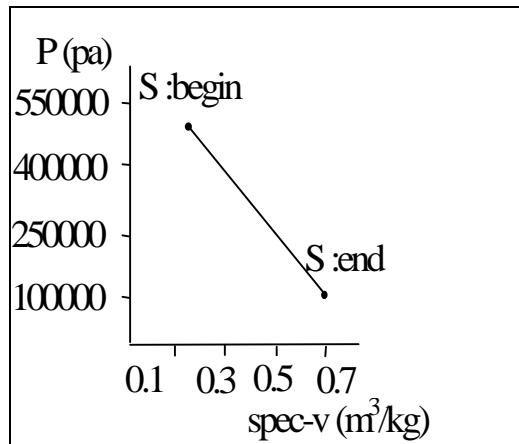


Figure 57: Problem description with graph as input

```
(add-problem :hb4.73
:givens '((static-thermodynamic-stuff (at s :begin))
  (substance-of (at s :begin) oxygen)
  (process cmp (at s :begin) (at s :end)
  (given-graph P-v-4.73)))
:goal '(find (nvalue (spec-work cmp))))
:answer '(175 kJ/kg))
```

Figure 58: TPS' representation for problem 4.73

```
(nvalue (P (at s :begin)) 500000 Pa)
(nvalue (spec-v (at s :begin)) 0.2 m^3/kg)
(nvalue (P (at s :end)) 200000 Pa)
(nvalue (spec-v (at s :end)) 0.7 m^3/kg)
(> (P (at s :begin) (P (at s :end)))
(< (spec-v (at s :begin)) (spec-v (at s :end)))
```

Figure 59: Statements added through graph analysis

Because the line shown in Figure 57 is not labeled, TPS does not extract any information about the process represented by the line. Once the data from the graph is added, the solution to the problem is relatively easy.

```
Found (SPEC-WORK CMP) = 175000.0
using
  SPEC-WORK=PDV-LINEAR
  (:= (SPEC-WORK CMP)
    (* (* (+ (P (AT S :BEGIN)) (P (AT S :END))) 0.5)
      (- (SPEC-V (AT S :END)) (SPEC-V (AT S
:BEGIN))))))
on
  (P (AT S :BEGIN)) = 500000
  (SPEC-V (AT S :BEGIN)) = 0.2
  (P (AT S :END)) = 200000
  (SPEC-V (AT S :END)) = 0.7
```

Figure 60: TPS' solution for problem 4.73

7.7 Diagrams as Output

We show the problem description taken from (Howell & Buckius, 1992) in Figure 61.

The problem requires students to draw the graph first to underscore the importance of finding the phase of the thermodynamic substance as an initial problem solving step.

As clearly seen in Figure 63, the thermodynamic stuff is in the gas phase throughout the process.

- 4.3S A piston-cylinder arrangement initially contains water at $P = 2 \text{ MPa}$ and $T = 300^\circ\text{C}$. The piston is moved to a final position where the volume is twice the initial volume. During the piston movement, there is heat transfer to the water in such a way that the pressure in the cylinder remains constant.
- (a) Sketch the process on a P - v diagram, indicating the relative position of the saturation curve.
 - (b) Find the work done per kilogram of water during the expansion.
 - (c) Find the heat transfer per kilogram of water during the expansion.
 - (d) Find the value of the enthalpy of the water after expansion

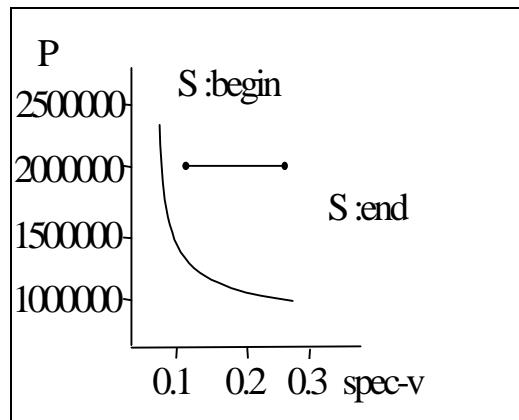
Figure 61: Problem requiring graph output

```

(add-problem-description :hb4.03
  :givens '((piston (at can :begin))
    (static-thermodynamic-stuff (at S :begin))
    (inside (at S :begin) (at can :begin))
    (direction (at can :begin) :up)
    (substance-of (at s :begin) water)
    (nvalue (P (at s :begin)) 2 MPa)
    (nvalue (T (at s :begin)) 300 C)
    (expanding expn (at s :begin) (at s :end))
    (isobaric expn)
    ;; volume is doubled
    (given-equation volume-doubled
      (volume-doubled
        (:= (V (at s :end)) (* 2 (V (at s
:begin)))))))
    (given-equation constant-mass-in-container
      (:= (SPEC-V (AT S :END))
        (* 2 (SPEC-V (AT S :BEGIN)))))))

(add-problem-goal :hb4.03a
  :description :hb4.03
  :goal '(plot-graph (P spec-v) expn))

```

Figure 62: TPS' description for problem 4.03**Figure 63: Graph produced by TPS for problem 4.03**

Only part of the graph is shown in Figure 63 because the process compared to the P-spec-v table is very small.

7.8 Diagrams for Result Verification

The most frequent use of diagrams is for result verification. In general, graphical information is more easily assimilated by students than numeric data or text-based problem statements. In thermodynamics, representing the process on a T-s diagram has the additional advantage of emphasizing how much the given process deviates from an ideal one. We provide the problem description in Figure 64, TPS' description in Figure 65, and the graph produced by TPS in Figure 66.

3.9S Indicate whether the following states for water are in the liquid, saturation, or superheated region. Specify the quality of the states in the saturation region.

State	P, kPa	$T, {}^\circ\text{C}$	$\nu, \text{m}^3/\text{kg}$
1	1700	200	—
2	1200	—	0.0010
3	—	75	3.0
4	500	202	—
5	350	—	0.005

Figure 64: Problem where graph is used for confirmation

```
(add-problem-description :hb3.09
  :givens '((basic-thermodynamic-stuff (at S :begin))
             (substance-of (at S :begin) water)))

(add-problem-goal :hb3.09a
  :description :hb3.09
  :extras '((nvalue (P (at S :begin)) 1700 kPa)
             (nvalue (T (at S :begin)) 200 C)))
  :goal '(find (phase-of (at s :begin))))
  :answer '(liquid (at s :begin)))
```

Figure 65: TPS' description for problem 3.09

Although T-s diagrams are usually preferred, in this example because pressure and temperature is already provided it is necessary to draw the P-T graph first. Once the P-T graph is drawn, the phase of the thermodynamic stuff can be calculated and further graphs can be produced if necessary.

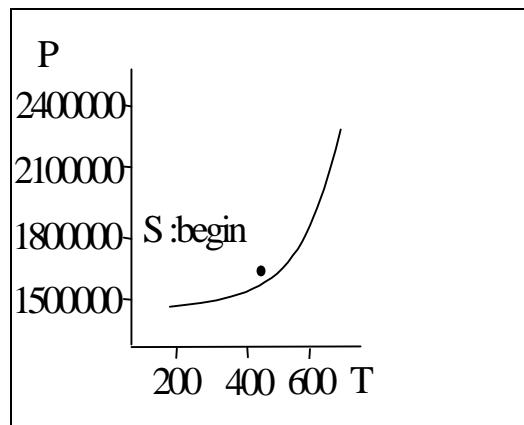


Figure 66: Graph produced by TPS as part of explanation

7.9 Chapter Summary

Diagrams are an integral aspect of problem solving. They are used to illustrate spatial relations, explain problems, and summarize information. TPS uses diagrams as part of the problem description, as answers to graphing questions and for verifying results if necessary. Our main contribution to diagrammatic reasoning is demonstrating how a cognitively plausible set of visual routines can be used to summarize graphs, perform comparative analysis and use graphs in the context of problem solving. In the next chapter, we provide expanded examples of how TPS solves problems.

