NORTHWESTERN UNIVERSITY

Topological Inference of Teleology: Deriving Function from Structure via Evidential Reasoning

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

By

John Otis Everett

EVANSTON, ILLINOIS

December 1997

ABSTRACT

Topological Inference of Teleology: Deriving Function from Structure via Evidential Reasoning

John Otis Everett

Reasoning about the physical world is a central human cognitive activity. One aspect of such reasoning is the inference of function from the structure of the artifacts one encounters. In this dissertation we present the Topological iNference of Teleology (TNT) theory, an efficient means of inferring function from structure. TNT comprises a representation language for structure and function that enables the construction, extension, and maintenance of the domain-specific knowledge base required for such inferences, and an evidential reasoning algorithm. This reasoning algorithm trades deductive soundness for efficiency and flexibility. We discuss the representations and algorithm in depth and present an implementation of TNT, in a system called CARNOT. CARNOT demonstrates quadratic performance and broad coverage of the domain of single-substance thermodynamic cycles, including all such cycles presented in a standard text on the subject. We conclude with a discussion of CARNOT-based coaching tools that we have implemented as part of our publicly-available CyclePad system, which is a design-based learning environment for thermodynamics.

ACKNOWLEDGMENTS

I would like to thank the following people for their contributions to the creation of a rich environment for learning and thinking over the past five years:

First and foremost, my advisor and friend Ken Forbus, for providing the guidance, encouragement, and unending stream of computing equipment that sustained this line of research. Larry Birnbaum and Ian Horswill, my committee members, for their support and feedback. Roger Schank, for founding a first-rate interdisciplinary institute in my backyard. Chris Riesbeck, for introducing me to the one true language. Chip Cleary, for introducing me to the Institute for the Learning Sciences and not laughing at my initial misconceptions concerning LISP and AI. Greg and Heather Collins, for those dinners out. Keith Law, for bringing his titanium-body camera to the hospital on a moment's notice. Ron Ferguson and Yusuf Pisan, my fellow travelers, for their help and kibitzing along the way. Andy Bachmann, the embodiment of an incredibly sensitive difference detector, for his hacking enthusiasm. John Demastri, for putting up with a certain amount of LISP imperialism and C++ bashing. Leo Ureel, Mike Brokowski, and Julie Baher for their commitment to making CyclePad work. Eric Goldstein, Chris Wisdo, and Jon Handler, for the Dissertation Avoidance Support Group. Sandor Szego, for his unflagging optimism and energy. David Foster, for being able to converse on topics other than ILS after hours. Brian Smith, for providing ongoing live entertainment during our tenure at the Institute.

Beyond ILS, I would also like to thank Francis and Eleanor Everett, my parents, for being supportive in the face of my departure from their known universe. Jack and Marilyn Bergers, my in-laws, for their patience, especially since they thought I was done when I passed the qualifying exam three years ago. And finally, Jeannine, my wife, for encouraging me to do what makes me happy, for putting up with five years of graduate school, and for giving me the most amazing self-organizing learning system, our son Jonah.

CONTENTS

ILLUSTRATIONS

TABLES

1 INTRODUCTION

The inference of function from structure is an essential cognitive skill for anyone who works with designed systems. Designers must understand the interplay between structure and function to create effective designs, engineers must be able to determine the function of a system if they are to verify its behavior or diagnose problems, and students must learn to make such inferences to bridge their learning from theory to practice.

Designed systems are typically represented via schematics or blueprints, which in and of themselves are incomplete; those who use these documents must bring knowledge about the function of the relevant components to bear in the process of understanding the systems they represent. Because the structure of a component often enables it to have more than one function, a central aspect of this understanding process is the inference of the intended function in the context of the design.

Automating this process would enable us to construct software coaching systems that offer appropriate advice based on inferences about the intent of the student's design, intelligent design assistants for verifying that a given structure would produce an intended function, and automated indexing and retrieval systems for cataloging designs based on their function rather than structural features. The difficulty in doing so lies in the combinatorics of the problem; if we suppose, conservatively, that each component can play three different roles, then for a system of twenty components there are 3^{20} or 3.5

billion functional construals. A thermodynamic system might have fifty components, an electrical circuit hundreds to millions, for VLSI.

In this dissertation we describe Topological iNference of Teleology (TNT) theory, an account of efficiently deriving function from structure via evidential reasoning. CARNOT, an implemented system based on this theory has correctly inferred the function of forty-nine single-substance thermodynamic cycles, including all such cycles contained in a standard text on the subject (Haywood 1991).

1.1 Context

This research builds on Forbus' work on qualitative reasoning about the behavior of physical systems (Forbus 1984). Although behavioral reasoning is useful for many tasks, it is generally insufficient when we are concerned with the intentions of a designer, which by definition lie in the social rather than the physical realm.

De Kleer (1979) was the first to investigate the derivation of function from structure. He proposed, for the domain of electronic circuits, a methodology using qualitative physics to map from *structure* (what the artifact is) to *behavior* (what the artifact does) and a separate, teleological reasoning process to infer *function* (what the artifact is for) from behavior. In contrast, we present here an account of deriving function from structure directly, via an evidential reasoning process, although we adopt de Kleer's nomenclature. In particular, we define a *function* to be an *intended behavior of a device.*

This work is similar in spirit to the research efforts of the Functional Reasoning community (e.g., Chandrasekaran 1994; Thadani 1994), yet it differs in that it excludes explicit representations of behavior from its knowledge base. Our goal in this regard is to construct a modular reasoning system that could be combined with a behavioral reasoner without concern for interactions in the respective knowledge bases.

Early versions of this work relied on dependency-directed search (Stallman and Sussman 1977) to filter out incorrect role assignments. Unfortunately the constraints that topology imposes on role assignments are not sufficiently binding, so the net result of such a search is a set of several equally likely construals with no principled means of preferring one over another. In response to this, we developed our evidential approach, which trades deductive soundness for efficiency and flexibility. Rather than rule out impossible construals, our approach attempts to rule in the most probable construal.

To arrive at this construal, TNT reasons about the topology of the input system, via a rich vocabulary for representing locality. This use of locality is similar in spirit to Sussman's *slice* concept (Sussman and Steele 1980), which used multiple overlapping local viewpoints to analyze electrical circuits. Davis (1983) has also noted that multiple representations of locality can provide inferential leverage in diagnostic applications.

The evidential reasoning algorithm employs Pearl's (1988) method for Bayesian hierarchical updating of belief. Our use of Bayesian probability theory arose from a conversation with Eugene Charniak (see also Charniak 1991), who strongly encouraged us to at least adopt what he terms "idiot Bayes," that is, the simplest instantiation possible of Bayesian theory that affords the appropriate inferential power without doing violence to the integrity of the formalism. This has remained our goal through the evolution of our plausible inference mechanism, and therefore we view our contribution in this regard as a lightweight evidential reasoning algorithm amenable to use with forward chaining rule engines for the purpose of qualitative inference. We do not claim to advance the state of the art in probabilistic or non-monotonic reasoning; with respect to these communities, our research is the beneficiary rather than a contributor.

1.2 Motivation

Over the past several years, we have been interested in the application of qualitative reasoning to engineering education. To this end, this work builds on the concept of the *articulate virtual laboratory* (AVL), a design-based exploratory learning environment that can provide students with explanations of its calculations and inferences, and on our experience in fielding a thermodynamics AVL called CyclePad (Forbus and Whalley 1994).

We have found that by itself, an exploratory environment such as CyclePad is insufficient. In particular, the space of possible designs is vast, and students can and do get lost in unproductive areas of this space. We have observed some of the problems with motivation and frustration that other investigators of exploratory environments have reported (cf. Reiser et al. 1994).

To address this issue in CyclePad, we decided to develop a software coaching system. If students were to use such a system it had to be fast enough to operate interactively. Our hypothesis was that an automated teleological reasoner would enable the construction of a coach that required as input only the current state of the problem, that is, the completed structure and the assumptions made to that point. Inferences about the function of the student's work would guide the advice presented to the student. In contrast to other work in intelligent tutoring systems (e.g., Baffes and Mooney 1996; Brown and Burton 1978; Brown and Van Lehn 1980), we do not require nor construct a model of the student, which simplifies the coach considerably.

The domain of thermodynamic cycles is of interest for several reasons. First, it concerns designed artifacts of considerable complexity, which provides a forcing function to move us beyond toy problems. In point of fact, the current version of CARNOT has achieved broad coverage of the domain (see Section 5.2.1).

Second, thermodynamics is fundamental to all engineering disciplines. For example, mechanical engineers must take heat from friction into consideration, electrical engineers must ensure the dissipation of the heat their circuits generate, and industrial engineers must take into account the thermal characteristics of the feedstocks to the processes they design. Engineering students are required to take at least an introductory thermodynamics class, and most engineering programs require a second course as well.

Finally, the centrality of design to this domain meshes with the design focus of articulate virtual laboratories. Students decide to become engineers because they want to design artifacts, and we believe we can tap into this intrinsic motivation to help students develop a deeper understanding of the domain than they would via conventional engineering curricula, with its emphasis on abstractions and analytical solutions.

1.3 Generality and Limitations

TNT theory has been developed and realized in great detail with reference to the domain of thermodynamics, but we have reason to believe that it would generalize to other physical domains, such as electronics, in which designs consist of topologies of interconnected components. TNT defines representations of structure and locality over which an evidential reasoning algorithm runs to produce plausible inferences of function. A primary purpose of the representations is to capture subtle nuances in the relative locality of components, which limits the applicability of this theory to domains which can be represented as graphs of components where the graph edges are the only means of causal communication.

TNT requires that one explicitly define the functions to be inferred for each recognized component. Thus in CARNOT a heater can be one of a preheater, a reheater, a heat-absorber, or a fluid-heater; these *roles* form an exhaustive partitioning of the space of function. Therefore the recognition of novel uses for an artifact is a task beyond the scope of this work; for example, a steel crowbar could function as an electrical conductor in a pinch, but this is not something a TNT-based system could infer, unless the situation was represented as an electrical circuit, a crowbar was a recognized device, and *conductor* was a recognized function of a crowbar device.

Finally, the evidential reasoning algorithm is completely domain-general, as it simply updates belief in certain propositions (in the case of CARNOT, these concern roles) based on available evidence. This algorithm requires that one be able to express domain knowledge in terms of evidential tests for or against given propositions (again, in the case of CARNOT, these are role propositions). It is also quite specific to our task at hand; we have traded the generality of a Bayesian belief network for a more computationally efficient method.

1.4 Contributions

The work presented here constitutes a performance theory of the teleology of physical domains. We characterize TNT as a performance theory because the concerns that have shaped it include runtime efficiency and the practicality of knowledge base development and maintenance.

Evidential reasoning enables efficient inference of function from structure in physical domains. A central contribution of this work is a demonstrably quadratic-time algorithm for such inference. TNT is the first account of evidential reasoning applied to the derivation of function from structure. The efficiency of our theory is due in large part to our representations of structural locality, which enable the inference of function directly from structure, without an intermediate behavioral simulation of the system.

Representing the domain knowledge required for the inference of function as evidential tests for or against particular roles enables our approach to scale. The scaling up of artificial intelligence systems is generally considered to be a function of algorithmic complexity, but another, equally important dimension is the scaling up of knowledge acquisition and maintenance. Casting knowledge in the form of evidence provides the domain expert with a straightforward means of expressing domain knowledge while at the same time enabling the dynamic composition of evidence into teleological inferences. CARNOT, our implementation of TNT, is the first teleological reasoning system we are aware of to achieve broad coverage of its domain; it currently identifies the functions of forty-nine thermodynamic cycles. A set of thirty-six cycles and CARNOT's construals thereof, may be found in Appendix B.

An automated teleological reasoner provides an efficient basis for the construction of software coaching systems. We have implemented an Analytical Coach that operates within the CyclePad system, and we have built a prototype of a case-based Design Coach, both of which are based on CARNOT. The Design Coach retrieves design cases that are functionally similar to the current design, and provides the user with access to the context of the cases. The Analytical Coach helps students find useful parts of design space by noting which assumptions diverge from the norm for that parameter. Norms require a context; for example, the typical value for pressure at the outlet of a water boiler in a vapor-cycle heat engine is quite different from the pressure at the outlet of a gas-turbine combustion chamber, yet both devices are represented as heaters, because in both cases we have an active heat-flow process.

The TNT architecture provides the student with explanations of its functional inferences. Although evidential reasoning is not deductively sound, we assert that this lack of soundness, when made explicit to students, is actually a pedagogical virtue, because it requires the student to consider the validity of the system's output. The system's explanations make clear the evidence for and against a given inference, providing the student with a basis for judging its accuracy. Taking advice at face value, whether from a human or machine expert, is rarely a good cognitive strategy.

Finally, as noted above, TNT has application beyond thermodynamics coaching. For example, a practicing engineer could use such a system to verify that a particular design will achieve its intended function. The teleological engine could also provide the foundation for automated indexing and retrieval of designs based on their function rather than their surface features.

Although introspection guided the development of the evidential reasoning algorithm, we make no claims of cognitive fidelity for TNT. We believe that the theory presents interesting hypotheses that could serve as the basis for further cognitive research. For example, aspects of the topological recognition algorithm are reminiscent of de

Groot's work (de Groot 1978), in which he showed that chess masters remember configurations of pieces as units, much as a literate person remembers words as units rather than as strings of characters. CARNOT, the system in which we have realized TNT, certainly runs in times comparable to a domain expert performing the same task, which at least does not provide an immediate reason to rule out cognitive plausibility.

We also make no pedagogical claims for this research, although pedagogical concerns have had a major influence on our thinking. The issues of how both the design and analytical coach should be implemented in order to effect measurable change in the depth of a student's domain understanding have yet to be addressed, and constitute an appropriate topic for another dissertation. Nonetheless, it would have been impossible to pursue such a research topic until the coaching engines that we have built based on this theory were in existence in a robust form suitable for distribution to students, and these tools are a primary contribution of this work.

1.5 Reader's Guide

The core of this work is contained in Chapters 3 and 4, which present the representations and the mechanism of the evidential reasoning engine respectively. Combined, these chapters constitute the Topological iNference of Teleology theory. Chapter 2 presents an overview of the domain of thermodynamic cycles sufficient for the purposes at hand and provides an illustrative example of the output of a TNT-based system based on a typical input cycle. While not necessary for understanding the material that follows it, this chapter does establish some useful context.

Chapter 5 discusses the CARNOT system, which is a completely implemented realization of the TNT theory that has correctly inferred the function of forty-six thermodynamic cycles to date (Appendix B presents thirty-six of these cycles and CARNOT's explanations thereof). This discussion includes a complexity analysis of the algorithm and an empirical analysis of CARNOT's performance on the first test set, comprised of 36 cycles. Chapter 6 describes the prototype coaching systems we have implemented based on CARNOT, while Chapter 7 discusses how this work relates to other research in the areas of plausible and qualitative reasoning. Finally, Chapter 8 concludes with a summary of this work and a discussion of directions for future research.

2
THE DOMAIN OF THERMODYNAMIC CYCLES

Artifacts incorporating thermodynamic cycles are pervasive. Virtually all electrical power generated today relies on a thermodynamic cycle in which massive boilers generate steam to turn turbines that drive generators. Refrigerators rely on essentially the same cycle, albeit running in reverse and supplied with a different working fluid that enables their operation at safer pressures. Automobile and jet engines operate in a so-called "open" cycle that takes in air from, and expels exhaust gases to, the environment, yet they

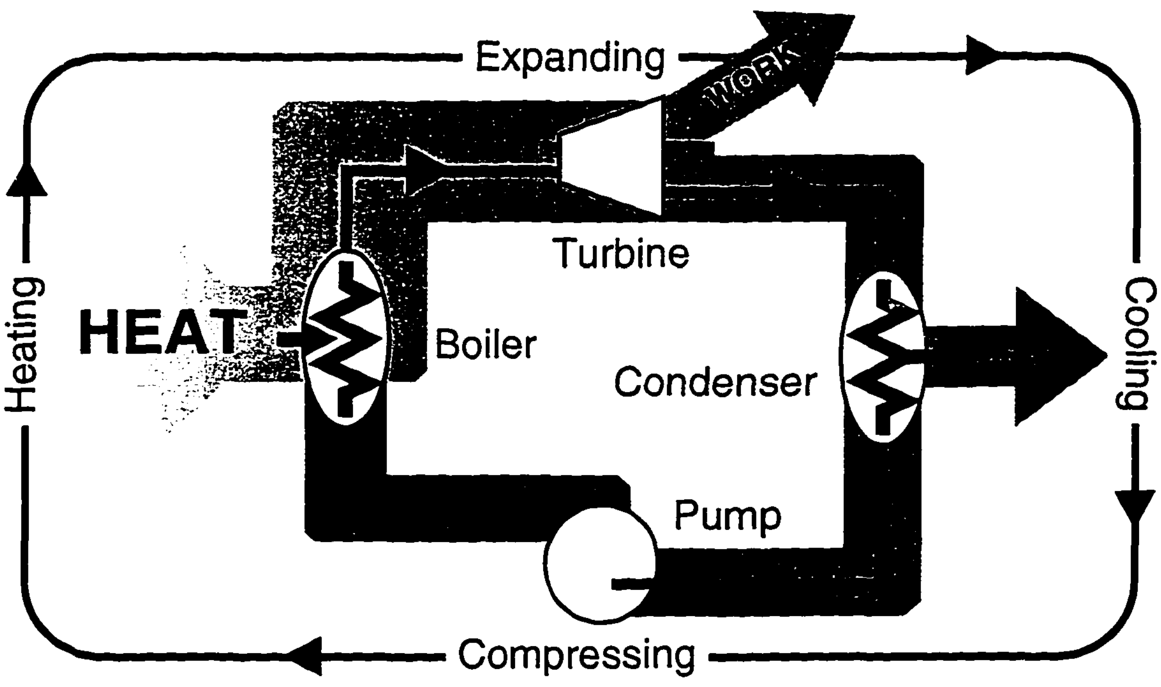


Figure 2.1 Simple vapor-cycle heat engine

may be analyzed as cycles by treating the atmosphere as a single reservoir of air. Industry relies on thermodynamic cycles for power, for liquefying gases (e.g., natural gas, nitrogen, oxygen), and for process steam.

2.1 An Overview of Thermodynamic Cycles

The defining characteristic of a thermodynamic cycle is that it operates between two reservoirs of different temperatures, typically by passing a working fluid[1] through a system of pipes and components. Figure 2.1 shows a simple cycle. This basic cycle (with some modifications to increase efficiency) is commonly used to generate electricity. Heat energy obtained from combustion or nuclear reaction converts the working fluid into vapor in the boiler. This vapor then expands in the turbine, causing its blades to rotate, producing work. The condenser returns the working fluid to its original state by ejecting heat to the environment. The pump ensures a steady supply of working fluid to the boiler and maintains the system's direction of flow.

Note that some heat must leave the cooler. This is a consequence of the Second Law of Thermodynamics; in practice it means that even an ideal cycle cannot achieve 100% efficiency. In other words, we cannot completely convert a given amount of thermal energy into mechanical energy.

[1] The term *fluid* in this work denotes a substance that flows, regardless of its phase. Gases and liquids are therefore both fluids.

We can view the cycle abstractly as a sequence of four physical phenomena; compressing, heating, expanding, and cooling. The design of thermodynamic cycles may be thought of as the manipulation of these four phenomena. The order in which these phenomena occur is dictated by physical law, and thus is a useful piece of evidence for reasoning about function. For example, a refrigerator cycle uses the same four phenomena in a different order—expanding, heating, compressing, and cooling—to achieve a refrigeration effect, and so phenomena order may be used to determine the type of the cycle.

Despite the fact that the constituent devices of this and other thermodynamic systems are complex artifacts designed to accomplish specific functions, there are significant ambiguities in the inference of function from structure in this domain. For example, a turbine may function as either a work-producer or a cooler, and in cryogenic cycles the latter is the desired function.

The *control volume* is an important theoretical concept in thermodynamics. It forms the boundary between the cycle and the rest of the environment. All thermodynamic systems are intended to achieve a change in their surrounding environment, which they do by exchanging energy and/or mass across the system's boundary. The control volume enables one to pinpoint these areas of interchange with the environment. For example, in Figure 2.1, energy in the form of heat crosses the control volume to enter the system at the boiler. Energy in the form of work leaves the system at the turbine, and finally the cooler ejects the inevitable "waste" heat to the environment.

2.2 Learning Thermodynamics

Thermodynamics is central to the study of engineering, because thermal processes are ubiquitous in engineered devices. For example, the design of the original Macintosh computer included carefully placed vents that cause the case to act as a chimney, funneling cooling air over the chips and avoiding the need for, and attendant noise of, an internal fan. Unfortunately, most courses in thermodynamics rely heavily on drill and practice in solving the basic equations.

This approach tends to decontextualize thermodynamics, abstracting it away from the realm of application. We believe that design-based problems can be used to strengthen the connection between thermodynamic theory and real-world problems. To test this hypothesis we have built CyclePad, an articulate virtual laboratory (Forbus and Whalley 1994). An articulate virtual laboratory is *virtual* in that it provides the user with the ability to build models of physical systems, such as thermodynamic cycles, without the inconvenience and expense of working with large and potentially dangerous physical components. It is *articulate* because the user can determine, via a hypertext system, the derivation of any conclusion the system generates. CyclePad has been in use at Northwestern University and the U.S. Naval Academy for the past two years, and we are continuing to collaborate with thermodynamics professors at these institutions and at

Oxford on the design and refinement of this system. A version of CyclePad that includes coaching systems based on our research is publicly available via the World Wide Web.[1]

2.3 An Example of a Typical Thermodynamic Cycle

Thermodynamic cycles are typically a topic of the second course in thermodynamics an engineering student may expect to take. In this section we will describe a cycle derived from the chapter on thermodynamic cycles of a standard introductory text for the field (Van Wylen, Sonntag, and Borgnakke 1994). This cycle, shown in Figure 2.2, is a jet-ejection air-conditioning system.

In this system chilled liquid flowing through Heater-2 absorbs heat from the area being air-conditioned (typically the passenger compartment of a transportation vehicle). This working fluid flow is then split, and part of it is pumped by Pump-1 to Heater-1, where an inflow of heat energy turns it into a high energy jet of steam. It flows at high velocity through Mixer-1, and in the process entrains and compresses the hot vapor from Splitter-3. Mixer-1 is therefore acting as a jet-ejection compressor, that is, a compressor which has no moving parts. Splitter-3 is acting as a flash-chamber, in which a sharp pressure drop causes the working fluid to flash into vapor; immediately upstream of it, Throttle-2 causes the requisite pressure drop. Because boiling requires heat, the portion of the working fluid that vaporizes extracts the heat required from the balance of the liquid which, now chilled, flows to Heater-2 to create the air conditioning effect.

[1] http://www.qrg.ils.nwu.edu/software.htm.

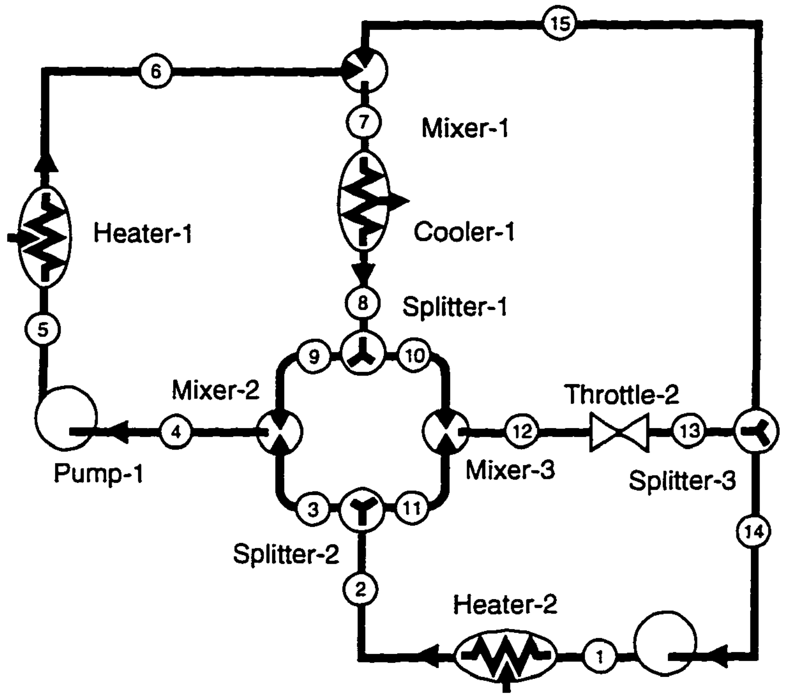


Figure 2.2 A jet-ejection refrigerator

The advantage of this design is that, aside from the two pumps which provide a mass-transfer function, there are no moving parts. It is essentially a heat-driven air-conditioner, and it typically uses a non-toxic working fluid—water—so it is safe for use in inhabited spaces. It would also be particularly useful wherever there is waste heat that could be used to drive the system, such as on a factory floor, to provide air conditioning for some spaces or cooling for processes that require particular environmental characteristics.

Notice that in this system there are two structurally identical heaters performing completely different functions, there are three mixers, two acting as flow joins and one as a compressor, and there are three splitters, two acting as flow-splits and one acting as a flash-chamber. This ambiguity in function is precisely what TNT is designed to resolve.

2.4 An Example of the Output of a TNT-Based System

There are three different types of output the user may obtain from a system based on TNT; (1) identification of a particular device's role along an explanation of why that role is believed true, (2) an explanation of the function of the whole system, or (3) a behavioral explanation that traces the path a packet of working fluid would follow, explaining the thermodynamic effects it encounters along the way. These explanations are based on simple natural language templates and phrases instantiated by rules. If the user were interested in the role of Heater-2 in Figure 2.2, which absorbs heat energy from the environment (i.e., it is the "business end" of this cycle), here is the explanation a TNT-based system would produce:

```
Q:   What is the role of Heater-2?
A:   Heater-2 is acting as a HEAT-ABSORBER
Q:   Why?
A:   Heater-2 is construed as a HEAT-ABSORBER because
     refrigeration cycles are more likely to use heaters as
     heat-absorbers than as energy-injectors.
Q:   Could Heater-2 be acting as a preheater?
A:   Heater-2 is unlikely to be acting as a PREHEATER
     because a heater on a non-work-generating subcycle of a
     refrigerator is unlikely to be preheating the working
     fluid and a refrigerator rarely has need to preheat its
     working fluid
```

These are predefined natural language templates into which the italicized symbols have been inserted. This explanation assumes that the user is familiar with the roles that a heater may play. If the user is not, definitions are available. Behaviorally, a heater may be thought of as a tube through which working fluid flows. A heat-conducting path to the environment causes a flow of heat to the working fluid in the tube, so we may conclude that the physical effect on the working fluid is an increase in its thermal energy, which will manifest as an increase in temperature and/or a change in phase. From an intentional point of view, there are two physical effects of particular interest to us in the context of reasoning about thermodynamic cycles; the flow of heat and the increase in the temperature of the working fluid. In the case of a heater playing the role of a heat-absorber, the intended effect is the flow of heat, and the increase in temperature is an unfortunate side-effect; if there were no increase in temperature, then we would have a thermal perpetual motion machine, and we wouldn't need the rest of the cycle.

The user may therefore conclude from the answer to the first question that the intended function of Heater-2 is to absorb heat from the environment. If the user had asked about Heater-1, the heater that powers the jet-ejection mixer, the system would have responded that it was acting as a heat-injector. In this capacity, the desired outcome is an increase in the thermal energy of the working fluid, and the requirement for a heat-flow from the environment is a sad fact of life; once again, if this were not necessary we would have a thermal perpetual motion machine.

The second question seeks an explanation for the construal of Heater-2. Notice that the answer is couched in terms of likelihood, because TNT is a plausible reasoning system. In this case, the explanation is simply that refrigerators are more likely to require heat-absorbers. The user might well not be satisfied with such an answer, perhaps believing that Heater-2 is preheating the working fluid (a preheater role is a specialization of the heat-injector role), and so can ask if it would be possible that Heater-2 is actually a preheater. The response in this case is again couched in terms of likelihood, and reveals that the system has also found evidence against such a role; the heater in question is not part of a work-generating substructure.

As an example of the second of the three types of explanations that TNT supports, the user might want to understand the intention of the system as a whole:

```
Q:   What is this system for?
A:   This is a refrigerator cycle, so it is intended to move
     heat from one location to another. To preserve this
     cycle's integrity, a jet-ejector is used to compress
     the working fluid. This takes place at MXR1. Jet
     ejectors have no moving parts and hence vapor
     refrigeration systems that use them for compression are
     inexpensive and safe to operate, although they produce
     effective cooling to levels well above the freezing
     point of the working fluid.
```

This answer is composed from the explanatory phrases associated with the plans instantiated for this cycle. Plans describe the functioning of cycles and major substructures in terms of the goals that they achieve. The system assumes a rational designer and imputes three goals; (1) achieving a change in the environment, (2) achieving that change efficiently, and (3) preserving the system (i.e., preventing the system from entering self-destructive states). In this case, this plan addresses the goal of achieving a change in the environment (i.e., the air-conditioning effect) and of preserving the system, via a design that has few moving parts.

Finally, the user might want to understand the behavior of the system at a detailed level. In this case the answer is composed by a set of rules that combine simple natural language templates, one for each device, into an overall description:

```
Q:  Explain the behavior of this cycle

A:  The liquid working fluid flowing through Heater-2
    absorbs heat from the environment.  The heated working
    fluid then flows to Splitter-2.  Splitter-2 splits the
    working fluid into two streams, one going to Mixer-2
    and the other going to Mixer-3.  Mixer-2 delivers
    working fluid from Splitter-2 and Splitter-1 to Pump-1.
    Pump-1 delivers liquid working fluid to Heater1.
    Heater-1 vaporizes the working fluid and delivers it to
    Mixer-1.  Mixer-1 acts as a jet-ejection pump, powered
    by the stream of high-energy working fluid from Heater-
    1.  It compresses the vapor from Splitter-3 and
    delivers the resulting mixture to Cooler-1.  Cooler-1
    cools the working fluid and delivers it to Splitter-1.
    Splitter-1 splits the working fluid into two streams,
    one going to Mixer-3 and the other going to Mixer-2.
    Mixer-3 delivers working fluid from Splitter-1 and
    Splitter-2 to Throttle-1.  Throttle-1 causes a drop in
    the pressure of the working fluid which results in the
    fluid becoming saturated.  This saturated mixture then
    enters Splitter-3.  Splitter-3 causes the working fluid
    to partially evaporate; gas, absorbing the heat of
    vaporization from the remaining liquid, exits Splitter-
    3 and flows to Mixer-1.  The chilled liquid flows to
    Pump-2.
```

In all of these answers, the difficult part is determining the role that a particular device plays. Plans are easy to recognize once role assignments have been made, and a behavioral description is easy to generate from a set of roles because roles entail expectations about behavior. The central position of a role in this theory results because the fundamental ambiguity is not in the causal model of the device, but in how the designer wants to exploit that causal model. A behavioral model of this cycle wouldn't enable us to explain this cycle because we need to make assumptions outside the realm of the underlying physical theory. The cycle simply doesn't contain the information we desire; it might well be that the cycle is intended to be a dynamic artwork. Thus the best we can do is provide the most likely explanation, not a correct one.

Our inability to deductively reason about the intention of a cycle is not so grim as it might seem. People routinely make such inferences in the social domain without a complete model, a fact that Schank has stressed (Schank 1986). Providing a plausible answer rapidly is often quite valuable. We contend that the evidential reasoner's lack of deductive soundness, when made explicit to students, is useful pedagogically, in that it requires the student to pass judgment on the validity of the advice. The system's explanations of the advice make clear the evidence for and against the inference, providing the student with a basis for judging its accuracy.

3
TELEOLOGICAL REPRESENTATIONS

TNT consists of two tightly interrelated components, a representation language and a reasoning mechanism. The representation language, which is the subject of this chapter, provides us with the ability to define localities within the topology from multiple points of view, and the reasoning mechanism, discussed in Chapter 4, enables the dynamic refinement of these representations as it propagates to quiescence.

The representations that TNT defines are shown in Figure 3.1. The flow of inference is from left to right, and the overlapping of the locality and role representations illustrates the tightly interleaved nature of the inference mechanism. Note that, aside

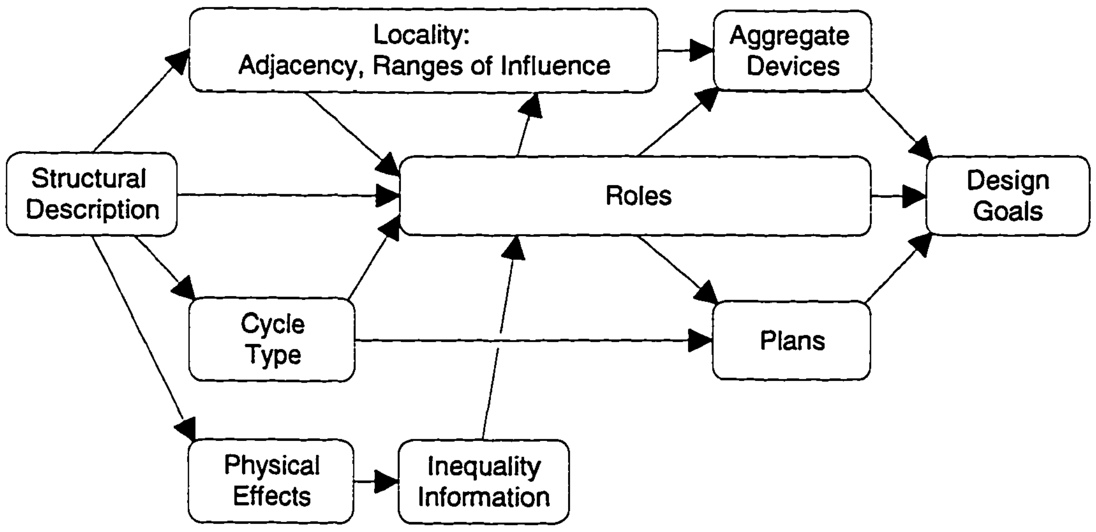


Figure 3.1 TNT representations

from cycle-type, nothing in these representations is specific to the domain of thermodynamics. They would apply equally well to any domain in which structure is represented as a graph whose edges constitute the sole mechanism for causal communication. The following discussion, however, is grounded in the domain of thermodynamics.

The design goals at the right of the figure are those that we touched on briefly in Chapter 2. Design goals provide the context for the explanation generated. To recap, there are three design goals:

1. Achieve a change in the environment
2. Achieve the change efficiently
3. Preserve the system

The three representations immediately to the left of design goals—aggregate devices, roles, and plans—are defined a priori to entail the achievement of one or more of these goals. Natural language templates associated with each of these representations provide explanatory text in terms of the design goals. For example, a pump has two potential roles, flow-producer or flash-preventer. In its capacity as flash-preventer, the pump increases the pressure of the working fluid so that subsequent injection of heat does not cause the fluid to prematurely vaporize. This role directly achieves the third design goal, that of preserving the system, because premature flashing would have adverse consequences for downstream devices, such as the system's main boiler, which might melt. Plans both directly achieve goals and may also associate a particular goal with the role of a particular device.

In the rest of this chapter, we will describe each of the representations shown in Figure 3.1, with the exception of the identification of cycle type, which is discussed in Chapter 4. In general our discussion will follow the left-to-right organization of Figure 3.1.

3.1 Structural Representations

The representation of the input structure is in terms of devices and their immediate connectivity, that is, their upstream and downstream neighbors. This representation balances the need for expressiveness against computational tractability. The primary abstraction in this representation is the fixing of the number of inlet and outlet ports for each device type. In reality, some thermodynamic components may have an arbitrary number of ports. We model such components as composites of our device representations, treating the latter as primitive stages; for example, a steam turbine in a regenerative power plant cycle would be modeled by connecting several turbine devices in series with splitters, which provide the necessary bleed ports for regeneration. Our set of devices, shown in Figure 3.2, is also the minimal set capable of modeling the single-substance cycles commonly found in textbooks; with this set we have constructed models of all such cycles to be found in *Analysis of Engineering Cycles* (Haywood 1991).