8. Evaluation

TPS solves over 150 thermodynamics problems taken from the first four chapters of a common thermodynamics textbook, as well as problems from other textbooks, and produces expert-like solutions. In this chapter, we examine how the textbook and the problem set for TPS was chosen to ensure broad coverage and provide an example comparing TPS' solution to expert solution.

8.1.1 Choosing a Balanced Set of Problems

In knowledge poor domains new problems can often be generated using automatically to test the performance of the problem solver. For knowledge-rich domains like thermodynamics, testing the system on automatically generated problems would not lead to meaningful results. Such automated problems would not capture the complexity of the domain, but would concentrate on a set of equations or a specific problem type. Choosing a balanced set of problems for testing a knowledge-rich problem solver is critical. We discuss three important issues in choosing a problem set:

- Finding a large number of problems
- Balancing variations among textbooks
- Having a representative sample of problems

To find a large number of problems, we examined a number of textbooks. Textbook problems provide reasonable approximations for the types of analyses performed by practitioners in the field. We examined a number of different textbooks including as (Howell & Buckius, 1992; Moran & Shapiro, 1995; Whalley, 1992; Wylen & Sonntag, 1985). Although different textbooks cover the same conceptual material, different textbooks also reflect the authors' style of teaching. We chose (Howell & Buckius, 1992) because it has more problems than any of the other textbooks. In fact, the textbook by Howell and Buckius has more problems than the other 3 textbooks combined (Moran & Shapiro, 1995; Whalley, 1992; Wylen & Sonntag, 1985). Having a large sample set ensures that the problem solver domain knowledge is not tailored to specific question type or to a specific author. The number of problems in each chapter is shown in Figure 67.

	Number of Problems	Percentage of Total
Chapter 2	79	23%
Chapter 3	59	18%
Chapter 4	199	59%
Total	337	100%

Figure 67: Distribution of problems by chapter

A similarity that most textbooks share in the organization of their problems is that even numbered problems and odd numbered problems only differ in some changes in numerical values. The answers to odd numbered problems is usually provided, so students can use problems to study and instructors can assign even numbered problems

as homework. We choose to limit our problem solver to odd numbered problems both for convenience and because including even numbered problems would not have contributed any new problem types or situations.

Having a large set of problems balances ensures coverage of the domain. We have chosen to exclude three categories of problems to partially balance the variations among textbooks and to keep the focus of our domain model on the common part of thermodynamics rather than the interactions between thermodynamics and other domains. The categories we have excluded are as follows:

- **Geometrical reasoning:** Knowledge of geometry does not overlap thermodynamics knowledge except in advanced thermodynamics when fluid mechanics is considered. As each author has their own style, Howell and Buckius have included a large number of problems involving slanted tubes. In all these problems, the concept of pressure equilibrium is explored. We felt it was justified to exclude this group of questions because none of the other thermodynamics textbooks elaborate on this topic. Representations that would have been required for these problems are at best tangential to thermodynamics knowledge.
- **Thermodynamics Arcaneries:** Problems that are used to demonstrate the limits of the current models or limits of thermodynamics are excluded from our

test set. These problems usually require introducing a unique perspective that is only used for the current problem. For example, a series of problems demonstrates that the van der Waals equation provides more accurate results than the ideal gas equation by introducing two additional constants. Although this equation is more accurate, the simple ideal gas equation is used for all other problems. These examples help students understand that all of the models used are approximations and more detailed models can be used when necessary.

- **Uncommon devices and substances:** We have chosen not to model some of the uncommon devices and substances that appear infrequently. These devices are introduced to capture students interest and to make connections with other engineering domains. Examples of such devices are manometers, electrical batteries and stretching rods. Uncommon substances, such as ethanol, are also not represented in our domain model. Our domain model has thermodynamic tables for methane, nitrogen, ammonia, refrigerant-12, water and ideal air, which is sufficient for representing the necessary concepts and most of the problems.

The number of problems excluded are tabulated by type in Figure 68.

	Problems excluded due to geometry	Problems excluded due to uncommon devices	Problems excluded due to thermodynamic arcanery

Chapter 2	16	7	1
Chapter 3	0	4	0
Chapter 4	2	1	1
Total	18	12	2

Figure 68: Excluded Problems

Statistics on problems attempted and solved by TPS are tabulated in Figure 69.

	Odd Problems with parts	Problems Excluded	Problems attempted by TPS	Problems solved by TPS	Percentage of Problems solved
Chapter 2	45	24	21	18	86%
Chapter 3	59	4	55	46	84%
Chapter 4	120	4	116	81	70%
Total	224	32	192	145	76%

Figure 69: Problems attempted and solved by TPS

TPS is able to solve 76% of the problems successfully. The problems that TPS cannot solve can be grouped as follows:

- Problems requiring integrals and complex symbolic algebraic simplifications cannot be solved by TPS. The equation solver used by TPS is not complete and in cases where numerical values are not available and 2 or more equations have to be combined to eliminate some intermediary variables, TPS can fail to simplify the equation.

- Most problems that introduce a new quantity can be handled by providing an equation in the problem statement; however, if the new defined quantity is representing a difference between two states, such as the change in volume, then solving the problem requires converting equations in the domain by subtracting them from each other. Identifying when such difference equations are needed and deciding which equations should be combined to create the necessary difference equations requires additional algebraic knowledge which TPS does not have.
- Problems that involve three dimensional graphs cannot be handled by TPS. For example, compressibility charts are used to represent reduced pressure, reduced temperature and compressibility factor, essentially projecting three dimensional information to a line graph. Three dimensional graphs are not very common as people often find them hard to read as well. In fact, the compressibility chart is the only three dimensional chart encountered in thermodynamics.
- Although our domain model uses compositional quantities, all values are not defined as compositional quantities. For example, the energy of an object is a compositional quantity composed of its kinetic, internal and potential energy. If there was another energy source, it could be easily added. The work of a process however is defined as the work done by the thermodynamics stuff, so

for problems where an outside object, such as a fan, is introduced the domain representation is insufficient to capture this phenomenon.

- TPS' domain model is focused on thermodynamics. Abstract problems, such as what happens to the temperature if the air-condition is turned off in a room with 20 people, or problems that combine every day knowledge with thermodynamics is beyond TPS' capabilities.

Extending TPS to solve 100% of the problems represents a number of challenges, the biggest one of which would be representing people's common sense knowledge so it can be tied to thermodynamics.

8.2 Comparing to Expert Solutions

When comparing TPS' solutions to expert solutions, we focus on which equations are chosen and used in the solution. An example natural language problem description and its corresponding TPS representation are shown in Figure 70.

Problem Description:

A storage tank contains air at the same temperature as the surroundings (35C). Suddenly it begins to rain, and the surrounding temperature drops to 20C. The tank was initially at 200kPa, and the tank volume is 10m³. (a). What is the final pressure in kilopascals in the tank (b) How much heat transfer in kilopascals occurred to or from the air during cooling?

```
(add-problem-description :hb4.15
  :givens '((container (at can :begin))
    (closed (at can :begin))
    (rigid (at can :begin))
    (static-thermodynamic-stuff (at S :begin))
    (inside (at S :begin) (at can :begin))
    (substance-of (at s :begin) AIR)
    (nvalue (T (at S :begin)) 35 C)
    (nvalue (P (at S :begin)) 200 kPa)
    (nvalue (V (at can :begin)) 10 m^3)
    (cooling cool (at s :begin) (at s :end))
    (nvalue (T (at S :end)) 20 C)))

(add-problem-goal :hb4.15a :description :hb4.15
  :goal '(find (nvalue (P (at s :end)))))
  :answer '(190.3 kPa))

(add-problem-goal :hb4.15b :description :hb4.15
  :goal '(find (nvalue (Q cool))))
  :answer '(-243.7 kj))
```

Figure 70: Natural language and TPS' representation of a problem

Comparing the expert solution shown in Figure 71 to TPS' solution in Figure 72, we can see that the main difference is that TPS' solution explicitly states the equalities between the container and the thermodynamic stuff inside it. Because this equality is seen as part of general knowledge it is excluded from the experts solution. Although TPS' solution looks significantly longer than the expert solution, it should be noted that this is due to how parameters are represented, and that mathematical notation used by people is a lot more compact than computer representations, rather than due to a difference in the number of steps taken to get to the solution.

Expert Solution:

$$(PV/T)_1 = (PV/T)_2$$

;; converting from Centigrade to Kelvin and substituting values in equation

$$P_2 = P_1 * T_2 / T_1 = 200 ((273 + 20) / (273 + 35)) = 190.3 \text{ kPa}$$

Figure 71: Expert Solution to part a

```

;; the equalities between the volume of the can and the thermodynamic
stuff inside it is not made explicit in expert's solution
Found (V (AT CAN :END)) = 10
using equality
(:= (V (AT CAN :BEGIN)) (V (AT CAN :END)))
Found (V (AT S :END)) = 10
using equality
(:= (V (AT CAN :END)) (V (AT S :END)))
Found (V (AT S :BEGIN)) = 10
using equality
(:= (V (AT CAN :BEGIN)) (V (AT S :BEGIN)))
;; TPS uses the same equation found in the expert solution
Found (P (AT S :END)) = 190264.481583644
using
PV/T=P2V2/T2
(:= (/ (* (P (AT S :BEGIN)) (V (AT S :BEGIN))) (T (AT S :BEGIN)))
(/ (* (P (AT S :END)) (V (AT S :END))) (T (AT S :END))))
```

Figure 72: TPS' solution to part a

The second part of this problem is slightly more complex. The expert solution and TPS' solution are shown in Figure 73 and Figure 74 respectively. It should be noted that both TPS' solution and the expert solution use the same version of ideal gas law to find the mass and the conservation of energy equation to calculate the value for heat.

$$Q = m(u_2 - u_1) = (PV / RT)_1 * (u_2 - u_1)$$

$$u_2 - u_1 = c_v * (T_2 - T_1)$$

$$u_2 = 195.46 + 0.8(213.4 - 195.46) = 209.81 \text{ kJ/kg}$$

$$u_1 = 213.40 + 0.4(231.36 - 213.40) = 220.58 \text{ kJ/kg}$$

$$m = (PV / RT)_1 = 200000 / (0.2870 * (273 + 35)) = 22.63 \text{ kg}$$

$$\text{so } Q = -22.63 * (220.58 - 209.81) = -243.7 \text{ kJ}$$

Figure 73: Expert solution to part b

```

Found (MASS (AT S :END)) = 22.6144446372954
using
IDEAL-GAS-LAW
(:= (* (P (AT S :END)) (V (AT S :END)))
      (* (MASS (AT S :END)) (R AIR) (T (AT S :END))))
Found (SPEC-U (AT S :END)) = 210041.975
using
IDEAL-GAS-INTERNAL-ENERGY
(:= (SPEC-U (AT S :END)) (* (CV AIR) (T (AT S :END))))
Found (U (AT S :END)) = 4749982.61514568
using
SPEC-U-DEFINITION
(:= (SPEC-U (AT S :END)) (/ (U (AT S :END)) (MASS (AT S :END))))
Found (MASS (AT S :BEGIN)) = 22.6144446372954
using
IDEAL-GAS-LAW
(:= (* (P (AT S :BEGIN)) (V (AT S :BEGIN)))
      (* (MASS (AT S :BEGIN)) (R AIR) (T (AT S :BEGIN))))
Found (SPEC-U (AT S :BEGIN)) = 220789.475
using
IDEAL-GAS-INTERNAL-ENERGY
(:= (SPEC-U (AT S :BEGIN)) (* (CV AIR) (T (AT S :BEGIN))))
on
(T (AT S :BEGIN)) = 308.15
(CV AIR) = 716.5
Found (U (AT S :BEGIN)) = 4993031.35888502
using
SPEC-U-DEFINITION
(:= (SPEC-U (AT S :BEGIN)) (/ (U (AT S :BEGIN)) (MASS (AT S :BEGIN))))
Found (Q COOL) = -243048.743739333
using
CONSERVATION-OF-ENERGY
(:= (+ (Q COOL) (ENERGY-BEGIN COOL) (ENERGY-INLET COOL))
      (+ (ENERGY-END COOL) (ENERGY-OUTLET COOL) (WORK COOL)))

```

Figure 74: TPS' solution to part b

There are two criteria that we look at when we compare solutions:

- Number of equations used
- The form of the equation chosen
- The order of the equations in the problem solution

In general, the number of equations used by experts is a subset of the equations used by TPS. The reason for this is that experts leave out equations and inferences. For example, an expert ignores whether it was the volume of the container or volume of the thermodynamic stuff that was stated in the problem description ; however, TPS is sensitive to subtle differences in the problem statement. Another factor that affects the number of equations is that the solution by the expert is tailored to the expected audience. Equations that are spelled out in earlier chapters can be skipped in later ones as the learner is expected to infer the simpler equations. Tailoring the output in this manner requires a detailed model of the learner, which is not part of TPS.

The first law of thermodynamics and a number of other equations have multiple forms which can be chosen when solving a problem. While TPS is bound by the equations in its domain model, an expert can more flexibly create new forms of these equations on the fly and use them in the solution. Because thermodynamics equations are rather complex, experts in general tend to stick to versions of equations that have been described in the chapter, but in other domains where the number of parameters per equation is less we would expect to see more equation construction on the fly.

Both experts and TPS is bound in general terms in the order they can find new values or solve equations. However, in many cases there is not a preferential order between two subgoals. For example, whether the volume of the thermodynamic stuff is calculated using the container volume or whether the area of the piston is calculated

using the diameter does not change the essence of the solution in any way and different experts are likely to alternate between solving one or the other first.

We have tabulated TPS' solution steps versus expert's solution steps in Figure 75. It can be observed from this table that as problems get more complex, the difference between the length of TPS solution and expert solution increases. There are two major contributors to this effect. First, experts start skipping steps more to keep the solutions concise and because they believe students should be able to infer intermediary stages as problems get more complex. Second, complex problems often involve the first law of thermodynamics which uses compositional quantities stating that the total energy remains constant. While TPS examines each member of this set (i.e., potential energy, kinetic energy) and ignores it as a separate step, the expert often simplifies the equation in one step.

	TPS Steps	Expert Steps	Percentage extra
Chapter 2	43	34	26%
Chapter 3	98	59	66%
Chapter 4	488	209	133%
Total	629	302	108%

Figure 75: TPS steps versus expert steps

The primary reasons that TPS' solution was longer was as follows:

- Equality equations that were necessary, but not shown in expert solution

- Table-lookups that were not shown as explicit steps

We did not find any significant difference in the order of the steps between TPS solutions and expert solutions. There were a few differences where a different form of the equations was used by TPS. For example, the expert solutions tended to prefer the first equation shown in Figure 76 and TPS would sometimes use the second equation shown in Figure 76. In all of these cases, mass was constant and known so the choice of equation did not effect the solution.

```
specific-work = Pressure * (specific-volume - specific-volume)
Work = Pressure * (Volume - Volume)
Work = specific-work * mass
```

Figure 76: Work equations

There were a few cases where TPS' solution was shorter than expert solution. This was due to the fact that TPS performs table-lookups as atomic operations. Problems where expert solution displayed the interpolation was longer.

8.3 Chapter Summary

TPS finds solutions that are similar to expert solutions. We believe that the more verbose solutions produced by TPS would be especially appreciated by students who are not feeling confident in inferring the missing steps. We show that TPS can solve over 75% of the problems from the first four chapters of a thermodynamics textbook proving itself to be a wide-covergae problem solver.

9. Related Work

The IPSA architecture differs from previous work in the way it brings together four different sets of skills and applies them to problem solving. We review the literature in problem solving, qualitative reasoning, planning, algebraic reasoning, and diagrammatic reasoning in the next sections and identify the contributions of this work to each of those areas.