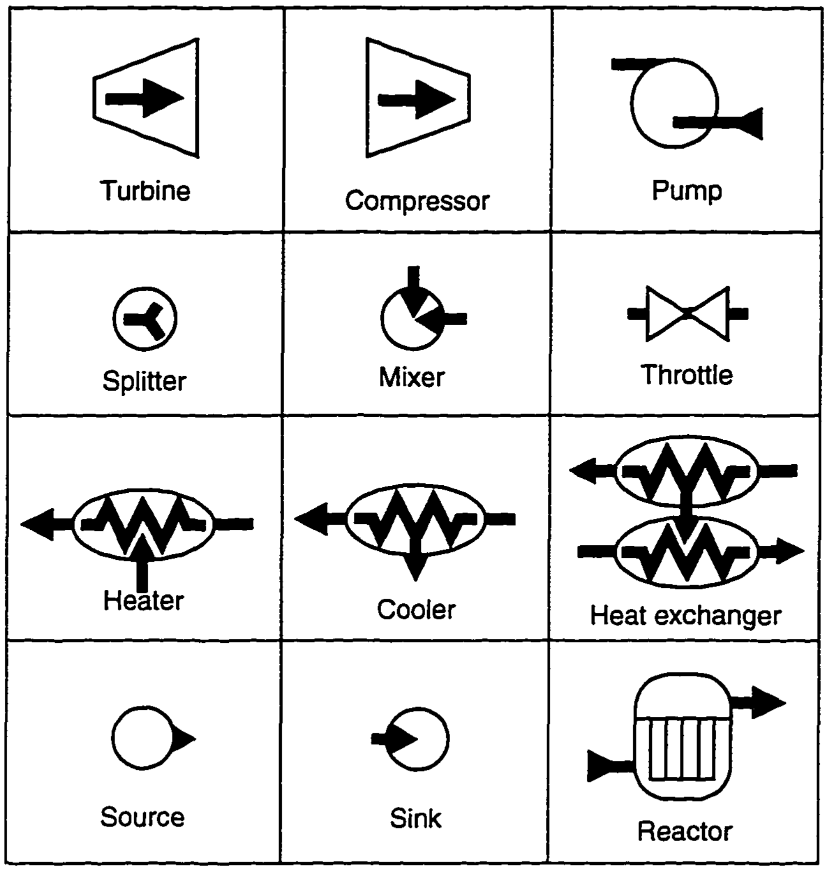


Figure 3.2 Schematic device representations

Some engineering notations include symbols for more specific types of components, such as jet-ejectors and flash-chambers, which we model as mixers and splitters, respectively. One might argue that the inclusion of such symbols would, at the extreme, render our theory superfluous, since each component symbol would have a one-to-one mapping to its function, eliminating any functional ambiguity in the structural representation.

There are two countervailing considerations, one practical, and one pedagogical. From the practical point of view, we would merely convert ambiguity in function into ambiguity in structure by adopting function-specific symbols. For example, the same pump may be acting as either a flash-preventer or a flow-producer, so we would end up with two symbols for the same component. A contractor constructing a system based on such a schematic would face the dual of the disambiguation task that we are addressing.

Pedagogically, a primary goal of CyclePad is to bring student to reflect on the thermodynamic processes taking place in the cycle at hand. Requiring the use of a limited palette of components, we believe, helps students to connect their understanding of abstract thermodynamic processes with the concrete realities of cycle design. Whereas a jet-ejector icon would provide the student with a black box ready for plugging into a design without further thought, forcing the student use a mixer to model a jet-ejector requires some thought about why such a model is appropriate.

3.2 Components and Their Potential Roles

An abstraction hierarchy of component types enables evidential tests to be written for the appropriate class of devices. This hierarchy is shown in Figure 3.3. The first level of abstraction is represented via a *genus* predicate, whereas the second is represented via a *phylum* predicate.

Each component is also defined to have from two to five distinct roles. The space of function is partitioned a priori into a taxonomy of these roles, as shown in Figure 3.4.

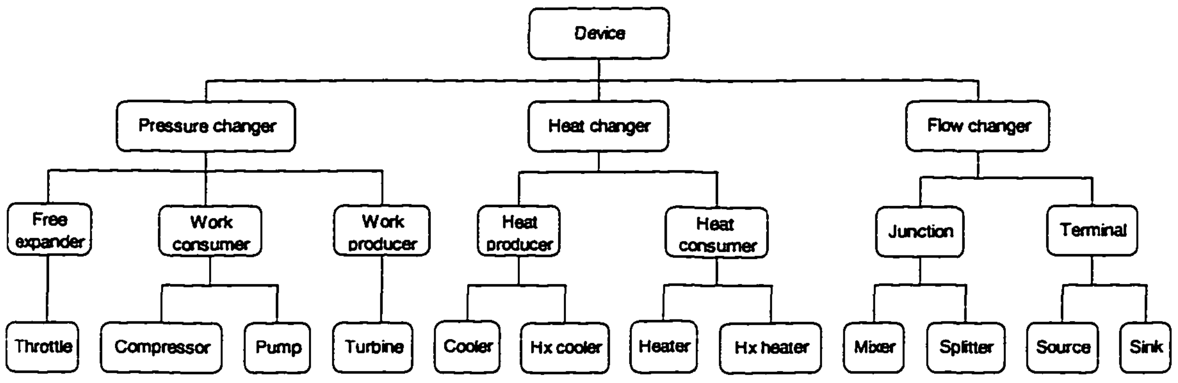


Figure 3.3 Device type hierarchy

Note that roles may be hierarchical. For example, the heat-injector role specializes into preheater, fluid-heater, and reheater, and the heat-ejector role of a cooler specializes into inter-cooler and fluid-cooler roles. This hierarchy enables TNT to better cope with ambiguity. For example, in some situations there may be evidence that a particular heater is a heat-injector, and we would like to be able to take this into account even when we are unable to disambiguate between preheater and a fluid-heater roles. The balance of this section describes the device representations; readers familiar with thermodynamics may wish to skim or skip this material.

Cooler. A cooler ejects heat energy from the working fluid to the environment. When acting as a heat-ejector, its purpose is to reduce the energy of the working fluid flowing through it. When acting as a heat-provider, its purpose is to provide heat to the environment. A hot-water radiator is actually a cooler operating as a heat-provider. A heat-ejector role is further partitioned into a fluid-cooler, which acts to cool its working fluid, and an intercooler, which is interleaved with compressor stages and is intended to reduce the work required of later stages by cooling the gas that is being compressed. The intercooler role subsumes the fluid-cooler role, yet we make this distinction because intercoolers are constituents of common thermodynamic structural “idioms” that provide valuable functional information. For example, an intercooler is an indication that the attendant rise in the temperature of the working fluid is an undesirable side-effect.

Heater. A heater injects heat energy from the environment into the working fluid. When acting as a heat-injector, its purpose is to increase the energy of the working fluid flowing through it. When acting as a heat-absorber its purpose is to cool the environment by absorbing heat from it. The coils in a domestic refrigerator comprise a heater that absorbs heat energy from the food in the refrigerating compartment. A heat-injector role is partitioned into fluid-heater, preheater, and reheater roles. A preheater adds heat to the working fluid upstream of the main heater (which acts as a fluid-heater) and a reheater adds heat between the stages of a turbine. Both are strategies for increasing the efficiency of the system (the second rational-designer goal), and hence are important roles to distinguish.

Reactor. A reactor, like a heater, injects heat-energy into the working fluid. Unlike a heater, however, it does not have a heat-path that crosses the system boundary to connect it to a heat-source in the environment. Reactors, therefore, are never heat-absorbers.

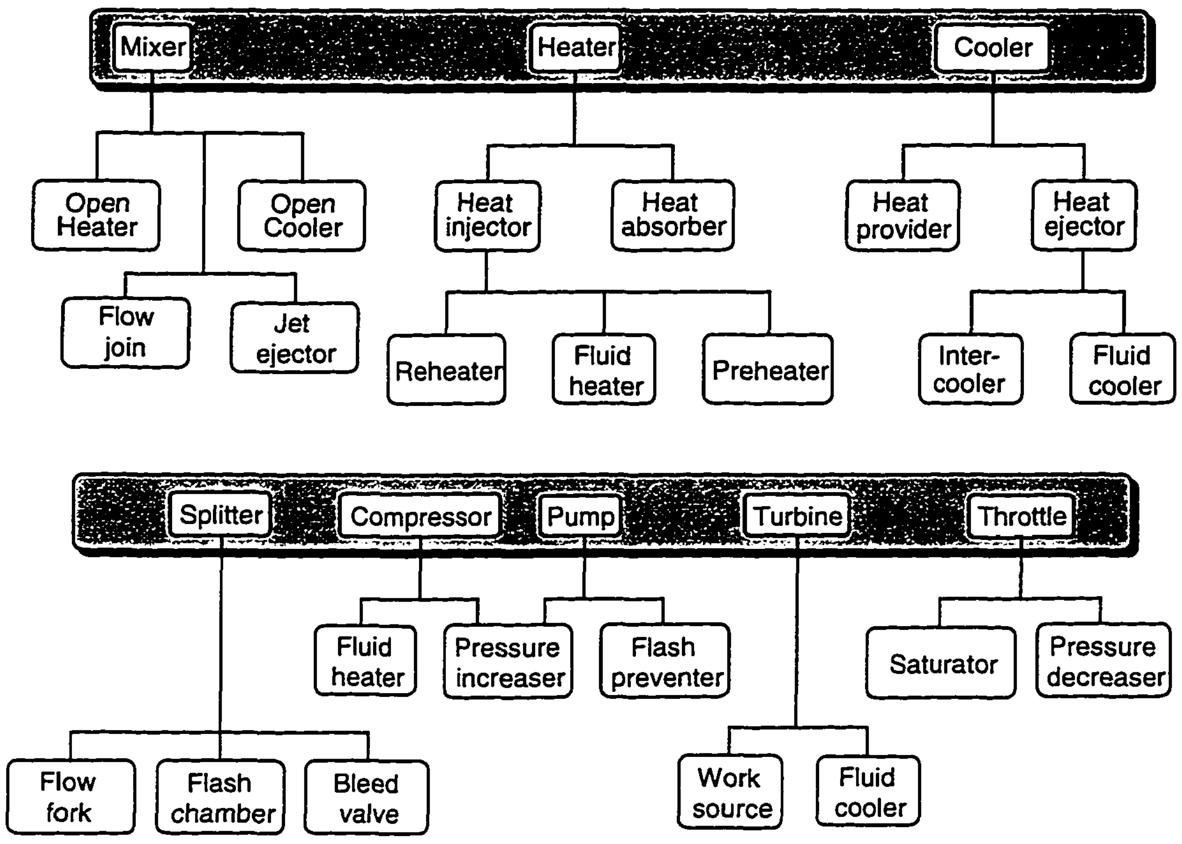


Figure 3.4 Role taxonomy

Heat-exchanger. The representation of heat-exchangers is slightly more involved. From a structural perspective, a heat-exchanger consists of a device with two inlets and two outlets, but from a functional perspective it is more useful to consider it to be a heater and a cooler connected via a heat-conducting path.

Hx-cooler. An hx-cooler is a cooler that ejects thermal energy to a heater, to which it is coupled. Hx-coolers take on the same roles as coolers, but additional testing is necessary to ensure that their corresponding heater halves take on appropriate roles. For example, if a hx-cooler is acting as a heat-provider, then its heater half must be acting as a heat-injector. Otherwise the design would fail to achieve any of the design goals, but would instead be arranging to provide heat to a device that in effect throws it away.

Hx-heater. An hx-heater is a heater that receives thermal energy from a coupled hx-cooler, but in other respects behaves like a heater. As in the case of the hx-cooler, additional testing is necessary to ensure that its corresponding half is playing an appropriate role.

Turbine. A turbine consists of a series of fan blades arranged along a shaft; high-energy working fluid expanding through the turbine causes the shaft to rotate, converting thermal energy into mechanical energy, or work. The most common role of the turbine is to produce work. However, the resisted expansion that takes place within the turbine also causes the temperature of the working fluid to fall appreciably, and so turbines can also act as fluid coolers. They are used in this capacity most often in gas-liquefaction systems, where the greater decrease in temperature offsets the greater complexity of a turbine over a throttle, which affords an unresisted expansion of the working fluid.

Compressor. A compressor is very similar to a turbine, but instead of allowing the working fluid flowing through it to expand, mechanical energy applied to its shaft causes the working fluid to be compressed. Compressors can only operate on gaseous working fluids, because condensation causes significant erosion of their fan blades. Compressors are among the least functionally ambiguous components, most often acting as pressureincreasers. There are circumstances, however, in which the attendant increase in thermal energy is the desired effect, and hence they can also act as fluid-heaters.

Pump. A pump, like a compressor, generally acts as a pressure-increaser, although it can only operate on liquids. If the working fluid is a mixture of liquid and gas (i.e., in the saturated phase) then the pump will cavitate, a process in which pockets of liquid boil and implode, creating shock waves likely to cause mechanical failure. For this reason, pumps can also act as flash-preventers; by increasing the pressure of the working fluid, they prevent it from flashing into a saturated vapor mixture. This is often necessary when preheaters are used to improve the efficiency of the cycle, and is an example of the third rational-designer goal, preserving the integrity of the system.

Throttle. A throttle, like a turbine, causes the working fluid to expand, but unlike a turbine a throttle generates no work. Throttles have two roles, saturator and pressure-decreaser. A saturator (which subsumes pressure-decreaser) is intended to cause the working fluid to change from either a gas or a liquid to a saturated mixture. A domestic refrigerator uses a throttle to cause the refrigerant that has been cooled in the coils on the back or bottom of the refrigerator to partially vaporize. This vaporization takes place in the cooling coils of the refrigerator, and, because vaporization requires heat, the refrigerant literally sucks the heat out of the contents of the refrigerator. Throttles are also used to step down the pressure of a working fluid to allow it to mix with another, lower-pressure stream. This is a common application in heat-engines, where steam bled from the turbine and used to preheat the working fluid flowing to the boiler must be fed back into the system at a lower pressure. In some cycles this reduction in pressure is assumed to occur within the heat-exchangers, so the throttles may be implicit in the design. TNT handles both explicit and implicit throttles; for an example, see Cycles 34 and 35 in Appendix B.

Mixer. Mixers are among the most flexible of components. They may act as simple flow-joins, as open heaters, open coolers, or as jet-ejectors. In addition, they may form the inlet half of a type of *aggregate device* called a steam drum, in which case they are considered to be playing the role of an open-heater. We describe aggregate devices in greater detail below, in Section 3.5. A flow-join simply joins two flows, and is subsumed by the other three roles. An open-heater is a direct-contact heat-exchanger that acts to increase the heat of one stream by mixing it with a hotter stream. Such mixing is more thermodynamically efficient, although it requires the streams to be at the same pressure, which may require more pumps in the system. Direct-contact heat-exchange is also used to deaerate the working fluid in power-plants, as oxygen dissolved in the working fluid can cause corrosion. An open-cooler acts to cool a given flow of working fluid by mixing it with a cooler fluid. Open-coolers are much less common than open-heaters. A jet-ejector uses a high-velocity stream of working fluid to entrain and compress another stream of working fluid, in effect acting as a pump. Whereas pumps receive an input of mechanical energy, jet-ejectors utilize thermal energy. This can be more efficient when there is a source of heat that would otherwise go to waste.

Splitter. Splitters, like mixers, can play several different roles. They may act as simple flow-forks, as flash-chambers, or as bleed-valves. A flow-fork splits a flow into two flows, and is subsumed by the other three roles. A flash-chamber is a container in which the working fluid, having just undergone a decrease in pressure, either by flowing through a turbine or through a throttle (which would be acting as a saturator), suddenly changes phase to a saturated mixture. The liquid part of the flow is then separated from the gas part. Flash-chambers are most commonly used in gas liquefaction plants, in which a cooled and compressed working fluid is allowed to precipitate in the flash-chamber. Occasionally industrial refrigerators or air-conditioners will make use of a flash-chamber, and in some nuclear plants the steam flowing into the final turbine stage first passes through a flash chamber to remove the liquid that would otherwise damage the turbine blades. Bleed valves are flow-forks between the stages of a turbine that are intended to bleed off a portion of the working fluid, typically for use in preheating the working fluid flowing to the boiler. Although bleed-valves are tantamount to flow-forks, they tend to be used in recurring design patterns and so we have found distinguishing this role to be quite useful in inferring the role of other components connected to them.

Source. A source simply supplies a working fluid, and hence plays only this one role.

Sink. A sink simply receives a working fluid, and hence plays only this one role.

3.3 Physical Effects

As we saw in the example in Chapter 2, several physical phenomena occur within a heater; there is a heat-flow from the environment to the working fluid, and as a result the temperature of the working fluid may increase, or the working fluid will undergo a phase change, such as boiling. Qualitative Process Theory (Forbus 1984) makes precise distinctions between a process, such as heat-flow, and its effects, such as a change in temperature. From a teleological perspective we are primarily concerned with effects, such as a change in temperature or phase, so our representations of process are minimal.

The central concern in TNT is the attribution of intention to effect. An effect can be intentional or a side-effect. In some cases, side-effects are beneficial, in others they have no impact, and at times they are necessary evils. For example, in a heater acting as a heat-absorber with a subcooled working fluid flowing through it, the temperature of the working fluid will rise; this is an unfortunate side-effect, because if it didn't we'd have an infinite heat-sink, which would be tantamount to a thermal perpetual motion machine. The heat-flow process from the environment to the working fluid will produce a decrease in the temperature of the environment, which in this case is the intended effect. In contrast, a heater acting as a heat-injector is intended to increase the thermal energy of the working fluid (i.e., inject heat), and the fact that the temperature of the environment decreases (or would decrease if we didn't maintain a steady rate of combustion) is an unfortunate side-effect.

As we have noted, TNT is a theory of reasoning directly from structure to function, without requiring a behavioral reasoning step. Therefore there are no causal models of the devices in this system. Propositions about effects and cycle requirements (described below) verge on behavioral knowledge, but are not sufficient to reason about behavior, and in fact are used mostly to detect errors in inference. Device-type statements and roles, which the domain expert defines, package most of the system's behavioral knowledge.

For each device, TNT instantiates inequality statements about temperature and pressure, the two parameters we can most directly manipulate within a cycle. For example, one consequence of finding a heater is the fact that the temperature of the outlet is greater than or equal to the temperature of the inlet. This soft inequality is necessary because the heater might be accepting a saturated liquid, that is, one just on the verge of boiling, and vaporizing it, returning saturated vapor. This process requires large amounts of energy, but doesn't produce a change in the temperature of the fluid. Only when the fluid is completely vaporized can further heat flow cause the temperature to rise, in which case the fluid is said to be superheated. For other devices, we can assert hard inequalities; a turbine, for example, must produce a decrease in pressure (and also a decrease in temperature, generally a neutral side-effect, although sometimes exploited in cryogenic cycles).

Performing a transitivity analysis on these statements enables us to detect some types of malformed cycles, which is particularly important in pedagogical contexts, as students may not always act as rational designers. Suppose that a student has constructed a cycle in which there are compressors but no throttles or turbines. If we start at an arbitrary point on the cycle, assume the pressure at that point is a reference pressure, and then employ transitivity to infer the relationship of pressures at downstream points, we will eventually find that the pressure never decreases, which of course is impossible.

This computation turns out not to be as useful as we first thought, due mostly to the presence of the soft inequalities (i.e., <=, >=), which introduce ambiguity into the transitive inferences. In some cases (e.g., the cycle is a heat engine that contains only compressors, indicating the working fluid never condenses) these soft inequalities may be "hardened" (i.e., converted into < or >), reducing the ambiguity and providing more inferential leverage.

Inequality information may also be used to determine the role of a device, particularly for mixers; a temperature difference across the inlets to a mixer is strong evidence that the mixer is acting as an open heat-exchanger (in which a hot stream mixes with a cooler stream in order to increase the temperature of the outlet), while a pressure difference is strong evidence that the mixer is acting as a jet-ejector (a type of pump, as we saw in the example of Chapter 2).

3.4 Domain Requirements

In addition to reasoning about physical effects, TNT also makes inferences at a global level, reasoning by exclusion based on domain characteristics. For example, heat-engines and refrigerators must operate between a high-temperature reservoir and a low-temperature reservoir, and hence both must contain at least one heater and at least one cooler. Heat-engines require a means of converting thermal into mechanical energy, so they require at least one turbine. TNT uses the *requires* and *singleton* relations to represent the global information used in this reasoning.

Requires: (requires *<entity>* *<required-role>*). The requires predicate states the requirements for the cycle, in terms of a role. For example, all heat-engines require a work-producer, a heat-injector, and a heat-ejector. A refrigerator requires both a heat-absorber and a heat-ejector.

Singleton: (singleton *<entity>* *<context>*). The singleton relation may apply in the global context, as in the case where a cycle has a single cooler, or in a context defined by other roles. For example, if a refrigerator cycle contains two heaters, one of which is construed as a heat-injector, then in this context the other must be a singleton heat-absorber if the cycle is to operate as a refrigerator.

3.5 Locality Representations

A critical aspect of mapping from structure to function is reasoning flexibly about locality. Although the representation of components via the (*<type>* *<name>* *<inlet>*

<*outlet*>) form captures information about directly-connected components, we also need to be able to describe and reason about a broader neighborhood, the structural context in which a particular component is embedded. For this purpose TNT defines the concepts of *adjacency*, *ranges of influence*, and *aggregate devices*. To illustrate these relations and how they support teleological inferences, we will use the cycle shown in Figure 3.5, which is a relatively simple regenerative Rankine heat-engine.

In this system the main fluid loop is indicated by the shading, and the power is generated by the three turbines in series across the top of this loop. Between these turbines are two splitters acting as bleed-valves that tap some of the high-energy steam

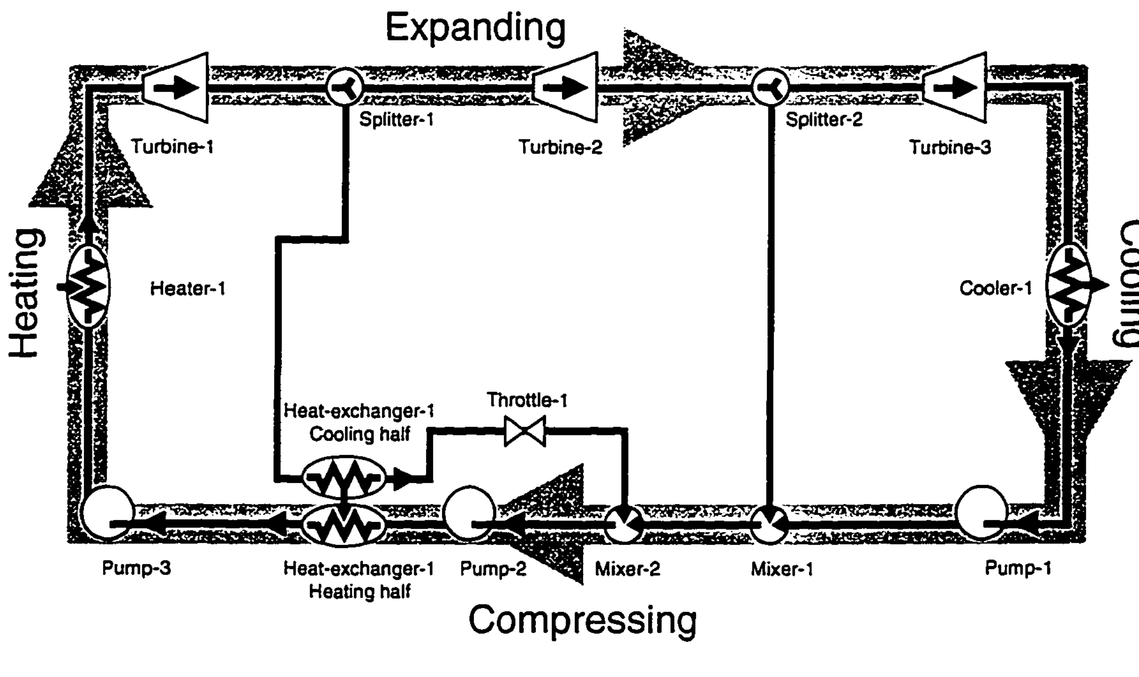


Figure 3.5 Regenerative Rankine power cycle

off for use in preheating the working fluid. This preheating occurs in Heat-exchanger-1 and in Mixer-1, which is acting as an open heat-exchanger. Preheating increases the average temperature of heat-injection, which increases the thermal efficiency of the cycle, an example of the second design goal. Heater-1 is acting as the boiler of the system, and Cooler-1 as the condenser. Pumps 1, 2, and 3 provide the compression. Note that, despite the greater complexity of this cycle, we still find the four fundamental processes—compressing, heating, expanding, and cooling—occurring in the same order as they did in the simple cycle presented in Chapter 2.

Adjacent: (adjacent *<upstream-component>* *<downstream-component>* *<path>*). Two components are adjacent if they are either directly connected to one another or connected solely via splitters and or mixers that are acting as flow-forks or bleed-valves and flow-joins respectively. Flow-joins and flow-forks have no effect on the thermodynamic properties of the working fluid, acting only to divide or join physical streams that are considered to be qualitatively identical in their intensive properties.[1] The adjacent relation therefore can be used to express the fact that two turbines joined in series by a splitter acting as a bleed-valve are in fact neighbors. Note that the adjacency relation depends on the current functional construal of the cycle, and will be retracted if a splitter or mixer connecting the two components is construed to have a role other than flow-join,

[1] An intensive property, such as temperature, remains constant when one changes the amount of a substance. Fill a coffee mug with hot coffee and the temperature of the coffee in the mug and in the pot will be the same. In contrast, extensive properties, such the thermal energy in the coffee pot, depend on the amount of the substance.

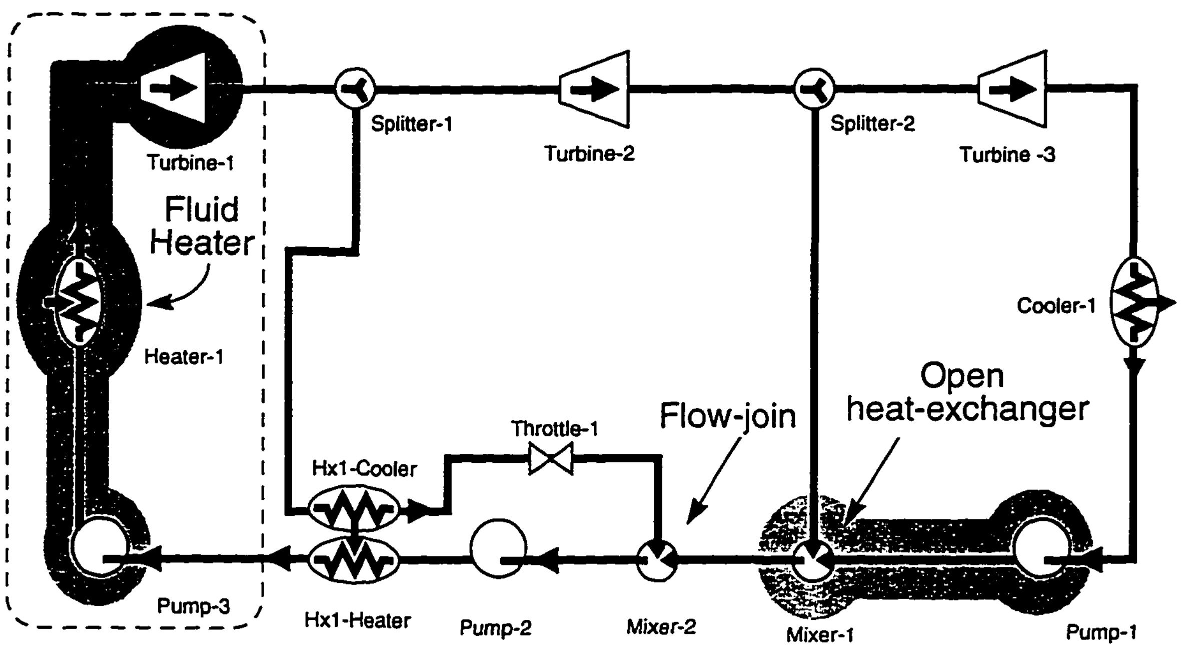


Figure 3.6 Adjacency

flow-fork, or bleed-valve. This is one example of the tight interleaving between the locality and role representations that occurs during the inference process.

In Figure 3.6 the shading connecting Pump-1 to Mixer-1 in the lower right portion of the diagram illustrates the simplest definition of adjacency, that of neighboring devices. If Mixer-1 were not acting as an open heat-exchanger, but only joining the flows from Splitter-2 and Pump-1, then nothing of thermodynamic interest would be occurring at this junction, and the adjacency relation would extend through it (and through Mixer-2 for the same reason) to Pump-2. Note that this relation might well change as more information becomes believed during the course of inference.

The locality enclosed by the dotted line on the left side of the diagram illustrates one way in which adjacency relations provide evidence for particular roles. In this cycle we have three heaters in this system, Mixer-1 in its capacity as an open heat-exchanger, the heater half of Heat-exchanger-1, and Heater-1. Which of these is the primary fluid-heater (i.e., the boiler) of the system? By definition, the primary heater of a system supplies high-energy working fluid to the work-producing devices, so therefore it must occur, topologically, immediately upstream of the turbines and downstream of the last compressing device. The adjacency relations between Pump-1 and Heater-1 and Heater-1 and Turbine-1 provide strong evidence that Heater-1 is the primary heater of this cycle, and is therefore acting as a fluid-heater (which is a specialization of the heat-injector role), because the heater is the last one upstream of the cycle's turbines.

Downrange: (downrange <*upstream-component*> <*list of downstream-components*>). Locality in thermodynamic cycles may be also thought of as the extent of a component's effect on the working fluid. For example, a pump's effect on the working fluid—increasing its pressure—persists until the working fluid passes through a device, such as a throttle or a turbine, that causes a reversal—in this case, a decrease in pressure—of the original effect. Regions of affect are defined to extend downstream from the component, and are termed ranges of influence; the downrange relation represents these ranges. Ranges may include splitters, as in the case of Turbine-2 in Figure 3.7.

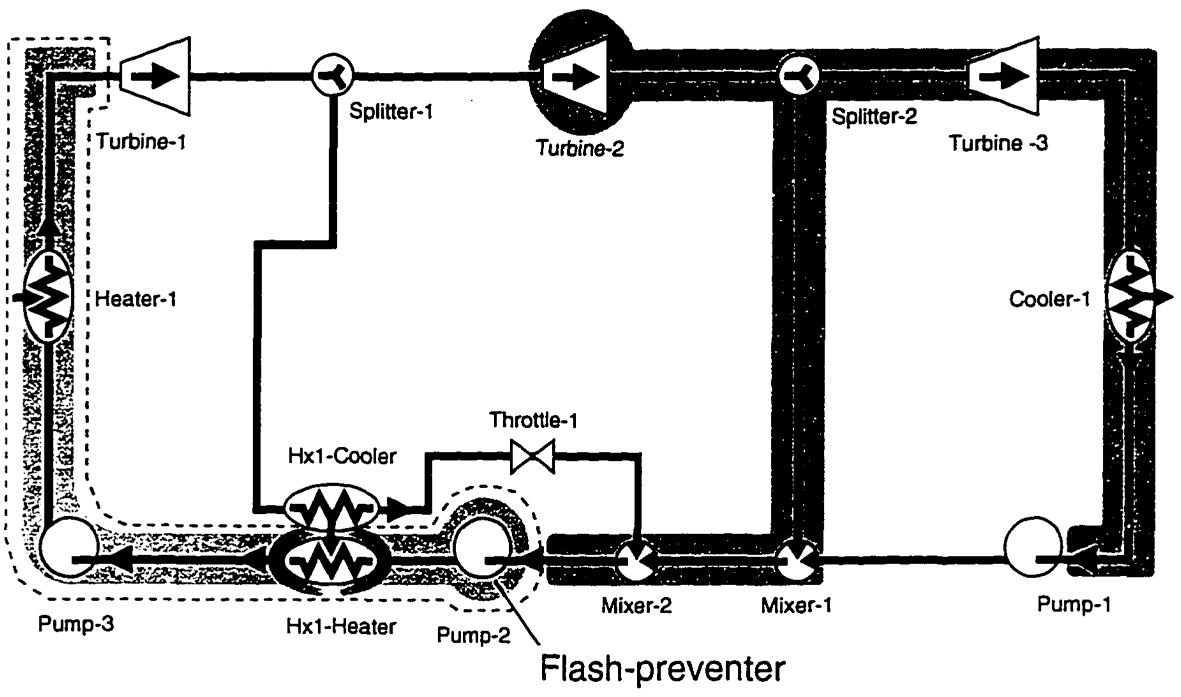


Figure 3.7 Ranges of influence

To understand how this relation may be useful, consider the range of influence of Pump-2, shown within the locality encompassed by the dotted line of Figure 3.7. Pumps may play one of two roles, flow-producer or flash-preventer. A flash-preventer increases the pressure on the working fluid in order to prevent it from vaporizing prematurely. In the above cycle, premature vaporization caused by the heat-injection occurring in Hx1-heater, would cause Pump-3 to stall, because handling partially-vaporized working fluids is a difficult engineering task. If the flow of working fluid to Heater-1, the boiler of the system, is interrupted even momentarily, the vast amount of thermal energy flowing through Heater-1 will cause serious damage, to the point of melting portions of the heater.

In a large powerplant, the boiler may be ten stories tall, and is generally suspended within a framework of girders rather than resting on a foundation because it will stretch as much as a yard in the vertical dimension before reaching operating temperatures.

Finding a pump with a heater and another pump downstream, and then another heater feeding a turbine downstream of that would be strong evidence for the first pump playing a flash-preventer role. One could define a specific template, yet this would prove unworkably brittle, because an arbitrary number of devices may occur within between the initial pump and the turbine without changing the strength of the evidence. For example, Mixer-2 might be positioned between Hx1-Heater and Pump-3 (in an arrangement known as *feed-forward*), rather than in its position upstream of Pump-2. The downrange relation provides us with a precise locality in which to look for a heater, then a pump, then another heater, and finally a turbine, in that order but without precisely specifying the relative locations of each device (just their relative ordering), and thus allowing for an arbitrary number of intervening devices. However, should one of these intervening devices be a turbine or throttle, that would terminate Pump-2's range of influence and prevent the inference, since one of the essential components (e.g., the downstream pump, Pump-3) would not be part of the range of influence.

Teleological Patterns. Certain configurations of devices that are playing particular roles occur with sufficient frequently that we gain leverage by representing them explicitly. In

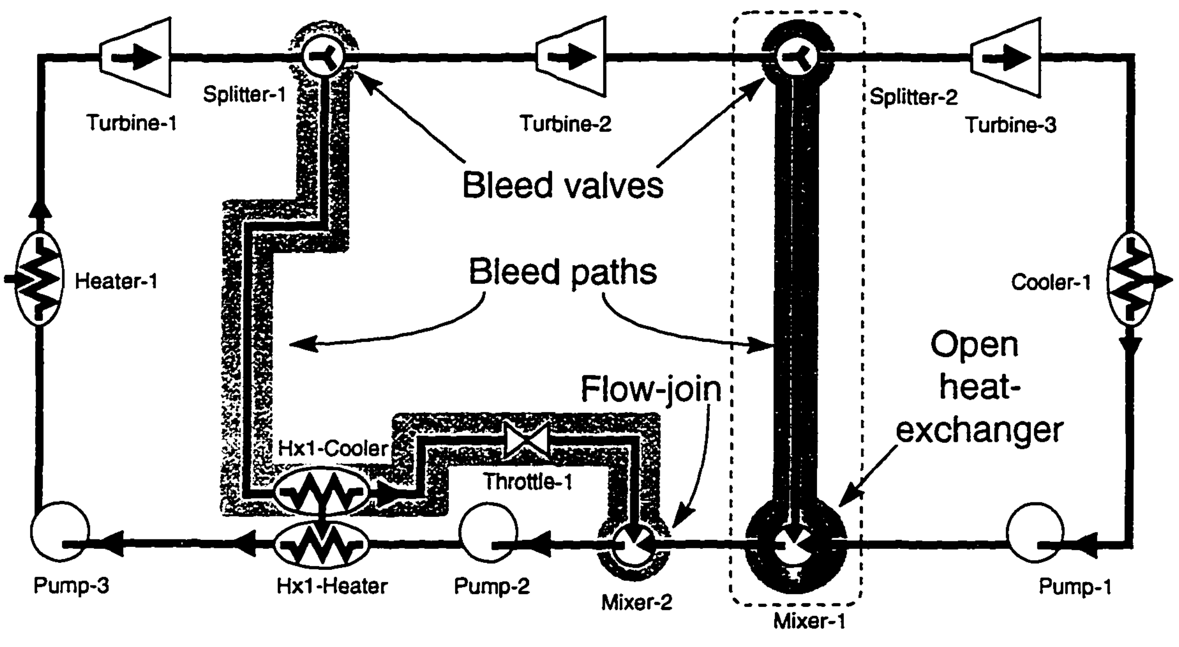


Figure 3.8 Examples of teleological patterns

our work with thermodynamic cycles, we have identified two such patterns, *bleed-paths*, and *subcycles*.

Bleed-paths enable the designer to achieve the second design goal, efficiency. In Figure 3.8 there are two bleed-paths in this cycle, one starting at Splitter-1 and terminating at Mixer-2, and the other starting at Splitter-2 and terminating at Mixer-1. Bleed-paths are common because it is a fact of thermodynamics that bleeding a small portion of the steam flowing through a turbine and using it to preheat the working fluid will increase efficiency; recall that, by the Second Law of thermodynamics, we cannot convert all the heat energy in this portion of steam into work. Although there will be some inefficiency in the heat-transfer process, we can utilize most of this heat energy for the purpose of preheating the working fluid, and therefore gain more advantage by bleeding this portion of steam than by extracting work from it.

Subcycles are the most general patterns, and may in fact subsume the entire cycle. In a system containing more than one subcycle, subcycles are connected solely via the heat-path of a closed heat-exchanger; by definition, no fluid-path can exist between two subcycles. Due to their generality, subcycles, unlike steam-drums and bleed-paths, may play several different roles and may also be the primary participants in plans. Figure 3.9 illustrates the roles a subcycle may play.