9.1 Engineering Problem Solving and Planning

Previous architectures for engineering problem solving have only worked in a limited their range of problems. The relative complexity of the thermodynamics domain has forced us to explicitly represent dynamic entities such as processes and actions, and reason about them explicitly while solving problems. In the following sections, we describe previous work on problem solving in complex domains, planning systems that have been extended as problem solvers, and the development of abstract plans.

9.1.1 Problem Solving in Complex Domains

Research in human problem solving has focused on tasks that do not require prior task-related knowledge. One of the few studies that examine problem solving in a knowledge-rich domain is (Bhaskar & Simon, 1977). This study argued that the subject used means-ends analysis in solving chemical engineering thermodynamics problems. The broader applicability of this research is limited because it studied only

one subject's problem solving skills. Other research (Larkin et al., 1980; Priest, 1986; Priest & Lindsay, 1992) indicates that expert behavior in knowledge-rich domains is significantly more complex and diversified.

Physics problem solving was one of the first areas in which problem solvers were tested, because it offers a uniform set of assumptions and a relatively simple set of equations. One of the early problem solvers, MECHO, was motivated by the desire to understand how to represent real world situations using a mathematical model (Bundy et. al. 1976). ISAAC is another problem solver that solved static and equilibrium physics problems (Novak, 1976). The problems ISAAC solved involved unambiguous situations. As a result, a series of algebraic manipulations sufficed to find the correct answer. CASCADE (VanLehn, Jones, & Chi, 1991) solved physics problems using analogy. The lack of a mechanism for reasoning about changes limited CASCADE to a small number of static problems (the number of problems CASCADE solved is not indicated in the papers). Another early system that offered a flexible architecture was Fermi (Larkin, Reif, Carbonell, & Gugliotta, 1985). Although Fermi solved approximately 25 problems from three different domains (a subset of fluid statics, DC circuits, and computations of centers of mass), its depth of knowledge in any of these domains was limited. In comparison, TPS has solved over 150 problems and provides models spanning four chapters of a common thermodynamics textbook.

9.1.2 Problem Solving as Planning

Planners that search the space of possible plans using backward chaining can be divided into those that use "eager-commitment" vs. those that employ "delayed-commitment" strategies. Systems that use delayed-commitment include SNLP, NOAH, NONLIN, and TWEAK (McDermott, 1978; Sacerdoti, 1977), (Chapman, 1987; McAllester & Rosenblitt, 1991; Tate, 1977) (Wilkins, 1984). In delayed-commitment, the ordering of operators is postponed. Ambros-Igerson (Ambros-Igerson & Steel, 1988) introduced a framework to interleave planning and execution to be able to deal with unexpected events.

On the other hand, there are a number of prominent systems that use a strategy of "eager-commitment". The benefit of this approach is the ability of the problem solver to reason from a continuously updated internal state. GPS (Ernst & Newell, 1969), STRIPS (Fikes & Nillson, 1971) , PRODIGY(Carbonell, Knoblock, & Minton, 1990; Veloso & Carbonell, 1988) and SOAR (Laird, Rosenbloom, & Newell, 1986; Rosenbloom & Laird, 1986) use eager-commitment when searching through possible states. FLECS (Veloso & Stone, 1995) allows the explicit choice of operator-ordering commitment strategy. Guptha (Guptha & Nau, 1992) shows that even in a simplified domain such as the Blocks World, finding an optimal plan is NP-hard.

One way to circumvent the difficulties of planning is through machine- or human-generated abstract plans that aim to capture the typical activities in that domain. A

method for automatically creating abstract plans is to drop sentences from domain descriptions. Several systems have been developed that use this method. ALPINE (Knoblock, 1991; Knoblock, 1994; Knoblock, Minton, & Etzioni, 1991; Koedinger & Anderson, 1990) automatically learns hierarchies of abstraction spaces by dropping sentences. The PARIS system (Bergmann & Wilke, 1995), based on skeletal plans (Friedland & Iwasaki, 1985), goes a step further. The development of abstraction beyond sentence dropping broadens the scope of how states can be abstracted into its domain model. However, it has been tested primarily in toy domains, such as the Towers of Hanoi (Simon, 1975).

In a reactive environment such as a robot acting in the physical world, plans have to be flexible to adapt to dynamic situations. Reactive action packages, called RAPS (Firby, 1989), are hierarchical building blocks that generate primitive robot actions at execution. TPS' plans are modeled after RAPS. Although in thermodynamic problem solving there is only one active agent, namely the problem solver; the ability to plan at a higher level provides flexibility to the evolving problem solving situation. Because there are no other agents in problem solving, the opportunistic nature of RAPS is only used when there is a suggestion that can lead to the problem goal directly.

Although solving problems based on first principles has a broader range of applicability, as the domain become more complex, the lack of guidance can cause the problem solver to fail. A number of techniques have been used for capturing and

organizing this information. Case-based reasoning (Kambhampati & Hendler, 1992; Kolodner, 1993; Schank, 1982) or analogy (Carbonell, 1986; Veloso & Carbonell, 1988; Falkenhainer, 1990; Gentner & Forbus, 1991; Veloso & Carbonell, 1993; Forbus et al., 1998) have good success rates when the new problems they encounter are similar to stored examples. However, adapting a known solution to a current problem requires a deep understanding of the domain and good matches may not always be available.

Although explanation-based learning (DeJong & Mooney, 1986; Fikes, Hart, & Nilsson, 1972; Ihrig & Kambhampati, 1997; Kambhampati & Kedar, 1994; Minton, 1988; Mitchell, Keller, & Kedar-Cabelli, 1986; Mooney, 1988) has been successfully used to generate the control knowledge for some domains, it has been shown to work only under limited representations (Etzioni, 1993; Minton, 1990).

9.2 Qualitative Reasoning

de Kleer (de Kleer, 1975) was the first to argue that qualitative reasoning is needed for problem solving and to apply qualitative reasoning to classical mechanical problems. Qualitative reasoning has been used successfully for analyzing complex systems (Sacks & Joskowicz, 1993; Yip, 1991), designing a chemical separation system (Sgouros, 1994) and designing controllers (Biswas, Manganaris, & Yu, 1993; Kuipers & Shults, 1994).

TPS uses the Compositional Modeling Language (Falkenhainer et al., 1994) in representing its domain model. CML allows representing assumption classes and entities in a hierarchical manner. We have extended the CML representations to incorporate control information into equations and model fragments.

One of the few systems that attempted to integrate qualitative and quantitative analysis is SCHISM (Skorstad & Forbus, 1990) which used qualitative reasoning to find closed sets of equations. SCHISM analyzed steady-state thermodynamics cycles. The IPSA architecture substantially broadens the scope of this work to include a larger set of thermodynamics problems.

In Chapter 4, we discussed the frame problem, initially described in (McCarthy & Hayes, 1969), and its implication for thermodynamics where both continuous and discontinuous changes must be represented. We address the frame problem using spatially bound temporally extended histories as introduced by (Hayes, 1985). Fikes presents one approach to handling the frame problem in the STRIPS representation (Fikes & Nillson, 1971). In this system, each state is made up of statements describing the current situation that are known to be true. By adding and deleting facts, operators allow transitions from one state to the next. Because each incremental change in a numerical parameter requires the creation of new states, reasoning about continuous change is problematic with the STRIPS model. Although STRIPS provides a convenient representation for representing discontinuous changes, to represent

continuous changes we must reason about processes and limit points (Fikes et al., 1972; Forbus, 1989; Kuipers, 1994; Weld, 1985).

Discontinuous changes are abrupt changes in the environment. A typical example of this is the closing and opening of a valve in thermodynamics. One approach to modeling discontinuous change is through using infinitesimal calculus (Davis, 1989; Weld, 1988) and treating the change as a rapid continuous change. TPS uses GIZMO to reason about continuous changes while keeping its internal representation in the form of time instants, where numerical equations can be applied. Using information on the interval between time instants allows TPS to reason about how long the process remained active. TPS also uses quantitative knowledge to constrain the number of states produced in qualitative analysis.