A subcycle acting as a work-generator will contain a heater, a turbine, a cooler, and a compression device, and constitutes a heat-engine. This and the energy-remover role, which removes heat from an environment (and hence acts as a refrigerator or heat-pump), are the most common roles. Work-generators are further specialized into simple-engines, topping, bottoming, and cascading subcycles. A topping subcycle is one whose

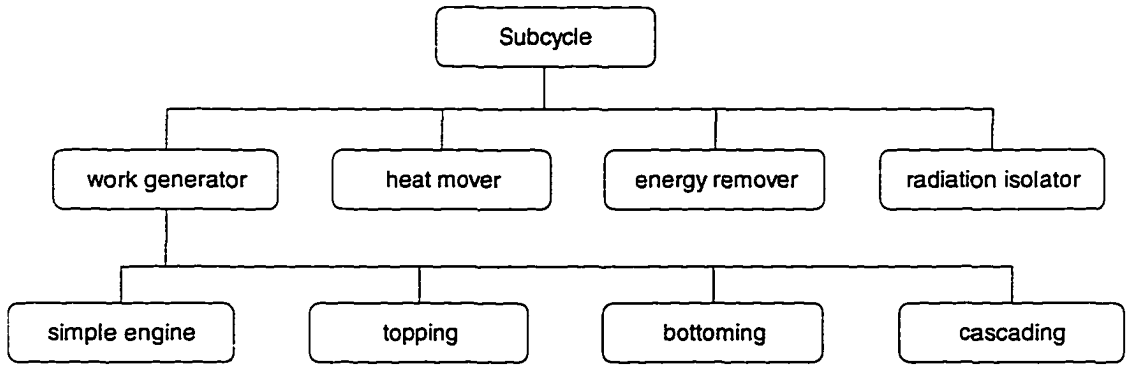


Figure 3.9 Subcycle roles

working fluid leaves the last turbine at a high temperature; gas-turbine engines are good candidates for topping subcycles. Rather than eject this heat to the atmosphere, a bottoming subcycle will utilize it to generate additional power, most likely via a vapor cycle. Finally, a cascading cycle would accept heat from a topping cycle and eject it to a bottoming cycle (in practice, there are few cycles comprised of more than two power coupled subcycles). Mercury or liquid sodium might be used in the topping cycle in this case.

A subcycle acting as a heat-mover is transporting heat from one location to another. A home heating system employing steam radiators might have such a subcycle for transporting heat from the central boiler throughout the building, or a heat-mover subcycle might be coupled to a power plant to provide district heating; some cities in Scandinavia utilize such systems to keep streets free of snow. Because a subcycle acting in this capacity isolates the working fluid from that of the power- or heat-producing subcycle, such a design enables the two subcycles to operate with different working fluids, or at substantially different pressures. Any contaminants that might infiltrate a dispersed heat-moving cycle cannot affect the cycle to which it is coupled, thereby preserving any turbomachinery present on that cycle.

Finally, a radiation isolator acts to contain the radioactivity of a reactor-driven powerplant to as small an area as possible, typically comprised of the reactor vessel itself, some pumps, and a closed heat-exchanger. Such isolation prevents the rest of the plant from becoming radioactive.

Aggregate Devices The final representation of locality is the aggregate device. In the cycle in Figure 3.10 we have three turbines in series producing the work output. If we consider these as a single device, then we can generate a simpler, functional topology for the cycle that facilitates comparison of cycles. Aggregate devices of non-opposing types can be interleaved, as indicated by the aggregate compression and heating processes. However, an expansion device will terminate a aggregate compressor, and a cooling device will terminate an aggregate heater. Thus it is possible for cycles to contain more

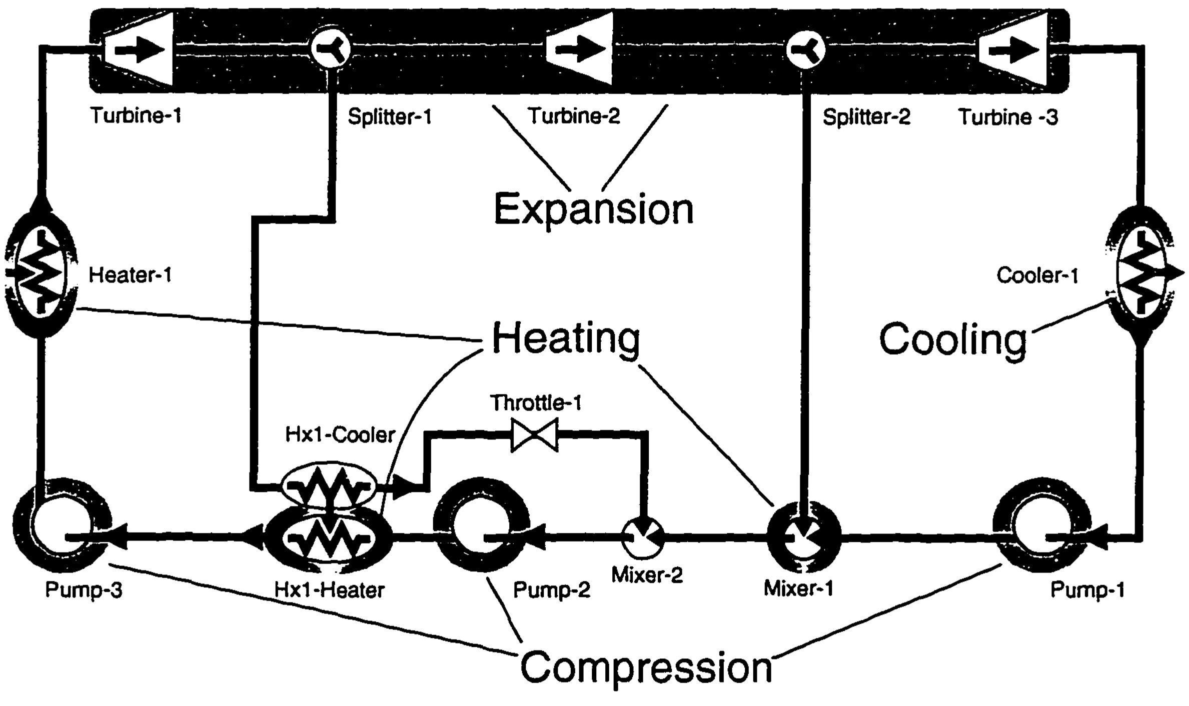


Figure 3.10 Aggregate devices

than one aggregate device of any type.

Aggregates are also useful in simplifying cycles that contain devices in parallel. Designers place turbines in parallel in order to keep the maximum turbine diameter to a minimum. Parallel turbines are aggregated into a single *pturbine* aggregate. Finally, TNT recognizes another type of aggregate called a *multi-port-chamber*, which is formed by connecting a mixer to a splitter. Multi-port chambers may act as *steam-drums* in heat-engines or as *flash-tanks* in gas-liquefying systems.

Steam-drums consist of a mixer connected to a splitter, and are used to separate steam from boiling water. The mixer of a steam drum plays the role of an open-heater, while the splitter plays the role of a flash-chamber, with saturated liquid exiting one outlet and saturated vapor the other. Steam-drums address the third design goal, preserving the system, by providing a reservoir of high-energy fluid for reacting to sudden changes in load and by enabling the precipitation of contaminant solids in the working fluid. Should such solids become entrained in the working fluid flowing through the high-energy portions of the cycle, such as the turbines, significant fouling and damage to the turbine blades may occur. Cycles 14 and 18 in Appendix B contain steam drums.

Flash-tanks provide the reciprocal function in refrigeration systems, cooling a hot gas by bubbling it through a cool liquid. Cycle 27 in Appendix B contains two flash tanks, one formed by MXR-2 and SPL-1 and the other by MXR-3 and SPL-2.

```
(defPlan <plan-name> {<plan-abstraction>}*
  <comment>
  (:goal <list-of-goals-achieved>
  :NL-phrase <passive natural language descriptor>
  :key-roles <list of roles central to plan>
  :conditions {<list of conditions for plan>}*
```

Figure 3.11 Syntax of defPlan form

3.6 Plans

Whereas roles represent inferences about function at the component level, *plans* represent functional inferences at the system level. Plans are patterns of function that have been abstracted and reified. They achieve one or more of the rational-designer teleological goals. As with roles, we organize ancillary domain knowledge around plans; plans also form the basis for the functional descriptions of cycles presented in Chapter 3. For example, descriptive knowledge of use to a coach may be associated with a particular plan. Unlike roles, plans are not mutually exclusive, so several may be active for a given cycle.

Plans are represented via the defPlan form, as shown in Figure 3.11. The *plan-name* provides instrumentation and debugging information. The optional *plan-abstraction* enables us to mark plans as specializations of more general plans, a fact that is incorporated into the generation of the cycle's explanation. The *comment* and *NL-phrase* provide the raw material for the generation of explanations. The *goal* of the plan is a list of all rational-designer goals that the plan achieves. The *key-roles* are those roles that are central to the plan, and finally the *conditions* are those facts that must be true

```
(defPlan direct-regenerate-with-minimal-delta-T (direct-contact-regenerate)
  "Direct contact is the most efficient form of heat exchange, but a large
   temperature difference in the streams will lead to irreversibilities.
   Increasing the pressure and intercooling a steam bleed can convert it to a
   dry saturated gas at the same temperature as the wet saturated liquid to be
   preheated"
  :goal (:increase-efficiency)
  :NL-phrase "superheated fluid bled from the turbine is saturated then is
              directly mixed with the boiler feed liquid"
  :key-roles ((role ?mxr open-heater ?bp))
  :conditions
  ((role ?intrclr intercooler ?prob)
   (hx-cooler ?intrclr ?clr-in ?clr-out)
   ((bleed-path ?bleed-valve ?devs)
    :test (and (member ?mxr ?devs)
               (member ?intrclr ?devs)))))
```

Figure 3.12 Example of a plan

prior to instantiation of the plan. Key-roles are distinguished from conditions to provide greater focus in explanations. The explanations shown in Appendix B are generated from instantiated plans.

Figure 3.12 shows an example of a plan form that describes a plan to achieve regeneration (preheating the boiler feed-fluid in order to increase the average temperature of heat-injection, and hence the cycle's thermal efficiency) in a manner that minimizes thermal inefficiencies. Note that this plan is a specialization of *the direct-contact-regenerate* plan, which only requires that hot working fluid be mixed in a mixer acting as an open heat-exchanger with working fluid on its way to the boiler. The key role in this case is an open-heater, where the actual regeneration takes place. The conditions specify that this plan is only active when the bleed-path that feeds the mixer contains a closed heat-exchanger acting as an intercooler. Because the intercooler role is defined with respect to compressors (an intercooler cools a gas being compressed, increasing efficiency by reducing the compression work necessary), we need not explicitly mention in the conditions the compressors that are an integral part of this plan. Appendix C lists all the plans in the CARNOT knowledge base.

4
REASONING

TNT infers function from structure in three steps, as shown in Figure 4.1. The first step consists of a topological parsing of the cycle in which TNT identifies teleological patterns (fluid-loops, steam-drums, bleed-paths) and subcycle structural elements, infers the cycle's type, and determines the downstream ranges of influence for each device. In the second step relevant evidence is instantiated and roles are inferred. In the third and final step, TNT identifies plans that achieve particular goals. This chapter provides a detailed description of these three steps.

```
Infer-function
1. Analyze cycle topology
2. Loop until no changes in evidence sets
    For each device D
            For each role R of D
                Update evidence set for R
                Propagate evidence for R
            Accept most likely role for D
3. Instantiate applicable plans
```

Figure 4.1 Reasoning algorithm

4.1 Step One: Topological Analysis

Topological analysis consists of summarizing certain aspects of the input structural description, identifying fluid-loops, subcycles, and influence-ranges, and determining the cycle type, as shown in Figure 4.2. Of these steps, fluid-loop identification is by far the most involved.

```
Analyze-topology
1. Summarize structural components
2. Identify fluid loops
3. Identify subcycles
4. Identify ranges of influence
5. Identify cycle type
```

Figure 4.2 Topological analysis algorithm

4.1.1 Summarizing Structural Information

Explicitly enumerating the constituents of device-type sets (e.g., the set of all turbines) sanctions several efficient local inferences about function. In particular, sets consisting of a single device (e.g., a single heater) provide inferential leverage. Recall that, by the Second Law of thermodynamics, all thermodynamic cycles must operate between two reservoirs of differing temperature. As we noted in Chapter 3, this requirement means that all cycles must have at least one heater and one cooler, because these devices are the only ones in our structural representation that have heat-paths connecting the cycle to the environment. A singleton heater in a cycle identified as a heat-engine is perforce a heat-injector, whereas if the cycle is identified as a refrigerator, it must function as a heat-absorber.

4.1.2 Parsing Cycle Topology into Fluid Loops

Fluid-loops are closed circuits that constitute paths over which a bit of working fluid could flow during steady-state operation of the cycle. As such, they respect the directionality of flow that is part of the structural representation of the cycle. TNT also

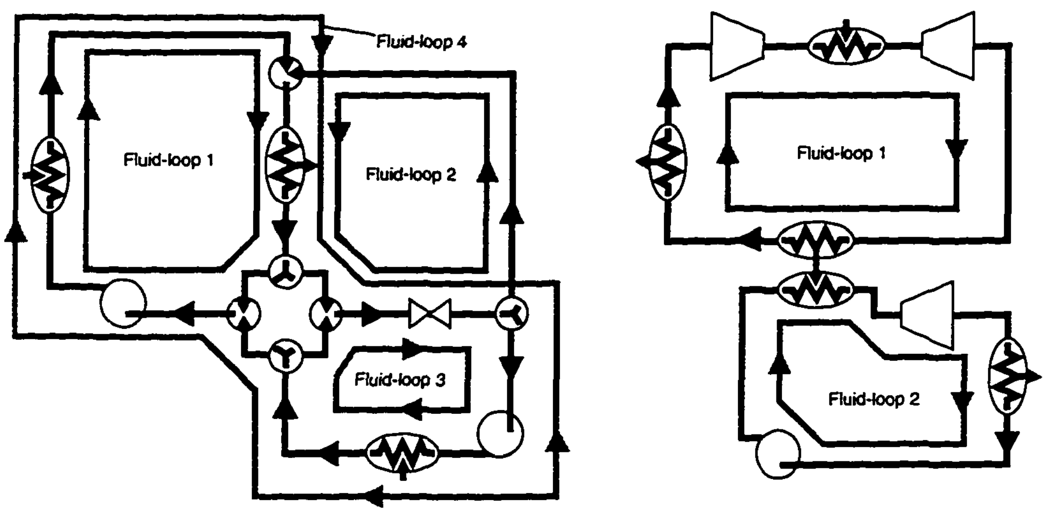


Figure 4.3 Examples of fluid loops

considers each potential path originating at a source and terminating at a sink to be a fluid-loop.

Fluid-loops may or may not overlap; Figure 4.3 illustrates both cases. TNT uses fluid-loops for three purposes; (a) to identify teleological patterns (e.g., bleed-paths), subcycles, and influence-ranges, (b) to infer the type of system (either heat-engine or refrigerator), and (c) to test for locality among components in the course of identifying roles.

The algorithm for fluid-loop identification, shown in Figure 4.4, starts at all mixers, splitters, and heat-exchanger halves and follows the structural linkages downstream until the starting device is encountered, or, if none of these devices are present, chooses an arbitrary starting point. These starting points

```
Identify-fluid-loops
  For each start-point SP in Fetch-start-points
    Until Fluid-loop-identification-complete
      For each fluid-loop FL in Trace-fluid-loops(SP)
          Assert-fluid-loop!(FL)

Fetch-start-points
    For each device D
      When Splitter?(D) or Heat-exchanger?(D)
        Collect D

Trace-fluid-loops(start)
    Find-fluid-loops(start,{ },{ },{ })

Find-fluid-loops(start,route,stuffs,route-start)
    For each device in Fetch-outlet-devices(start)
      Let common-stuff = Find-common-stuff(start,device)
        If Source-to-sink-route?(route-start,device) then
          Let stuffs = stuffs & common-stuff
          Make-fluid-loop-proposition(stuffs,device,route)
        Elseif Outlet-connected-to-inlet?(device,route-start) then
          Let stuffs = stuffs & common-stuff & Find-common-stuff(device,route-start)
          Make-fluid-loop-proposition(stuffs,device,route)
        Elseif Member(device,route) then
          do nothing
        Else
          If Contains-devices?(route) then
            Let route = device & route
          Else
            Let route = device & start
          Let stuffs = common-stuff & stuffs
          If route-start = { } then
            Let route-start = start
          Find-fluid-loops(device,route,stuffs,route-start)
```

Figure 4.4 Algorithm for finding fluid-loops

comprise the set of devices that complicate topologies; without them our representation constrains the topology to a single simple loop consisting of all devices in the cycle. Searching from each of these devices ensures that we find all possible fluid-loops, at the cost of potentially redundant calculations. To minimize such redundancy, the predicate fluid-loop-identification-complete? returns true when all devices and all connecting stuffs are members of at least one fluid-loop, and this causes identify-fluid-loops to terminate.

When trace-fluid-loops encounters a splitter, it caches the path up to that splitter and one of the splitter's outlets for further processing and continues the search down the other outlet of the splitter. Each such partial fluid-loop is later completed by recursive calls to trace-fluid-loops.

For the purpose of analyzing fluid-loops, it proves useful to break the loop at a particular point, and consider that point the start of the loop. We break fluid-loops immediately upstream of the first compressing device to be found after the last expansion device, because the working fluid is closest to the conditions of the surrounding environment here, and this provides a convenient criterion grounded in the domain. Automobile engines, which operate in a so-called "open" cycle, break the cycle at this point, taking in working fluid (i.e., air) immediately prior to compressing it, and exhausting it immediately after the power stroke.

4.1.3 Identifying Subcycles

Subcycles are the largest substructures that we represent. In many cases the entire cycle will consist of a single subcycle. The system on the right side of Figure 4.3 contains two subcycles. The salient feature of this and any cycle that contains multiple subcycles is

```
Identify-subcycles
  Let all-fluid-loops = all fluid-loops identified in cycle
  Let disjoint-fluid-loops = Pop(all-fluid-loops)
  For each fluid-loop FL in all-fluid-loops
        For each disjoint-loop DL in disjoint-fluid-loops
            If FL shares devices with DL then
                Replace DL with Union(FL,DL)
                Else Adjoin(FL,disjoint-fluid-loops)
  Return disjoint-fluid-loops
```

Figure 4.5 Algorithm for finding subcycles

that the only allowed connection between any subcycles is the heat-path of a heat-exchanger; the working fluids in any two subcycles will never mix.

The algorithm for the identification of subcycles is shown in Figure 4.5. We iterate over all identified fluid-loops, merging all that share components.

4.1.4 Finding Downstream Ranges of Influence

A range of influence is defined for each component, and consists of all those downstream components that receive working fluid whose parametric values have been set at least in part by the action of the device in question. For example, the range of influence for a pump consists of all downstream components up to but not including the first throttle or

Table 4.1
Criteria for ending ranges of influence

Device Type	Influence Range Terminator
Pump, Compressor	Turbine, Throttle
Turbine, Throttle	Pump, Compressor
Heater, Hx-heater, reactor	Cooler, Hx-cooler
Cooler, Hx-cooler	Heater, Hx-heater, reactor
Splitter	Mixer
Mixer	Splitter
Source	Sink
Sink	Source

```
Find-influence-ranges
   For each device D in all devices of the cycle
      Find-influence-range(D,D,{ })

Find-influence-range(device,start,result)
   For each outlet-device OD immediately downstream of device
      If Member(OD,result) then
         Return result
      Elseif Range-terminator?(OD,start)
         Return Adjoin(OD,result)
      Else
         Find-influence-range(D,start,Adjoin(OD,result))
```

Figure 4.6 Algorithm for finding ranges of influence

turbine, as these are the only two types of devices that can reverse the pressure-increasing effect a pump has on the working fluid. Likewise, the range of influence of a turbine extends up to the first downstream pump. Ranges of influence include all possible paths of flow from the original device. Table 4.1 shows the criterion used for ending the range of each type of device.

We derive ranges of influence for each device by traversing the cycle's fluid loops until we encounter devices of the opposite type, accumulating those devices on the paths to the terminators. This tends to break the cycle into groups that correspond to the four canonical cyclic processes (compression, heating, expansion, and cooling). Figure 4.6 shows the algorithm.

4.1.5 Determining Cycle Type

Once all fluid-loops have been found, deriving the cycle-type is a global operation, illustrated by the flow chart in Figure 4.7. This decision process is applied to each fluid-loop in the cycle, because it is possible to have a refrigeration cycle that contains a power-generating fluid-loop within it. Refrigeration loops are preferred over heat-engine loops in the determination of cycle-type, because refrigerators may contain heat-engines to provide the power necessary to move heat from one location to another, but heat-engines have no need nor use for refrigeration loops.

In the first step the cycle-type-finding algorithm checks for the presence of turbines in the cycle as a whole. Since these are the only devices capable of producing power, their absence is definitive proof that the cycle is not a heat-engine, and hence by exclusion must be a refrigerator. Should we find a turbine in the cycle, the next step tests each fluid-loop for the canonical heat-engine ordering of devices (compress, heat, expand, cool). If all fluid-loops exhibit such an ordering than we infer the cycle is a heat-engine. If not, we attempt to reach the same conclusion by ruling out the possibility that any loop is a refrigeration loop, this time by using the canonical ordering for refrigerators (compress, cool, expand, heat). If this fails we infer the cycle is a refrigerator.

If the cycle is identified as a heat-engine, we then identify the fluid-loop (or possibly fluid-loops, in the case of parallel turbines) within each subcycle that contains the greatest number of turbines as the *primary fluid-loop* for that subcycle, as this is the fluid-loop that would produce the greatest power.

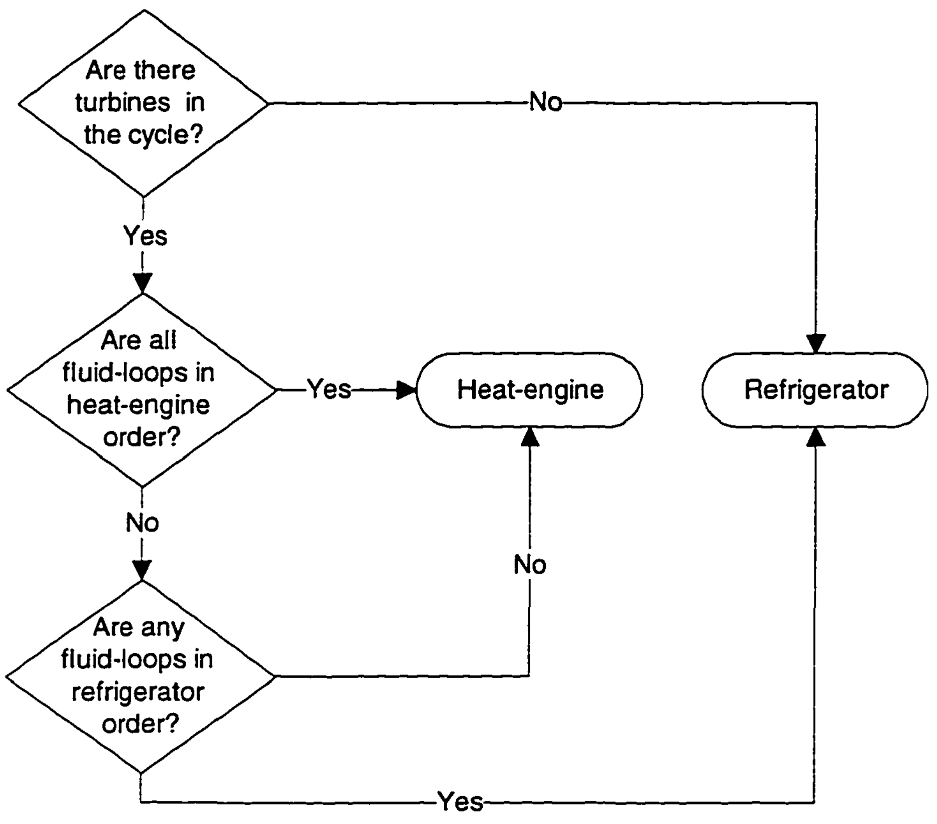

Are there turbines in the cycle?
No
Yes
Are all fluid-loops in heat-engine order?
Yes
Heat-engine
Refrigerator
No
No
Are any fluid-loops in refrigerator order?
Yes

Figure 4.7 Determining cycle type

The primary fluid-loop is then used to identify bleed paths, which originate at splitters interleaved among the turbines of the primary-fluid-loop (and hence playing the role of bleed-valves) and terminate in mixers that lie on the primary-fluid-loop.

4.2 Step Two: Role Inference

The TNT evidential reasoning process for inferring roles directly from topological information is probabilistic. This eliminates the need for a simulation of the system's behavior, thereby greatly improving efficiency and enabling the construction of a modular reasoning system that may be combined with other styles of reasoning to solve complex problems. We defer further discussion of the relative merits of probabilistic reasoning to Chapter 7, and focus in this chapter on how the reasoning algorithm works.

The role inference algorithm alternates between local propagation of facts via logical dependencies and a global operation that closes sets of evidence, one set for each potential role of each device. Once the evidence sets are closed, we apply a standard technique for Bayesian updating within a hierarchy described in (Pearl 1988), which in this case is the hierarchy of roles described in Chapter 3. Evidence for and against particular roles is propagated through the hierarchy. At the conclusion of this propagation, the most likely role is assumed to be true, and the algorithm iterates until no new facts are introduced and there are no changes in the membership of the evidence sets.

In the following section we provide an overview of Bayesian probability theory and how it applies to teleological reasoning. We will then describe the inference algorithm in detail. Readers familiar with Bayesian theory may wish to skip this section.

4.2.1 Bayesian Probability Theory

The heart of Bayesian inference is the formula

$$P(H|e) = \frac{P(e|H)P(H)}{P(e)} \qquad \textbf{Equation 4.1}$$

which enables us to invert conditional probabilities. A conditional probability, such as P(H|e) is read as “the probability of hypothesis H given evidence e”. The mathematical definition of a conditional probability is

$$P(H|e) = \frac{P(H \wedge e)}{P(e)}$$ **Equation 4.2**

To wrap some context around this, let's assume that our hypothesis H is *Heater-1 is playing the role of a heat-absorber*, and our evidence e is *cycle containing Heater-1 is a refrigerator*. If we further assume that all cycles are either refrigerators or heat-engines, and all heaters are either heat-absorbers or heat-injectors, then we can construe the probability of the conjunction of H and e as a particular assignment of values to the random variables Cycle-Type and Heater-Role. In this simple case, we can completely specify all such assignments in a joint probability distribution table, as shown in Table 4.2.

Each cell of this table represents a unique state of the world. For example, we can determine from this table that *P(H $\wedge$ e)* is 0.35, that is, that there is a 35% chance that a random cycle will be a refrigerator and will have a heater acting as a heat-absorber. We can also determine from this table that the P(e), the chance that a randomly chosen cycle is a refrigerator is 0.45, the sum of the first row. The conditional probability that Heater-1 is acting as a heat-absorber given that the cycle is a refrigerator is therefore 0.35/0.45 or

Table 4.2
Joint probability distribution

Cycle Type	Heater Role		
	Heat-absorber	Heat-injector	Sum
Refrigerator	0.35	0.1	0.45
Heat-engine	0.05	0.5	0.55
Sum	0.4	0.6	1.0

0.78.

Note that this table completely specifies all possible states of the world. For a trivial example such as this, in which there are only two random variables (cycle-type and heater-role), we could simply use this table. Unfortunately, this approach doesn't scale, since in the general case there would be 2^n entries in the table for n Boolean variables. This is the motivation for reasoning in terms of conditional probabilities, because they enable the calculation of just that part of the joint distribution we need.

For the purpose of recording knowledge about a domain, it is preferable to do so in terms of the probability of a symptom given the underlying cause, because this relationship is generally invariant with respect to changes in the world. However, when confronted with a symptom (i.e., something we can sense directly), we want to be able to calculate the probability of the underlying cause given the system. The Bayesian formalism allows us to make the conversion from one conditional probability to the other.

To further elucidate this point, let's consider the domain of medical diagnosis.[1] Suppose that there are three diseases that typically cause a circular rash, blotfilism, wring fever, and scadamilitis. If we have a domain expert encode her knowledge about these diseases in terms of the circular rash, we might end up with three rules:

- If *circular-rash* then *blotfilism* with probability 0.73
- If *circular-rash* then *wring fever* with probability 0.99

[1] This discussion is a synthesis of the material presented in several texts on Bayesian probability theory; the most germane are (Pearl 1988) and (Russell and Norvig 1995).

- If *circular-rash* then *scadamilitis* with probability 0.64

Given these probabilities, if we are confronted with a new patient displaying a circular rash, our diagnosis would always be wring fever, since that is the most likely. The drawback of this approach is that these probabilities were formed from a particular set of experiences. Suppose that our domain expert was a doctor from Zaire, where wring fever is especially common; another domain expert, say from Chicago, might assess the probabilities entirely differently, leading to different diagnoses. Even more troubling, suppose that there is a local outbreak of scadamilitis; the medical staff in the local emergency room would quickly incorporate this piece of knowledge about the world into their diagnoses. Common sense would dictate that scadamilitis should be the first disease to suspect, even though a reasoning system based on the above rules would continue to proffer diagnoses of wring fever.

Now let's suppose that we seek out domain experts in each of these diseases and ask them to formulate rules about how often the circular rash may be expected to appear, given that the patient is suffering from the disease. We might derive the following rules:

- If *blotfilism* then *circular-rash* with probability 0.77
- If *wring fever* then *circular-rash* with probability 0.85
- If *scadamilitis* then *circular-rash* with probability 0.69

This knowledge is independent of the current state of the world, since it represents a causal property of the disease; blotfilism will continue to produce a circular rash 77% of the time whether or not there is an epidemic of scadamilitis. There is evidence that human knowledge is organized more along these lines; for example, Tversky and Kahneman (1982) have shown that medical practitioners prefer to assess probabilities in this manner.

Now, however, we have to apply the Bayesian formula to determine the probability of a given patient suffering from any of these diseases. Let's suppose that we're interested in the probability that our patient has contracted blotfilism. We know that P(circular rash|blotfilism) = 0.77, and we'd like to determine P(blotfilism|circular rash). To use Equation 4.1 we need to know P(blotfilism) and P(circular rash). Let's suppose that blotfilism is a rare disease, striking only 1 in 10,000 people. Circular rashes, however, are relatively common, so we can expect 1 in 100 people to have a circular rash. Therefore we have:

$$P(Blotfilism|Rash) = \frac{0.77 \times 0.0001}{0.01} = 0.0077$$

So the probability that our patient is suffering from blotfilism is still very small, but at the same time, it has increased from 1 in 10,000 to about 1 in 130. Unlike our prior situation, in which our reasoner is insensitive to changes in the environment, this calculation directly incorporates such information. For example, let's suppose there is an epidemic of blotfilism, raising the probability from 1/10,000 (our prior probability) to 1/200. In this case, the probability of our patient suffering from blotfilism would be:

$$P(Blotfilism|Rash) = \frac{0.77 \times 0.005}{0.01} = 0.385$$

Our patient's chances of having blotfilism have now risen from 0.0077 to 0.385, an increase of 50 times. Notice that our original assessment of the linkage between circular rashes and blotfilism remains invariant, but the rest of the formula enables us to incorporate knowledge about the current state of the world, in terms of the prevalence of blotfilism and circular rashes.

Knowledge about teleology in TNT is expressed in terms how likely a particular conjunct of facts about the system (typically topological propositions) is, given that we know a particular device is playing a certain role. When we propagate this evidence, we use the Bayesian formalism to convert this type of knowledge into a probability that the role is true, given the conjunct of facts. Probabilistic knowledge expressed in this manner depends primarily on the expert's model of the domain, and tends not to be biased by the expert's experience with particular cycles.

As is the case in the medical domain, knowledge expressed in terms of topological evidence (i.e., the symptom) implying a particular role (i.e., the disease), is fragile. It also turns out that providing such estimates is much harder, because it is dependent on the composition of the cycle population being processed. For example, suppose that a particular configuration that enables the use of a mixer as a jet-ejector only occurs in cogeneration systems used in medium-sized industrial applications. An expert on cogeneration systems would be likely to overestimate the probability, whereas an expert on cryogenic systems would be likely to underestimate the probability. Moreover, a knowledge base built on such estimates, even if carefully calibrated for the relative occurrence of different cycle-types in the universal population of cycles, would be unable to compensate for an epidemic of cogeneration cycles in the input.

Probabilistic knowledge in TNT is expressed in terms of likelihoods, which are ratios of conditional probabilities. A likelihood is defined as

$$\lambda_h = \frac{P(e|H)}{P(e|\neg H)}$$ **Equation 4.3**

where a λ greater than 1 indicates confirmation and a λ less than 1 indicates suppression of the hypothesis. The range of confirmation is thus [1,∞], whereas the range of suppression is [0,1].

Our use of likelihoods rather than absolute probabilities provides a better fit of the theory to the intuitive notions one brings to bear when assessing probabilities. Artificial Intelligence researchers in the mid-1970s, notably Buchanan and Shortliffe (1984), rejected Bayesian inference in favor of other formalisms, such as certainty factors and the Dempster-Shafer theory. In particular, as Pearl notes,

> [they] observed that experts who provide rules such as $e \rightarrow h(0.7)$ (to read "Evidence e suggests hypothesis h to a degree 0.7") may well agree that P(h|e) = 0.7, but became uneasy when confronted with the logical conclusion that P(¬h|e) = 0.3. The experts would claim that the observations were evidence (to degree 0.7) in favor of h and should not be construed as evidence (to degree 0.3) against h. Such apprehensions were part of the reason the developers of early expert systems abandoned probabilistic reasoning and adopted less orthodox calculi. (Pearl 1988, p343).

Pearl argues that the underlying reason for this apprehension was the mistaken attempt to obtain absolute probabilities, rather than relative likelihoods from the researchers.

> The reason for the experts' unease with $P(\neg h|e) = 0.3$ is that the phrase "evidence in favor of a hypothesis" leads us to expect an increase in the probability of the hypothesis from $P(h)$ to $P(h|e)$...accompanied by the appropriate decrease in $P(\neg h|e)$. On the other hand, if $P(\neg h|e) = 0.3$ is viewed as the final product of the rule $e \rightarrow h(0.7)$, it may often mean an increase in $P(\neg h)$, say from 0.01 to 0.3, and that would violate the spirit of the rule.
>
> The likelihood ratio formulation has a built-in protection against such confusion because it conveys only change information; evidence in favor of h is encoded by $\lambda > 1$ and will always produce $P(h|e) > P(h)$, while evidence opposing h is encoded by $\lambda < 1$ and will always result in $P(h|e) < P(h)$ (Pearl 1988, p343).

4.2.2 Representing Teleological Knowledge

TNT specifies that knowledge is encoded in the form of evidential tests. Each test is specific to a particular role. A test succeeds if the conjunction of condition facts it specifies are found to be true. Success causes the instantiation of an evidence proposition, which is used to update the prior probability of the role. The syntax of evidential test forms is presented in Figure 4.8.

The *test-name* enables instrumentation, such as introducing noise into the likelihoods or disabling particular tests, and facilitates debugging. The *role-tested-for* is the name of the role for which evidence will be asserted. The *likelihood* is a number in the range [1,∞]. The *device-proposition* is the form that the test must find in the database

```
(defEvidence <test-name> <role-tested-for> <likelihood> <device-proposition>
  <comment>
  {<condition>*|(<condition> :test <test-function>)})
```

Figure 4.8 Syntax of defEvidence form

```
(defEvidence Pmp-Tst6 flash-preventer 6.0 (pump ?pmp1 ?in ?out)
   "A pump feeding an open heat-exchanger with another pump downstream of both
    is probably intended to prevent the heat-exchange from vaporizing the
    working fluid"
  (cycle-type :heat-engine ?reason)
  (downrange ?pmp1 ?pmp-range)
  ((role ?mxr open-hx ?prob)
   :TEST (member ?mxr ?pmp-range))
  (downrange ?mxr ?mxr-range)
  ((pump ?pmp2 ?p2-in ?p2-out)
   :TEST (and (member ?pmp2 ?pmp-range) (member ?pmp2 ?mxr-range))))
```

Figure 4.9 Example of an evidential test

prior to firing, and thus acts as a relevance criterion. The *comment* provides a natural-language summary of the intent of the test, and is incorporated into the dependency record, enabling a limited natural language explanation of inferences.

The rest of the test form consists of an arbitrary number of *conditions*, which may be in one of two forms, a simple-pattern or a pattern-plus-test-function. The test-function is an arbitrary expression that may take as arguments any of the pattern variables in the scope of the test form. In the absence of a test-function, the pattern must unify with a particular proposition in the database. When a test-function is present, the pattern must first unify, and then the test-function must return true.

An evidential test form is shown in Figure 4.9. Our implementation utilizes a pattern-directed inference system, so symbols with an initial question mark (e.g., `?reason`) should be construed as pattern variables subject to partial unification. The pattern `(pump ?pmp ?in ?out)` therefore would match the proposition `(Pump Pump-1 S10 S20)`, with `?pmp` bound to `pump-1`, `?in` bound to `S10`, and `?out` bound to `S20`.

Because we have defined a type abstraction hierarchy for all devices, tests are not limited in application to particular types of devices. For example, if the test in Figure 4.9 applied to both pumps and compressors, we would replace the `(pump ?pmp1 ?in ?out)` form with a `(genus ?dev work-consumer)` form.

Although likelihoods are defined over the range [0,∞], with likelihoods less than one acting to decrease the prior probability, we have found that it is easier to write and compare evidence test forms uniformly in terms of the range [1,∞], so we have added some syntactic sugar that enables this for suppressive evidence. Figure 4.10 shows a test that lowers the probability that a pump is playing the role of a flash-preventer. The (:not...) syntax causes TNT's parsing mechanism to take the reciprocal of the likelihood.