9.3 Algebraic Reasoning

Bundy (Bundy & Welham, 1979) indicated that experts are different from novices in that expert mathematicians have a set of implicit methods for problem solving. Bundy proposed formulating these implicit methods and teaching them as a part of mathematics knowledge. Formalizing these methods creates an explicit control vocabulary that can be taught to students and used by computer problem solvers to guide equation solving. We have done an analogous analysis using the functional roles equations play in problem solving.

Shavlik and DeJong (Shavlik & DeJong, 1990) examine how to use explanation-based learning in mathematical domains. They focus on generalizing solutions by eliminating variables from equations. The general equation produced can then be applied to a larger set of problems. This approach is useful for examining how knowledge is acquired and generalized. When modeling expert problem solving, starting first from frame equations and then simplifying them is more advantageous to reduce the problem space.

Williams introduced hybrid algebra that allows the combining of qualitative and quantitative algebras (Williams, 1991). Although this new system has significant advantages over qualitative algebra, in the context of problem solving where numerical values are often available, such an extension is not necessary. Decompositional modeling introduced by (Williams & Raiman, 1994) solves nonlinear algebraic systems by dividing the statespace. The focus of our research has been in introductory thermodynamics. For more advanced analysis, decompositional modeling may be required to solve problems.

Having a vocabulary for classifying equation allows us to develop heuristics that exploit equation classifications to achieve more expert-like performance. Just like the symmetrical substitution method described earlier, symmetry has been proposed as one of the essential elements of human reasoning and to explain new phenomena as well as develop both mental and physical models (Ishida, 1995; Leyton, 1992; Weyl, 1952).

9.4 Diagrammatic Reasoning

As Chandrasekaran et al. point out (Chandrasekaran, Narayanan, & Iwasaki, 1993) diagrams are used in a variety of fields including scientific discovery, economic reasoning, architecture, creative and conceptual design, mathematical proofs, biological reasoning, molecular analysis, planning, college physics and procedure instructions. The general spatial reasoning problem is too unconstrained to be analyzed. Research has focused on more constrained problems such as motion in limited domains (de Kleer, 1975; Forbus, 1980) and mechanical mechanisms (Faltings, 1987; Forbus, Nielsen, & Faltings, 1991; Gelsey, 1989; Joskowicz, 1987; Nielsen, 1988). Although diagrams are important tools in planning and problem solving, there has been very little research linking diagrams to knowledge-rich domains.

One program that explored the use of diagrams in problem solving is WHISPER.(Funt, 1980) WHISPER uses perceptual primitives to refer to diagrams during its problem solving. Using center of gravity information, the program determined when objects were not stable and might rotate or slide as a result. WHISPER's domain knowledge is very limited and it has only been tested on few problems to demonstrate diagrammatic reasoning. Yip and Zhao (Yip & Zhao, 1996) present three problem solvers--KAM, MAPS, and HIPAIR-- that produce images from continuous data and perform at expert level in domains where numerical methods are insufficient. Similarly, TPS uses line graphs to find out the phase of a thermodynamic

stuff. Obtaining phase information purely from tables would require more complex numerical analysis.

Ullman (Ullman, 1984) introduced the concept of visual routines as a goal-oriented visual processing facility. Visual routines express domain-specific visual skills. Mahoney (Mahoney, 1992) extends Ullman's work by defining image chunks, formed using topological information, that can be used for higher level goals. In TPS we ignore the problem of recognizing and identifying graph objects and concentrate on interpreting their interactions. One inspiration for TPS' diagrammatic reasoner is the metric diagram/place vocabulary model of spatial reasoning (Forbus, 1980; Forbus et al., 1991). TPS' visual routine processor is its metric diagram.

POLY_A (McDougal & Hammond, 1993) uses visual operators to specify which objects in the diagram to inspect in the course of solving geometry proofs. POLY_A's operators are very specific to the geometry domain (such as LOOK-AT-LEFT-BASE-ANGLE). SONJA (Chapman, 1991) on the other hand uses very general action oriented visual operators for playing a video game. TPS' operators are specific for examining line graphs and not specific to any particular domain.

Pinker (Pinker, 1990) describes psychological factors contributing to difficulty in reading graphs. Pinker suggests an architecture similar to TPS' diagrammatic reasoning module, but his main emphasis is on recognition of different graph types through

general graph schemas and the difficulties in understanding different graphs, rather than providing a concrete computational model for graph interpretation. Gattis and Holyoak (Gattis & Holyoak, 1994) look at the impact of goals and conceptual understanding on graph interpretation. Gattis and Holyoak's most significant finding is that the variable being queried should be assigned to the vertical axis, so that steeper lines can map to faster changes in the queried variable. We view this result as further evidence that graph semantics and graph interpretation is separate from the domain.

Lohse (Lohse, 1993) describes a computer program called UCIE which uses graph schemas to predict response times to answer questions about the graph. UCIE's graph schemas for information retrieval are similar to TPS' general graph knowledge.

Anderson and McCartney (Anderson & McCartney, 1996) propose a case-based diagrammatic solution to the n-queens problem. Although this approach shows how CBR and diagrammatic reasoning can be combined, developing a more general framework would require more substantial analysis and extension of diagrammatic operators.

The REDRAW system presented in (Tessler, Iwasaki, & Law, 1995) reasons in the domain of civil engineering using qualitative analysis. The "structure layer" and the "diagram layer" presented is similar to the use of labels and graphical objects adapted in

TPS. The diagrammatic operators used in REDRAW allow the program to modify the diagram as the design specifications are altered.

TPS' graph descriptions are mainly produced by domain independent graph rules. Tabachneck, Leonardo and Simon (Tabachneck, Leonardo, & Simon, 1994) demonstrate how novices have difficulty integrating visual and verbal information. Novices fail to provide answers that could be obtained by simple perception whereas experts see the answer immediately. When domain rules are not used, TPS suffers from a similar problem. TPS cannot answer any questions about variables besides the ones explicitly mentioned on the graph even when the answer is visually available. Part of becoming an expert in a domain is creating the necessary domain rules, so that inferences about objects not labeled in the graph can be made.

10. Conclusion

IPSA is the first problem solving architecture of which we are aware that combines qualitative, diagrammatic, algebraic reasoning and plans to provide a robust model of engineering problem solving. This work contributes to each of these research areas as well as providing methods about how these different styles of reasoning can be combined in an integrated architecture. We supported our claim by showing that the TPS system, IPSA's application to thermodynamics, is able to solve a large set of problems from a complex and knowledge-rich domain.

In addition to contributing to the understanding and design of engineering problem solvers, this work outlines a set of skills on which engineering instructors should concentrate. Currently, the emphasis in engineering education is on teaching the relevant equations for the domain. As TPS demonstrates, although this knowledge is necessary, expertise and a deeper understanding of the domain can best be developed through explicit instructions in planning and identifying domain abstractions.

We have demonstrated our claims about the IPSA architecture through its implementation in the TPS system. TPS solves over 150 problems taken from several textbooks and covering chapters equivalent to those taught in introductory thermodynamics. TPS' ability to solve multiple classes of problems gives further proof that the IPSA architecture is broadly applicable to engineering problem solving. In

addition to solving a large number of problems, we have shown that TPS' produces solutions similar to those of experts.

10.1 Contributions to Qualitative Reasoning

Qualitative reasoning provides the necessary leverage to represent domain abstractions and enable high level system analysis. We have used qualitative reasoning to achieve two specific goals. First, TPS uses qualitative reasoning to determine which parameters remain constant throughout a process. Finding constant parameters is an essential initial step in system analysis in order to calculate subsequent values and solve the problem. Second, TPS uses qualitative reasoning to determine hidden states that are not mentioned explicitly in the problem description, but are crucial in analyzing the system. The equations applicable throughout hidden states differ from those that are relevant in the initial and final stages defined in the problem statement. Therefore, hidden states must be accounted for explicitly in the process of solving the problem. Although qualitative knowledge has been frequently employed to determine the boundaries for parameters, our approach of interleaving qualitative and quantitative reasoning is unique.