An evidential test instantiates an evidence proposition of the form shown in Figure 4.11 when the conjunct of the device-proposition and all conditions plus any applicable tests are simultaneously true. Each evidence proposition is the consequent of the conjunct of the device-proposition and all conditions, and as such is dependent on these propositions to remain true. Because role propositions are permitted as conditions in tests (as in Figure 4.9), it is quite possible that an evidence proposition asserted with a

```
(defTest Pmp-Tst4 (:not flash-preventer) 4.0 (pump ?pmp1 ?in ?out)
  "Pumps without other pumps or turbines downstream of them are probably not
   intended to prevent the working fluid from flashing"
  (cycle-type :heat-engine ?reason)s
  ((downrange ?pmp1 ?pmp-range)
   :TEST (and (notany #'pump? ?pmp-range) (some #'turbine? ?pmp-range))))
```

Figure 4.10 Example of a suppressive test

```
(evidence <device-name> <device-role> <likelihood> <test-name>)
```

Figure 4.11 Syntax of an evidence proposition

true label may at a later step in the processing become unknown, if a role that it depends on is no longer believed. This is the primary source of the tight interleaving between locality propositions and roles that we illustrated in Figure 3.1.

Evidential tests typically test for structural configurations that are indicative of particular roles. Nonetheless, there is nothing in their definition to preclude one from writing tests predicated on other information. To explore this possibility, we have built into CARNOT a transitive inequality reasoner that attempts to deduce pressure or temperature differentials across the inlets of mixers. Should it find one, there are evidential tests that instantiate evidence for jet-ejectors or open heat-exchangers. (We present an example of such reasoning in Section 4.2.3.1 below). The inference mechanism, therefore, is not specific to reasoning from structure to function. Appendix C lists all evidential tests in CARNOT's knowledge base.

4.2.3 How Plausible Reasoning Works

Plausible inference, as we have noted, alternates between local propagation of logical dependencies, and a global operation that generates sets of evidence. The algorithm is summarized in Figure 4.12. Local propagation will cause any evidential test whose conditions are met by facts known to be true to fire, instantiating an evidence proposition.

```
Update-role-beliefs
    Do-local-propagation
    Loop until no changes in evidence sets
        For each device D
            For each role R of D
                Update evidence set for R
                Propagate evidence for R
            Accept most likely role for D
        Do-local-propagation
```

Figure 4.12 Algorithm for role inference

The global set-closing operation then enables us to determine the most likely role based on known evidence.

The initial propagation instantiates role propositions for each role of each device. The knowledge base contains prior probabilities for each role, which indicate the chance of that role being true given no information about the input cycle. For example, a turbine is most often used as a work-source, but in some cryogenic applications, the designer may take advantage of the large temperature drop that occurs across a turbine, so its role in this case would be to cool the fluid. Without knowing anything in advance about the input cycle, the best one can do is rely on general domain knowledge, which in this case indicates that turbines act as work-sources 95% of the time; this is the prior probability for a work-source role. Since roles partition the space of functions, the prior for a turbine acting as a fluid-cooler role is necessarily 0.05.

The firing of an evidential test establishes a logical clause in which the truth of evidence proposition depends on the conjunct of the test's conditions; should one or more of those conditions lose support at a later stage in the processing, the evidence proposition will no longer be believed.

```
Update-role-beliefs
    Do-local-propagation
    Update-evidence-and-roles

Update-evidence-and-roles
    For each entity E in cycle entities
        Update-range-of-influence for E
        Update-evidence-set of E
        For each piece of evidence Ev in updated evidence set for E
            Update-belief(Ev)
    Update-role-truth-labels for all entities

Update-role-truth-labels
    For each entity E in entities
        Unless Most-likely-roles(E) = True-roles(E)
            For each role R in True-roles(E)
                Set-truth-label(R) = unknown
            For each role R in Most-likely-roles(E)
                Set-truth-label(R) = true
    Do-local-propagation
    When New-facts-instantiated? or Change-in-evidence-sets?
        Update-evidence-and-roles

Change-in-evidence-sets?
    For each device D
        For each role R of device D
            Unless Fetch-evidence-propositions(R) = Propositions-of(Evidence-set-of(R))
                Return true
```

Figure 4.13 Belief-revision algorithm

Once local propagation has reached quiescence, we enter a loop which iterates over each role proposition of each device, collecting all evidence facts known to be true and comparing this set to a proposition about the set of evidence bearing on this role.

Should there be a difference, we update this set proposition and revise our belief in the role, based on the new evidence.

When this iteration is complete, we accept the most likely role for each device as the true role. Because role propositions may appear in evidential test conditions, we must therefore do another local propagation, and if there are either new propositions instantiated or changes in the extant evidence sets, we must repeat the above iteration. In general this algorithm reaches quiescence in from one to twelve iterations.

In theory it is possible to fail to reach quiescence for a given knowledge base. It is possible to perform a static analysis of the knowledge base to detect the potential for such looping. Fortunately, domain knowledge tends to be represented in such a way that loops do not arise. We will discuss this further below, after we examine some of the finer points of the algorithm, which is shown in detailed form in Figure 4.13.

The first subtle point in this algorithm occurs in the call to update-range-of-influence in update-evidence-and-roles. Ranges of influence may change as new roles become believed. For example, a mixer is initially construed as a flow-join, which would allow the range of influence of any upstream device except a splitter to pass through it. However, should the inference mechanism construe the mixer as a jet-ejector, this will terminate the influence range for any upstream throttles and turbines. Likewise, should the mixer be construed as an open-heater or open-cooler, this would terminate the influence ranges of any upstream coolers or heaters (respectively) that originally incorporated the mixer.

The updating of evidence sets requires that we gather up all evidence propositions instantiated in the local propagation step and make a new proposition that this set constitutes all evidence bearing on the role in question. There are four possible outcomes; (a) there is no change, (b) one or more new pieces of evidence have been introduced, (c) one or more pieces of evidence are no longer believed, or (d) both (b) and (c) have occurred.

The actual updating of probabilities, as implemented in update-belief, is a direct application of Pearl's algorithm for evidential reasoning in taxonomic hierarchies (Pearl 1988). As such, we do not consider it part of the contribution of this work, and so only summarize it here (see Appendix A for more detail). It is a message-passing scheme, in which the directly affected node of the hierarchy passes excitatory messages to its parent and inhibitory messages to its siblings. This is necessary because the hierarchy partitions the space of function, so at each level of abstraction the probabilities of the roles must sum to 1.0. For example, suppose that we find evidence in favor of a heater playing the role of a fluid-heater. The probability that it is a heat-injector (the parent of fluid-heater) should therefore also increase, while the probability that it is a preheater or reheater (siblings of fluid-heater) should decrease. Finally, the probability that heater is a heat-absorber, the sibling of heat-injector, should decline.

When we find new evidence, we simply propagate the associated likelihoods, using the current probability of the role as the prior. The belief updating algorithm has the desirable property of enabling incremental composition of evidence.

In the case of one or more pieces of evidence losing support, a situation we term an *evidence set dropout*, we reset the belief of the role to the original value and repropagate all evidence. Although this is a bit profligate, the propagation of evidence is so fast that the algorithmic complexity required to back out the effect of the evidence no longer believed is not worth the gain in processing speed. We apply the same mechanism to the fourth case, in which there is both an evidence set dropout and new evidence.

Once we have completed this global operation over all devices, we have to do the local propagation and also check the state of the evidence sets. This is necessary because we may have just retracted belief in one role and asserted belief in a different role for a given device. Changes in the truth values of role propositions have the potential for three effects on working memory. First, a new role belief may enable an as-yet untriggered evidential test to fire, instantiating new evidence. Second, retraction of role belief may result in some evidence propositions losing support and dropping out of their evidence sets. And third, change in role belief may invalidate a particular range of influence, because ranges of influence are dependent on the conjunct of the roles of all devices in the range.

To determine whether or not working memory has reached quiescence, we call the procedures new-facts-instantiated? and change-in-evidence-sets?, which will return true if either new evidence facts are present in the database or if at least one evidence set proposition has dropouts or new members. Only when both of these predicates return false do we terminate the update of belief.

As we have noted, it is possible that, for a particular knowledge base, this algorithm will never reach quiescence. In practice, this has not proven to be a problem, due to the structure of the domain knowledge we have encoded. Although thermodynamic cycles are topologically cyclic, and hence in theory two devices could mutually influence one another, we have found that the knowledge we encoded (without consideration for this issue, as the final form of the updating algorithm evolved in parallel with the development of the knowledge base) does not exhibit such pathological mutual recursion, because it is in terms of downstream ranges of influence. For example, a flash-preventing pump requires the presence of a heater, a pump, another heater and a turbine downstream, in that order. Turbine and heater roles, however, generally do not depend on upstream devices, such as pumps.

The potential for not reaching quiescence led to the development of a static knowledge-base analyzer that can scan the knowledge base at compile time and indicate evidential tests that could cause the updating algorithm to cycle among several alternative construals. This analyzer checks each evidential test for a role proposition in its conditions. For each role proposition found, it examines all tests for or against that role. The analyzer recurs on each role found in each of these tests, and terminates when it either encounters rivals to the original role or when no new tests contain roles in their conditions. A rival role is either a sibling, or an abstraction or specialization thereof. Finding a rival role indicates that we have a situation in which the algorithm may not attain quiescence, because the original role is acting, indirectly, as evidence for a rival role. If this evidence is strong enough, then the original role may be retracted in favor of the rival. This will cause the rival to lose support, and the original to be reinstated, perpetuating the loop.

We have run the analyzer on our knowledge base and found no such loops. In fact, it turned out to be quite hard to deliberately construct such a loop with evidential tests that had at least the surface appearance of reasonableness.

4.2.3.1 Examples of Evidential Inference

Identifying a Flash Preventer. A pump acting as a flash-preventer is raising the pressure of the working fluid so that subsequent heating will not cause it to vaporize prior to reaching the primary boiler of the system. Flashing would cause downstream pumps to stall, interrupting the flow of working fluid to the boiler, which, in the case of a modern electrical power plant, could well melt.

Let us examine how a test would identify evidence for this role. The example test we presented in Figure 4.9 is shown in Figure 4.14, with its conditions numbered for reference. The first condition limits this test to heat-engines, since preventing flashing is not typically a problem in refrigerators. The second condition acquires the influence range of the pump in question. The downrange proposition contains a list of all devices in the downstream range of influence which will be bound to the pattern variable `?pmp-range`. Note that each condition establishes an environment in which its bindings and all those of prior conditions are present.

The third condition looks for a mixer that is currently believed to be acting as an open heat-exchanger and tests to determine whether or not this mixer is a member of the pump's downstream range of influence by searching the list bound to `?pmp-range`. The fourth condition acquires the mixer's influence range; this is needed because the final condition of this test is seeking a pump downstream of both the original pump and the open heat-exchanger. Flash-prevention is only necessary when the fluid passing through a pump is both heated and then pumped, in that order, downstream of the pump in question. By predicating the conditions of this test on the influence ranges of the devices in question, we ensure that no throttling, expansion, or cooling processes will occur

```
(defEvidence Pmp-Tst6 flash-preventer 6.0 (pump ?pmp1 ?in ?out)
   "A pump feeding an open heat-exchanger with another pump downstream of both
    is probably intended to prevent the heat-exchange from vaporizing the
    working fluid"
 1  (cycle-type :heat-engine ?reason)
 2  (downrange ?pmp1 ?pmp-range)
 3  ((role ?mxr open-hx ?prob)
     :TEST (member ?mxr ?pmp-range))
 4  (downrange ?mxr ?mxr-range)
 5  ((pump ?pmp2 ?p2-in ?p2-out)
     :TEST (and (member ?pmp2 ?pmp-range) (member ?pmp2 ?mxr-range))))
```

Figure 4.14 Example of a test for flash-preventer evidence

between any set of devices this test matches.

At the same time, however, this test does not prescribe a particular configuration of the components involved. In fact, we might have an arbitrary number of different components interleaved among these three, so long as none could possibly generate a cooling or expansion process.

Identifying a Open-Heater. Although most of the knowledge base is cast in terms of topological information, TNT is by no means limited to this form of reasoning. To illustrate an alternative form of reasoning, we have implemented a transitive inequality reasoner.

For the jet-ejector and open heat-exchanger roles of a mixer, a temperature difference across the inputs indicates an open heat-exchanger, while a pressure difference implies a jet-ejector. Thus when we find a mixer in the structural input, we express interest in finding inequalities in either pressure or temperature across the mixer's inputs.

We use the fluid-loops identified in the initial structural parsing of the cycle and the physical effect inequality information (see Section 3.3) to deduce an inequality statement via transitive reasoning for the parameters in which we have expressed interest. For example, in the cycle fragment shown in Figure 4.15, the fact that the mixer is an open heat-exchanger can be qualitatively deduced. The transitivity reasoning proceeds as follows:

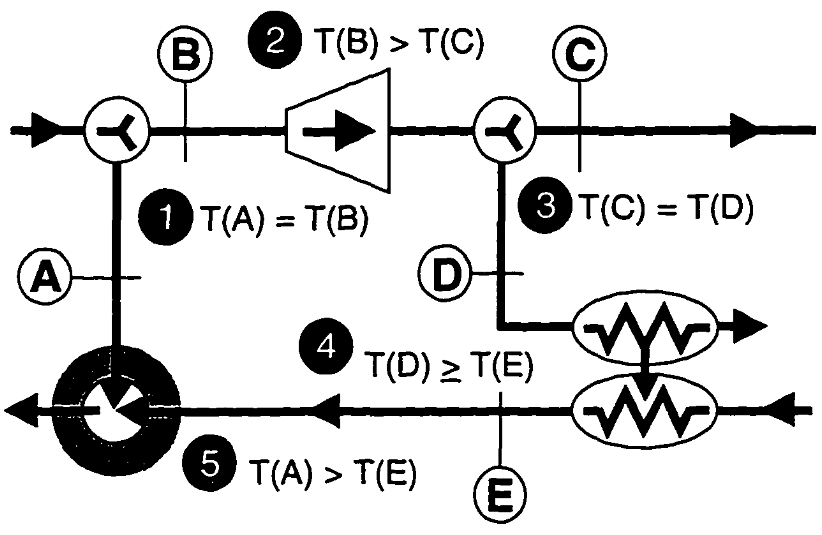


Figure 4.15 Cycle fragment illustrating qualitative transitive inequality reasoning

1. No temperature drop across a splitter gives T(A) = T(B)
2. Temperature drop across a turbine gives T(B) > T(C).
3. No temperature drop across a splitter gives T(C) = T(D).
4. T(D) ≥ T(E) because perfect heat transfer in the heat-exchanger would make T(D) = T(E).
5. By transitivity we have, T(A) = T(B) > T(C) = T(D) ≥ T(E), or T(A) > T(E).

4.3 Step Three: Identifying Plans

Plans, as discussed in Section 3.6, are patterns of function that have been abstracted and reified. Unlike roles, plans are instantiated deterministically, with no probabilistic updating of belief, because the ambiguity attending the presence of a plan tends to be much less than that surrounding a potential role. Essentially, the hard work is done in the recognition of individual roles, and by contrast the recognition of plans is simple and direct. The defPlan form simply expands into a forward-chaining rule that executes when the conjunction of its conditions is satisfied.

A plan instance proposition is instantiated for each conjunction of conditions found. Thus there may be several plan instances active in a given cycle. For example, the plan to regenerate via direct-contact mixing is instantiated four times in Cycle 7 of Appendix B. However, two of these instances are subsumed by a specialization of this plan, *direct-regenerate-with-minimal-delta-T,* which has more specific conditions.

5
IMPLEMENTATION AND EMPIRICAL EVALUATION: CARNOT

TNT has been fully implemented in Common Lisp in a system called CARNOT. The current version of this system currently correctly identifies the function of forty-nine thermodynamic cycles, using a knowledge base contains 107 evidential tests, as shown in Table 5.1. We estimate that this version of CARNOT has achieved 90% coverage of the domain of single-working fluid thermodynamic cycles. Of the forty-seven cycles CARNOT has analyzed, thirty-two are from *Analysis of Engineering Cycles* (Haywood

Table 5.1
Summary of CARNOT tests for and against particular roles

Device	Role	Pro	Con
Cooler	Fluid-cooler	1	1
	Heat-ejector	2	1
	Heat-provider	4	0
	Intercooler	1	0
	Total	8	2
Heater	Fluid-heater	5	0
	Heat-absorber	4	0
	Heat-injector	5	0
	Preheater	4	5
	Reheater	2	2
	Total	20	7
Heat-exchanger	Fluid-cooler	1	0
	Heat-absorber	1	0
	Total	2	0
Mixer	Flow-join	3	1
	Jet-ejector	4	2
	Mxr-heater	6	0
	Mxr-cooler	3	0
	Total	16	3
	Grand Total (107 tests)	*88*	*19*

Device	Role	Pro	Con
Pump	Flash-preventer	2	3
	Flow-producer	2	0
	Total	4	3
Splitter	Bleed-valve	5	1
	Flash-chamber	9	3
	Flow-fork	2	0
	Total	16	4
Subcycle	Bottoming	1	0
	Energy-remover	2	0
	Heat-mover	2	0
	Radiation-isolator	1	0
	Simple-engine	1	0
	Topping	1	0
	Work-generator	1	0
	Total	9	0
Throttle	Pressure-decreaser	3	0
	Saturator	5	0
	Total	8	0
Turbine	Fluid-cooler	3	0
	Work-source	2	0
	Total	5	0

1991), and comprise all single-substance cycles presented in this text, which is considered to be a standard for this field. The other fifteen cycles are drawn from other introductory thermodynamics texts.

In this chapter we first present a complexity analysis of the algorithms as they have been realized in CARNOT. We then present empirical evidence congruent with this analysis, and examine certain other performance characteristics of CARNOT, including the sensitivity of the algorithm to the specific values of the likelihoods in the knowledge base.

5.1 Computational Complexity

Empirically, the complexity of the inference process, as implemented in CARNOT, is $O(n^2)$ in the size of the input cycle, as we will see in the latter half of this chapter. We will now show that this is a reasonable finding. As you may recall, the inference algorithm consists of four parts; (1) structural analysis, (2) initial instantiation of evidence, (3) role inference, and (4) plan instantiation. We analyze each part in turn below, taking the number of devices in the input cycle to be the basic measure of input size. This is a good but not perfect metric, because if we have two cycles of the same size, the one containing more splitters will require more computation due to the additional fluid loops introduced into the topology. We address this issue in our worst-case estimates.

5.1.1 Structural Analysis

The first step of structural analysis, summarizing the structural components and grouping them into sets by type, requires a single linear pass through the devices of the system. The second step, identifying fluid-loops is more computationally intensive due to the potential for flow-forks in the topology. The algorithm (originally presented in Figure 4.6) is repeated here in Figure 5.1.