10.2 Contributions to Planning

While equations, assumptions, and ordinal constraints are the primitive building blocks for problem solving, plans are the architecture that combines these basic units. In IPSA, plans provide the bridge between qualitative reasoning, algebraic reasoning, and

diagrammatic reasoning by determining when and how these techniques should be applied. IPSA incorporates a planning vocabulary to model expert strategies.

We show that plans capture expert knowledge and provide a framework for solutions making them closer to expert solutions. A feature that textbooks cannot provide is showing students the range of choices for plans and equations that is available in the current situation. TPS can make this information available to students as well as providing reasons for why a specific choice is made at that step.

10.3 Contributions to Algebraic Reasoning

Algebraic knowledge is at the heart of engineering problem solving because it provides the tools to combine, simplify, and manipulate equations. Our contributions in this area are in identifying functional equation types, expressing domain-specific directionality preferences of equations, and taking advantage of symmetrical substitution to simplify equations. In the first instance, identifying equation types allows TPS to eliminate equations based on higher goals. Equation directionality makes it possible for TPS to prefer one equation over another without explicit global preferences. Symmetrical substitution provides guidance in combining equations in a semantically meaningful manner. All of these strategies promote efficient problem solving. We believe the strategies we discussed would be applicable to all engineering domains where equations play a dominant role.

The ties between algebraic knowledge and domain expertise have largely been ignored due to the view that students already possess the necessary algebraic knowledge. Our work shows how domain knowledge reorganizes what is normally looked upon as purely mathematical manipulations. Making this link explicit in engineering education should allow students to develop expertise more quickly.

10.4 Contributions to Diagrammatic Reasoning

To our knowledge, TPS is the first system that uses diagrammatic reasoning to solve a large set of problems in a knowledge-rich domain. Diagrams are used as input to the system, as a part of the output, and in verifying numerical results. Because people use diagrams frequently in the context of problem solving, any system that attempts to parallel human reasoning must incorporate diagrams at all stages of problem solving. By integrating diagrams into IPSA, we provide a more plausible model of expert problem solving behavior.

In addition to reading and interpreting diagrams, we have also identified a set of visual routines to perform the necessary diagrammatic operations. We believe that IPSA's visual routines can be generalized further from line graphs to other graph types.

10.5 Limitations

The IPSA architecture provides the necessary primitives to express both engineering domain knowledge and control knowledge. It is possible that another domain would require a style of reasoning other than qualitative, algebraic, or diagrammatic

reasoning. In that case, a new module would have to be added to IPSA. However, IPSA currently integrates those modules that we have found so far to be essential to engineering problem solving.

The limitations of TPS can be divided into three categories. First, TPS' domain knowledge is concentrated in the area of thermodynamics. As a result, TPS cannot handle problems that introduce complex geometric configurations or problems that require common sense reasoning outside of thermodynamics.

Secondly, TPS' control knowledge is static. Students will often know what principles to apply based on the chapter from which the problem is taken. We have deliberately not incorporated such heuristics into TPS, as these strategies do not reflect expertise in problem solving so much as mirror the rote-centered problem solving approach to engineering education.

Thirdly, problems that can be represented for TPS in part reflect the modeling choices in the domain model. For example, representing devices that have an arbitrary number of input and output flows is cumbersome but still possible. The conservation of mass equation, which is instantiated using compositional quantities, connects all the inputs and outputs. However, if a second equation is provided relating a specific input and output which helps reduce the conservation of mass equation to a simpler form, the necessary algebraic manipulation is beyond TPS' algebraic reasoning capability.

10.6 Future Work & Possible Extensions

The IPSA architecture provides the necessary framework to support an intelligent learning environment. In building such an environment, pedagogical, implementation, and usability issues would have to be considered. In the sections below, we suggest extensions that require further research.

10.6.1 Qualitative Reasoning

The problem-guided envisionment strategy we have used in IPSA has been successful, but we believe that a closer integration of qualitative reasoning and quantitative reasoning would enhance the fluidity of problem solutions. A further improvement would be to automatically construct qualitative domain theories from equations represented in the domain model. Combining qualitative and quantitative model construction further would facilitate the building of modular, scalable domain theories.

10.6.2 Planning

Our emphasis in this research is on engineering problem solving, and as a result we have approached planning from a practical rather than theoretical perspective. Although automated methods would not have been sufficient to generate TPS' plans, such methods could be used to fine-tune and expand plans entered by experts. The planning language is strongly connected to the underlying problem solving structure. Separating plans from the underlying processes could allow experimentation with multiple planning languages using the same system.

10.6.3 Algebraic Reasoning

We have chosen not to use any of the commercially available products for solving sets of equations because these systems do not reflect expert cognitive processes and remain black boxes for the novice. Developing an algebraic reasoning system with an open architecture would benefit IPSA and enable TPS to solve a larger set of problems. This would allow a more seamless integration of expert reasoning and algebraic reasoning.

10.6.4 Diagrammatic Reasoning

The IPSA architecture reasons with line graphs, the most common type of diagram in engineering. Although bar graphs, pie charts, and other complex diagrammatic representations are not frequently encountered in engineering domains, extending IPSA's diagrammatic reasoning ability would make this architecture applicable beyond engineering. In IPSA, we implement a cognitively plausible model for diagrammatic reasoning. Further research in this area could tease out exactly how people interpret diagrams in the process of problem solving. Another natural extension to IPSA would be to implement image chunks (Mahoney, 1992), which would enable it to analyze scanned images. This extension would not fundamentally alter our model of graph understanding, but would enable a more natural way of entering diagrams to be interpreted.

10.6.5 Analogical Reasoning and Learning

An aspect of problem solving not explored in this research is how past examples help in solving current problems. There is evidence that some students copy solved examples verbatim when attempting new problems in the same chapter and other students use comparisons with other problems to understand problems better. Analogical reasoning could be used to extract chapter-specific plans to apply to similar types of problems (Falkenhainer, 1990; Forbus et al., 1998; Gentner & Forbus, 1991).

The holy grail of problem solving has been finding the perfect learning algorithm that could be used in many domains. Although some learning algorithms have been successful in micro-worlds (DeJong & Mooney, 1986; Minton, 1988; Mitchell, Keller, & Kedar-Cabelli, 1986; Mooney, 1988), their applicability to knowledge-rich domains has been very limited. We would be curious to see what kinds of plans would be generated through such systems. Even if plans have to be entered by experts, learning algorithms could be used to hone plan selection strategies.

10.7 Summary

In conclusion, we believe that IPSA provides a stable architecture for engineering problem solving. We hope that IPSA will prove as useful foundation for cognitive simulation studies of human problem solving, and as a component in intelligent tutoring systems and learning environments for engineering. Making intelligent learning environments available to students should help generate enthusiasm for engineering and thus may contribute to educational goals in the classroom.

References

- Ambros-Igerson, J., & Steel, S. (1988). Integrating planning, execution and monitoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, St Paul, MN, 83-88.
- Anderson, M., & McCartney, R. (1996). Diagrammatic reasoning and cases. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, OR
- Batali, J. (1991). *Automatic acquisition and use of some of the knowledge in physics texts*. Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA.
- Bergmann, R., & Wilke, W. (1995). Building and refining abstract cases by change of representation language. *Journal of Artificial Intelligence Research*, 3, 53-118.
- Bhaskar, R., & Simon, H. A. (1977). Problem solving in Semantically Rich Domains: An Example of Engineering Thermodynamics. *Cognitive Science*, 1, 193-215.
- Biswas, G., Manganaris, S., & Yu, X. (1993). Extending component connection modeling for analyzing complex physical systems. *IEEE Expert*, 8(1), 48-57.
- Bundy, A., Luger, G., Stone, M., & Welham, R. (1976). *MECHO: Year One* (Technical Report D.A.I. Research Paper No. 22). Edinburgh, Scotland: University of Edinburgh.
- Bundy, A., & Welham, B. (1979). Using meta-level inference for selective application of multiple rewrite rules in algebraic manipulation. In *Proceedings of The International Joint Conference on Artificial Intelligence*, 1017-1027.
- Carbonell, J. G. (1986). Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2, pp. 371-392). Los Altos, CA: Morgan Kaufmann.
- Carbonell, J. G., Knoblock, C. A., & Minton, S. (1990). Prodigy: An integrated architecture for planning and learning. In K. VanLehn (Ed.), *Architectures for intelligence*. Hillsdale, NJ: Erlbaum.
- Chandrasekaran, B., Narayanan, N. H., & Iwasaki, Y. (1993, Summer). Reasoning with diagrammatic representations. *AI Magazine*, 14, 49-56.
- Chapman, D. (1987). Planning for conjunctive goals. *Artificial Intelligence*, 32, 333-378.
- Chapman, D. (1991). *Vision, instruction, and action*. Cambridge, MA: MIT Press.

- Davis, E. (1989). Order of Magnitude Reasoning in Qualitative Differential Equations. In D. S. a. d. K. Weld, Johan (Ed.), *Readings in Qualitative Reasoning about Physical Systems* (pp. 422-434). MK:ADR: MK.
- de Kleer, J. (1975). *Qualitative and quantitative knowledge in classical mechanics* (Technical Report 352.). Cambridge, MA.: MIT AI Lab.
- de Kleer, J. (1977). Multiple Representations of Knowledge in a Mechanics Problem Solver. In *Proceedings of The International Joint Conference on Artificial Intelligence*, 299-304.
- de Kleer, J., & Sussman, G. J. (1980). Propagation of constraints applied to circuit synthesis. *Circuit Theory and Applications*, 8, 127-144.
- DeJong, G., & Mooney, R. (1986). Explanation based learning: An alternative view. *Machine Learning*, 1, 145-176.
- Ernst, G., & Newell, A. (1969). *GPS: A case study in generality and problem solving*. New York: Academic Press.
- Etzioni, O. (1993). A structural theory of explanation-based learning. *Artificial Intelligence*, 60(1), 93-139.
- Falkenhainer, B. (1990). A unified approach to explanation and theory formation. In J. W. Shavlik & T. G. Dietterich (Eds.), *Readings in Machine Learning* (pp. 373-403). San Mateo, CA: Kaufmann.
- Falkenhainer, B., Farquhar, A., Bobrow, D., Fikes, R., Forbus, K., Gruber, T., Iwasaki, Y., & Kuipers, B. (1994). *CML: A compositional modeling language* (KSL-94-16): Knowledge Systems Laboratory, Stanford University.
- Falkenhainer, B., & Forbus, K. (1991). Compositional modeling: Finding the right model for the job. *Artificial Intelligence*, 51, 95-143.
- Faltings, B. (1987). Qualitative Kinematics in Mechanisms, *The International Joint Conference on Artificial Intelligence* (pp. 436-442).
- Fikes, R., & Nillson, N. (1971). STRIPS: A New Approach to the application of theorem proving to problem solving. *Artificial Intelligence*(2), 189-208.
- Fikes, R. E., Hart, P. E., & Nilsson, N. J. (1972). Learning and executing generalized robot plans. *Artificial Intelligence*, 3, 251-288.
- Firby, J. (1989). *Adaptive execution in complex dynamic domains*. Ph.D. Thesis, Yale University, New Haven, CT.
- Forbus, K. (1980). Spatial and qualitative aspects of reasoning about motion. In *Proceedings of the First Annual AAAI Conference*, Los Altos, CA, 170-173.
- Forbus, K. (1984a). Qualitative Process Theory. *Artificial Intelligence*, 24(3), 85-168.

- Forbus, K., & de Kleer, J. (1993). *Building Problem Solvers*: MIT Press.
- Forbus, K., Everett, J., Ureel, L., Brokowski, M., Baher, J., & Kuehne, S. (1998). Distributed coaching for an intelligent learning environment. In *Proceedings of the Twelfth International Workshop on Qualitative Reasoning*, Cape Cod, Massachusetts, 57-64.
- Forbus, K., Nielsen, P., & Faltings, B. (1991). Qualitative Spatial Reasoning: The Clock Project. *Artificial Intelligence*, 51(1-3), 417-471.
- Forbus, K. D. (1984b). *Qualitative Process Theory* (Technical Report 789). Cambridge, MA: Massachusetts Institute of Technology.
- Forbus, K. D. (1987). The Logic of Occurrence, *The International Joint Conference on Artificial Intelligence* (pp. 409-415).
- Forbus, K. D. (1989). Introducing actions into qualitative simulation. In *Proceedings of The International Joint Conference on Artificial Intelligence*, Detroit, MI, 1273-1278.
- Friedland, P. E., & Iwasaki, Y. (1985). The concept and implementation of skeletal plans. *Journal of Automated Reasoning*, 1(2), 161-208.
- Funt, B. V. (1980). Problem-Solving with diagrammatic representations. *Artificial Intelligence*, 13, 201-230.
- Gattis, M., & Holyoak, K. J. (1994). How graphs mediate analog and symbolic representation. In *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, Hillsdale, NJ, 346-350.
- Gelsey, A. (1989). Automated Reasoning about Machine Geometry and Kinematics, *The 3rd IEEE Conference on AI Applications* (pp. 182--187).
- Gentner, D., & Forbus, K. (1991). MAC/FAC: A Model of similarity-based retrieval. In *Proceedings of the Cognitive Science Society*
- Gupta, N., & Nau, D. (1992). On the complexity of blocks world planning. *Artificial Intelligence*, 56(2-3), 223-254.
- Hayes, P. J. (1985). The second naive physics manifesto. In J. Hobbs & B. Moore (Eds.), *Formal theories of the commonsense world* (pp. 1-36): Ablex Publishing Corporation.
- Hayes, P. J. (1990). Naive Physics I: Ontology for Liquids. In D. S. Weld & J. D. Kleer (Eds.), *Reading in Qualitative Reasoning About Physical Systems*. San Mateo, CA: Morgan Kauffman Publishers, Inc.

- Horswill, I. (1995). Visual Routines and visual search: a real-time implementation and an automata theoretic analysis. In *Proceedings of International Joint Conference on Artificial Intelligence*, Montreal, Canada, 56-62.
- Howell, J. R., & Buckius, R. O. (1992). *Fundamentals of engineering thermodynamics* (2nd ed.). New York: McGraw-Hill Inc.
- Ihrig, L. H., & Kambhampati, S. (1997). Storing and indexing plan derivations through explanation-based analysis of retrieval failures. *Journal of Artificial Intelligence Research*, 7, 161-198.
- Ishida, Y. (1995). Symmetry-based reasoning about equations of physical laws. In *Proceedings of the Ninth International Workshop on Qualitative Reasoning about Physical Systems*, 84-93.
- Joskowicz, L. (1987). Shape and Function in Mechanical Devices. In *Proceedings of AAAI-87*
- Kambhampati, S., & Hendler, J. A. (1992). A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence*, 55, 193-258.
- Kambhampati, S., & Kedar, S. (1994). A unified framework for explanation-based generalization of partially ordered partially instantiated plans. *Artificial Intelligence*, 67, 29-70.
- Knoblock, C. A. (1991). Search reduction in hierarchical problem solving. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, Anaheim, CA, 686-691.
- Knoblock, C. A. (1994). Automatically generating abstractions for planning. *Artificial Intelligence*, 68, 243-302.
- Knoblock, C. A., Minton, S., & Etzioni, O. (1991). Integrating abstraction and explanation-based learning in Prodigy. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, Anaheim, CA, 541-546.
- Koedinger, K. R., & Anderson, J. R. (1990). Abstract Planning and Perceptual Chunks: Elements of Expertise in Geometry. *Cognitive Science*(14), 511-550.
- Kolodner, J. L. (1993). *Case-based reasoning*: Morgan Kaufmann.
- Kuipers, B. (1994). *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*. Cambridge, MA: MIT Press.
- Kuipers, B. J. (1986). Qualitative simulation. *Artificial Intelligence*(29), 289- 388.
- Kuipers, B. J., & Shults, B. (1994). Reasoning in logic about continuous systems. In *Proceedings of The 8th International Workshop on Qualitative Reasoning about Physical Systems*, Nara, Japan., 164-175.

- Laird, J., Rosenbloom, P., & Newell, A. (1986). Chunking in soar: The anatomy of a general learning mechanism. *Machine Learning*, 1, 11-46.
- Larkin, J., McDermott, J., Simon, D. P., & Simon, H. A. (1980). Models of competence in solving physics problems. *Cognitive Science*, 4(4), 317-345.
- Larkin, J. H., Reif, F., Carbonell, J., & Gugliotta, A. (1985). *Fermi: A flexible expert reasoner with multi-domain inferencing*. Pittsburgh, PA: Carnegie-Mellon University.
- Leyton, M. (1992). *Symmetry, Causality, Mind*: MIT Press.
- Lohse, G. L. (1993). A cognitive model for understanding graphical perception. *Human-Computer Interaction*, 8(4), 353-388.
- Mahoney, J. V. (1992). *Image chunks and their applications* (Technical Report EDL-92-3). Palo Alto, CA.: Xerox Parc.
- McAllester, D., & Rosenblitt, D. (1991). Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 634-639.
- McCarthy, J., & Hayes, P. J. (1969). Some philosophical problems from the stand point of artificial intelligence. In B. Mettzer & D. Michie (Eds.), *Machine Intelligence* (Vol. 4,). Edinburgh: Edinburg University Press.
- McDermott, D. (1978). Planning and acting. *Cognitive Science*, 2(2), 71-109.
- McDougal, T. F., & Hammond, K. J. (1993). Representing and using procedural knowledge to build geometry proofs. In *Proceedings of The Eleventh National Conference on Artificial Intelligence*
- Minton, S. (1988). *Learning search control knowledge: An explanation-based approach*. Boston, MA: Kluwer.
- Minton, S. (1990). Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, 42(2-3), 363-391.
- Mitchell, T., Keller, R., & Kedar-Cabelli, S. (1986). Explanation based generalization: A unifying view. *Machine Learning*, 1, 47-80.
- Mooney, R. J. (1988). Generalizing the order of operators in macro-operators. In *Proceedings of the Fifth International Conference on Machine Learning*, San Mateo, CA, 270-283.
- Moran, M. J., & Shapiro, H. N. (1995). *Fundamentals of engineering thermodynamics* (3rd ed.). New York: John Wiley and Sons, Inc.
- Nielsen, P. E. (1988). A Qualitative Approach to Mechanical Constraint, AAAI88 (pp. 270-274).

- Novak, G. (1976). *Computer Understanding of Physics Problems Stated in Natural Language.* , University of Texas, Austin, TX.
- Pinker, S. (1990). A theory of graph comprehension. In R. Freedle (Ed.), *Artificial intelligence and the future of testing* (pp. 73-126). Hillsdale, NJ: Lawrence Erlbaum Associates.
- Priest, A. (1986). Inference strategies in physics problem solving. In A. G. Cohn & J. R. Thomas (Eds.), *Artificial Intelligence and its Applications* . New York: John Wiley & Sons Inc.
- Priest, A., & Lindsay, R. (1992). New light on novice-expert differences in physics problem solving. *British journal of Psychology*(83), 389-405.
- Rosenbloom, P., & Laird, J. (1986). Mapping explanation-based learning onto SOAR. In *Proceedings of the Third National Conference on Artificial Intelligence*, Philadelphia, PA
- Sacerdoti, E. D. (1977). *A structure for plans and behavior*. New York: American Elsevier.
- Sacks, E., & Joskowicz, L. (1993). Automated modeling and kinematic simulation of mechanisms. *Computer Aided Design*, 25(2), 106-118.
- Schank, R. C. (1982). *Dynamic memory: A theory of learning in computers and people*. New York: Cambridge University Press.
- Sgouros, N. M. (1994). *Representing Physical and Design Knowledge in Innovative Design (Separation Systems, Intelligent Artifacts)*. PH.D., Northwestern University.
- Shavlik, J. W., & DeJong, G. F. (1990). Learning in mathematically-based domains: understanding and generalizing obstacle cancellations. *Artificial Intelligence*, 45, 1-45.
- Simon, H. (1975). The functional equivalence of problem solving skills. *Cognitive Psychology*, 7, 268-288.
- Skorstad, G., & Forbus, K. (1990). Qualitative and quantitative reasoning about thermodynamics. In *Proceedings of The 10th Annual Conference of the Cognitive Science Society*
- Tabachneck, H., Leonardo, A. M., & Simon, H. A. (1994). How does an expert use a graph? A model of visual and verbal inferencing in economics. In *Proceedings of the Sixteenth Annual conference of the Cognitive Science Society*, Hillsdale, NJ, 842-847.

- Tadepalli, P., & Natarajan, B. K. (1996). A formal framework for speedup learning from problems and solutions. *Journal of Artificial Intelligence Research*, 4, 445-475.
- Tate, A. (1977). Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, 888-900.
- Tessler, S., Iwasaki, Y., & Law, K. (1995). Qualitative structural analysis using diagrammatic reasoning. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*
- Ullman, S. (1984). Visual routines. *Cognition*, 18, 97-159.
- VanLehn, K., Jones, R. M., & Chi, M. T. H. (1991). Modeling the self-explanation effect with Cascade 3. In *Proceedings of The Thirteenth Annual Conference of the Cognitive Science Society*, Hillsdale, NJ, 137-142.
- Veloso, M., & Stone, P. (1995). FLECS: Planning with a flexible commitment strategy. *Journal of Artificial Intelligence Research*, 3, 25-52.
- Veloso, M. M., & Carbonell, J. G. (1988). Integrating derivational analogy into a general problem solving architecture. In *Proceedings of the First Workshop on Case-Based Reasoning*
- Veloso, M. M., & Carbonell, J. G. (1993). Towards scaling up machine learning: A case study with derivational analogy in Prodigy. In S. Minton (Ed.), *Machine learning methods for planning* (pp. 233-272): Morgan Kaufmann.
- Weld, D. S. (1985). Combining Discrete and Continuous Process Models, *The International Joint Conference on Artificial Intelligence* (pp. 140-143).
- Weld, D. S. (1988). Comparative Analysis. *AIJ*, 36, 333-374.
- Weyl, H. (1952). *Symmetry*: Princeton University Press.
- Whalley, P. B. (1992). *Basic Engineering Thermodynamics*. New York: Oxford University Press.
- Wilkins, D. E. (1984). Domain-independent planning: Representation and plan generation. *Artificial Intelligence*, 22, 269-301.
- Williams, B. C. (1991). Theory of interactions. Unifying qualitative and quantitative algebraic reasoning. *Artificial Intelligence*, 51, 39-94.
- Williams, B. C., & Raiman, O. (1994). Decompositional modeling through caricatural reasoning. In *Proceedings of Twelfth National Conference on Artificial Intelligence*, Seattle, Washington
- Wylen, G. J. V., & Sonntag, R. E. (1985). *Fundamentals of Classical Thermodynamics* (3rd ed.): John Wiley & Sons Inc.

- Yip, K. (1991). *KAM: A system for Intelligently Guiding Numerical Experimentation by Computer*. Cambridge, MA: MIT Press.
- Yip, K., & Zhao, F. (1996). Spatial aggregation: Theory and applications. *Journal of Artificial Intelligence Research*, 5, 1-26.