The identification of fluid-loops employs a nested loop. The outer loop iterates over all possible starting points for a fluid-loop. A starting-point is either a heat-exchanger, to ensure we identify all loops connected only by such devices (since starting at a particular arbitrary point and traversing the topology would never lead to those devices on the other side of the heat-exchanger), a splitter, or a mixer. In practice, the set of starting points typically contains three to five devices, and rarely contains more than ten; in the worst case, this outer loop would iterate over all devices. The inner loop traverses the connectivity graph of the cycle from the starting-point until it returns to the starting point. In a cycle with no splitters, this would require exactly N comparisons, where N is the size of the input cycle. In the no-splitter case, then, the outer loop would execute once for each heat-exchanger, or if there are no heat-exchangers, execute once from an arbitrary starting point. The simplest case, in which there are no heat-exchangers, is linear in N, since the outer loop executes once. In the worst no-splitter case (ignoring thermodynamic constraints, such as the necessity for other device types), all cycle devices would be heat-exchangers, and the outer loop would execute N times, resulting in performance quadratic in N.

In the worst splitter case, the cycle would be composed of the most possible number of splitters. Leaving aside thermodynamic constraints (e.g., the requirement for

```
Identify-fluid-loops
  For each start-point SP in Fetch-start-points
    Until Fluid-loop-identification-complete
      For each fluid-loop FL in Trace-fluid-loops(SP)
        Assert-fluid-loop!(FL)

Fetch-start-points
    For each device D
      When Splitter?(D) or Heat-exchanger?(D)
        Collect D

Trace-fluid-loops(start)
    Find-fluid-loops(start,{ },{ },{ })

Find-fluid-loops(start,route,stuffs,route-start)
    For each device in Fetch-outlet-devices(start)
      Let common-stuff = Find-common-stuff(start,device)
        If Source-to-sink-route?(route-start,device) then
          Let stuffs = stuffs & common-stuff
          Make-fluid-loop-proposition(stuffs,device,route)
        Elseif Outlet-connected-to-inlet?(device,route-start) then
          Let stuffs = stuffs & common-stuff & Find-common-stuff(device,route-start)
          Make-fluid-loop-proposition(stuffs,device,route)
        Elseif Member(device,route) then
          do nothing
        Else
          If Contains-devices?(route) then
            Let route = device & route
          Else
            Let route = device & start
          Let stuffs = common-stuff & stuffs
          If route-start = { } then
            Let route-start = start
          Find-fluid-loops(device,route,stuffs,route-start)
```

Figure 5.1 Algorithm for finding fluid-loops

```
Find-influence-ranges
   For each device D in all devices of the cycle
      Find-influence-range(D,D,{ })

Find-influence-range(device,start,result)
   For each outlet-device OD immediately downstream of device
      If Member(OD,result) then
         Return result
      Elseif Range-terminator?(OD,start)
         Return Adjoin(OD,result)
      Else
         Find-influence-range(D,start,Adjoin(OD,result))
```

Figure 5.2 Algorithm for finding ranges of influence

at least one heater and one cooler), the most splitters a cycle could possibly contain is N/2, since we must rejoin these paths into a single return flow to the inlet of the original splitter. In this situation, the outer loop would execute once for each splitter, or N/2 times. The inner loop, since it recurs in order to trace all possible fluid paths, would execute twice for each splitter (i.e., once for each branch) and once for each mixer encountered, or N + N/2 times. During each of these executions the algorithm would make a maximum of N-1 comparisons to determine if it had completed a fluid-loop. In this circumstance, we have $O(N^3)$ performance bounded by $0.5N \times 1.5N \times (N-1)$.

The identification of ranges of influence, shown again in Figure 5.2, is worst-case linear in N for each device, resulting in $O(N^2)$ complexity. The worst case would require a comparison of the starting device to each other device in cycle; in practice ranges of influence contain between one-quarter and one-third of the devices in the cycle.

5.1.2 Role Inference

The role inference algorithm (shown in Figure 5.3) relies on both local and global properties. In CARNOT, a forward-chaining rule engine coupled to a logic-based truth maintenance system (LTMS) performs the local propagation. Although this is merely one possible means of implementing TNT, we believe it is a reasonably efficient one. The complexity characteristics of the LTMS are well-known—its core Boolean constraint propagation algorithm is linear in the number of facts in the database—so we take this as a given lower bound on performance and focus our analysis on the characteristics of the TNT algorithm. Forward-chaining rule engines, of course, are perfectly capable of exponential performance, given a pathological set of rules, so we will pay particular attention to the characteristics of the rule engine. In this particular rule-engine, all rules that match are executed, and those that match are guaranteed to execute exactly once.

The first operation, do-local-propagation, instantiates propositions arising from the structural analysis of the cycle. There are three types of propositions; a set of constant size that encodes information about the role hierarchy, a set linearly proportional to the number of tests in the database, and a set linearly proportional to the number of devices in the input cycle. As an example of the first type of proposition,

```
(ROLE-SPEC HEATER (REHEATER HEAT-INJECTOR))
```

```
Update-role-beliefs
   Do-local-propagation
   Update-evidence-and-roles
Update-role-beliefs
   Do-local-propagation
   Update-evidence-and-roles

Update-evidence-and-roles
   For each entity E in cycle entities
      Update-range-of-influence for E
      Update-evidence-set of E
      For each piece of evidence Ev in updated evidence set for E
         Update-belief(Ev)
   Update-role-truth-labels for all entities

Update-role-truth-labels
   For each entity E in entities
      Unless Most-likely-roles(E) = True-roles(E)
         For each role R in True-roles(E)
            Set-truth-label(R) = unknown
         For each role R in Most-likely-roles(E)
            Set-truth-label(R) = true
   Do-local-propagation
   When New-facts-instantiated? or Change-in-evidence-sets?
      Update-evidence-and-roles

Change-in-evidence-sets?
   For each device D
      For each role R of device D
         Unless Fetch-evidence-propositions(R) = Propositions-of(Evidence-set-of(R))
            Return true
```

Figure 5.3 Belief-revision algorithm

represents the fact that, for a device of type `heater`, `reheater` is a specialization of the `heat-injector` role. The number of these facts depends on the size of the role hierarchy, which rarely changes, and would only do so in the event of a major extension to the system's capabilities, such as the addition of a new device type. As an example of the second type of proposition, there are control facts of the form

```
(ACTIVE-TEST SBC-TST1 WORK-GENERATOR 2.5)
```

that encode information used for instrumenting the performance of the system. By changing the truth labels of these facts, we can selectively disable tests. Since the size of the knowledge base will change only rarely and incrementally once it has achieved sufficient coverage of the domain (as we have in our CARNOT implementation), this set may be considered constant in size. As an example of the third type of proposition, there are several structural facts asserted about each device:

```
(THR1 HAS-OUTDEV SPL3)
(THROTTLE THR1 S8 S9)
(S8 HAS-SINK THR1)
```

In the current version of CARNOT, there are between nine and twelve facts asserted for each device, with the actual number depending on the type of device.

During this instantiation operation, the potential for nonlinear behavior would arise in any proposition that related devices. For example, a relation that holds between all pairs of devices would require an $O(N^2)$ operation to instantiate $[N \times (N - 1)]/2$ facts.

The only proposition with this potential is the *adjacent* relation, which occurs, at a minimum, N times, once for each device in the system. This minimum will be exceeded insofar as there are splitters or mixers in the input system, because, as we saw in Section 3.5, the adjacency relation spans splitters and mixers that are acting as flow-splits and flow-joins respectively. The pathological case of a cycle comprised solely of mixers and splitters will result in $[N \times (N - 1)]/2$ adjacency relations; in practice the actual number will lie between these two bounds. As adjacency propositions typically comprise 5% of the total propositions, this behavior has a minimal impact on the overall linear behavior of the initial fact instantiation.

The next step is update-evidence-and-roles, which consists of an iteration over all cycle entities (i.e., devices and subcycles). The first operation within this iteration is an updating of influence ranges. Influence ranges depend on the roles of all devices within the range, because a change in one or more of these roles may affect the influence range. For example, suppose a cooler has a mixer in its range of influence; if the role of this mixer changes from a flow-join to an open-heater, the influence range must be terminated at this point. The complete updating operation is worst-case $O(N^2)$, because it iterates over all cycle devices, and the actual update is itself a linear operation.

The next operation within the iterative loop, update-evidence-set, executes once for each device in the cycle. In each execution, it compares the contents of a set proposition containing a list of all evidence propositions that were labeled true at the last update to the current state of the database, in which there may be new evidential propositions or old propositions that have lost support. This comparison is linear in the size of the evidence set, which is typically contains one to five members; the maximum is determined by the number of tests for a particular role, and is currently seven, for the open-heat-exchanger role. Taking the number of evidential propositions per device to be roughly constant (since each test will at most execute once), the complete set updating operation is therefore $O(N)$.

The next operation is the actual Bayesian updating step, which is applied to each piece of evidence. This update is performed via a message-passing algorithm that operates over the role hierarchy. The actual update is a simple mathematical calculation that executes in constant time (see Appendix A for the specific formula). The number of updates that occurs is equal to the size of the role hierarchy for the role in question. The fewest updates in our system is two, for a throttle, whereas, at a maximum, a heater requires five and a subcycle requires eight. In the worst case, then the Bayesian update step requires eight mathematical calculations; and in the average case (computed from our initial test set of thirty-six cycles) it will execute 3.4 times. Considering this operation to execute in constant time, the overall probability updating operation therefore executes in O(N), since we iterate over all devices.

Update-role-truth-labels complicates matters, because it may cause the entire updating algorithm to re-execute. The first part of this operation, updating truth labels, is a linear operation over the roles of each cycle entity. The next operation, do-local-propagation, is linear as described above; in practice we can expect it to execute in less time now because the structural implications have already been instantiated; in fact, it is possible that no new facts will be instantiated. Notice, however, that we have to test for a disjunction of two conditions; either the instantiation of new facts, or changes in evidence sets. Even if there are no new facts, the change in role truth labels may cause a change in the truth labels of one or more pieces of evidence, requiring that we re-execute the entire updating algorithm.

The new-fact test is implemented as a simple check of a counter within the rule-engine, so it executes in constant time. The change-in-evidence-sets? test, however, must iterate over all roles of each device, so it executes in linear time. Since it terminates as soon as it finds a change its average case is *O*(N/2).

Determining whether or not these tests will provoke another execution of the algorithm is difficult, because it depends largely on the characteristics of the particular input cycle. We can show, via static analysis of a particular knowledge base (as described in Section 4.2.3 above), that the algorithm will in fact terminate for that knowledge base. In general, the number of iterations will be proportional to N, since large cycles have more possible construals than smaller ones. Among cycles of a given size, however, we may see significant variation in the number of iterations required to reach quiescence, due to what we term the *degree of novelty* of the cycle. A cycle in which most devices play the roles predicted by the initial prior probabilities in the knowledge base will have a low degree of novelty, and therefore the algorithm will often reach quiescence within a single iteration. On the other hand, a cycle in which several components are used in novel ways will require more iterations to reach quiescence.

Given that our knowledge base consists of a finite number of tests, and that each device has an average number of tests associated with it, and that our rule engine guarantees to run rules exactly once for each set of antecedents, then for any knowledge base for which the static analysis has demonstrated an absence of loops, we can be sure that the role inference algorithm will achieve quiescence. To estimate the worst case, let us suppose that each device in N has associated with it four pieces of evidence, and that the cycle's topology is such that all four pieces will eventually be instantiated, but that at each iteration, only one piece is in fact instantiated. In this situation, we would require 4N iterations to reach quiescence, so the outer loop would be linear in N. Given the worst case quadratic performance of update-evidence-and-roles, our overall worst case would be $O(N^3)$. In practice, a large fraction of the total evidence is instantiated within the first two iterations, and even in the worst of the cycles we have examined, the actual iterations were more on the order of N/2, or almost an order of magnitude faster than the worst case.

5.1.3 Plan Instantiation

Plan instantiation is a local operation that is linearly proportional to the size of the input cycle. Because plans generally describe intention at the level of the entire cycle, they tend to have several antecedent conditions (typically between six and twelve). On the one hand, each antecedent requires a matching operation, which increases the computational cost of plan instantiation. On the other hand, the large conjunction of conditions is unlikely to match more than once for a given cycle, so we can expect relatively few redundant plan instantiations[1].

5.2 Empirical Performance

CARNOT has been fully implemented in Allegro Common Lisp for Windows on a Pentium-class computer with 32MB of RAM. We have constructed two test sets, one for development and one for validation. The development test set consists thirty-six cycles, and includes all thirty-two single-substance cycles described in (Haywood 1991). We considered the initial development phase complete when CARNOT ran correctly on this first test set.

To determine the generality of the theory, once the initial development phase was completed, we created a second test set of ten cycles, each deliberately chosen to be as different as possible from the first test set. This test set was never used in the knowledge base development, and so CARNOT's performance is indicative of how general our knowledge base is.

In addition to the complexity analysis presented above, we have measured CARNOT's actual CPU time required for each cycle in both test suites and analyzed the resulting data. To address the analytical issues in determining the complexity of the role belief updating algorithm, we have also instrumented this part of the code in order to

[1] The LTMS dependency structure is a graph, rather than a tree, so multiple instantiations of the same consequent will not result in duplicate consequent propositions.

measure its actual behavior. Finally, we have conducted experiments to determine the sensitivity of role inference to the actual likelihood numbers in the knowledge base.

In this section, we will first discuss CARNOT's performance on the validation test set and what this implies for the domain coverage of our current knowledge base. We will then discuss empirical data on algorithmic complexity, and in particular on the behavior of the role-inference algorithm. Finally, we will discuss the effect of noise on the likelihood estimates in the knowledge base.

5.2.1 Performance on Validation Test Set

CARNOT correctly infers the roles and plans for all thirty-six cycles in the primary test set, which we used during the development of the algorithm and knowledge-base. The primary goal in our design of TNT was to create a general account of teleological knowledge that would extend gracefully to unanticipated inputs. To test CARNOT's

Table 5.2
Results of CARNOT run on validation test set

Cycle	Devices	Roles	Correct	Percent Correct
1	5	7	7	100
2	9	10	8	80
3	8	9	9	100
4	9	10	10	100
5	7	9	8	89
6	9	9	8	89
7	9	11	11	100
8	10	13	12	92
9	16	18	18	100
10	13	15	14	93

performance on this dimension, we constructed a validation set of ten cycles that were deliberately chosen to be as dissimilar to cycles within the primary test set as possible. Table 5.2 shows the results of running CARNOT on this test set.

CARNOT correctly identified on average 94.3% of the roles in the ten cycles that comprise the validation test set. This set was not constructed until after the bulk of the knowledge base was complete, and we never ran CARNOT on this set until the validation run, so the knowledge base was not specifically tailored to these cycles. Based on these results, we believe that representation of functional knowledge in terms of tests for evidence is sufficiently general as to allow one to construct a teleological knowledge base from a test set of reasonable size and coverage and expect it to generalize to much of the domain in question.

5.2.2 Scaling Up

We have instrumented CARNOT to report the time required to solve each cycle. The results of running CARNOT on the first test set of thirty-six cycles are depicted in Figure 5.4. The time in seconds on the vertical axis is the result of running on a 133MHz Pentium processor with 32MB of RAM. Although the largest cycle required about four minutes for a solution, notice that cycles with twenty devices (which are still large from a pedagogical perspective) require around thirty seconds. We believe that this performance is well within acceptable limits for coaching applications. A student who has just spent twenty minutes constructing a cycle is likely to wait twenty seconds for the system to

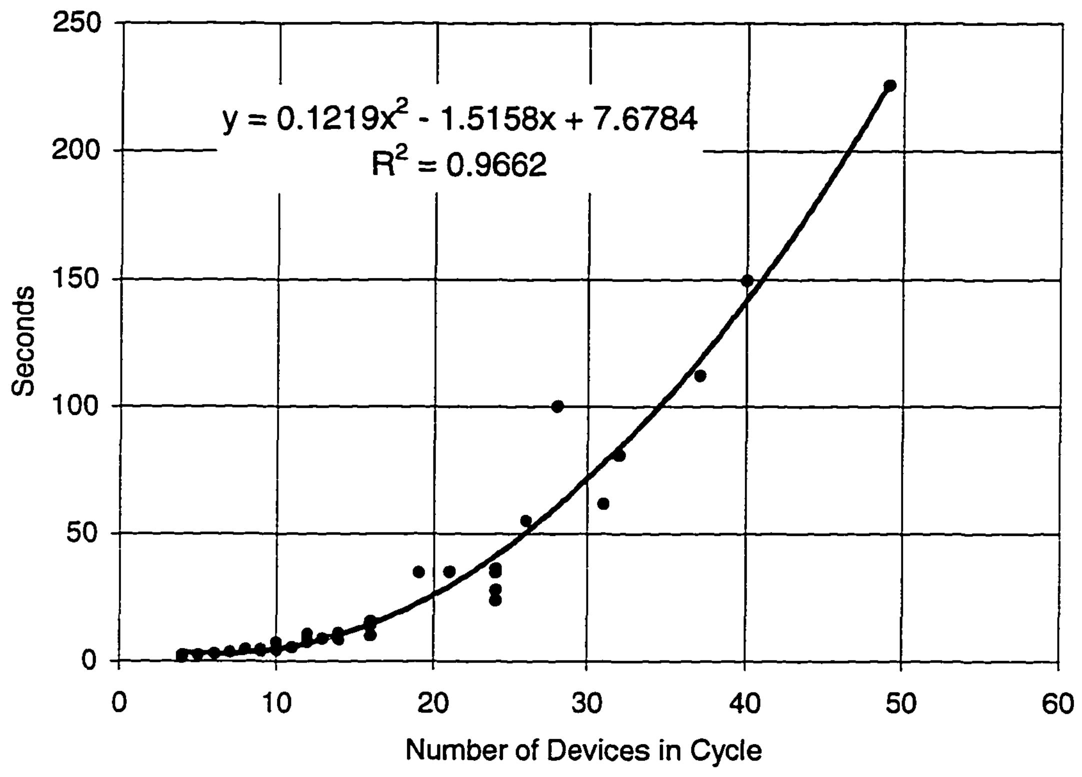


Figure 5.4 Empirical performance for CARNOT on Test Set A

provide coaching advice. Moreover, this wait is only required once after each design change; CARNOT caches all information necessary for the coach, so subsequent queries execute within the time required for looking up the answer.

5.2.3 Characteristics of the Role Inference Algorithm

As we saw in Section 5.1.2 , the role inference algorithm iterates to quiescence. In the extreme case, we can guarantee, via a static analysis of the knowledge base, that it will in fact terminate. However, the tightly interleaved nature of the algorithm makes it quite difficult to develop an average case analysis of its complexity.

The number of iterations required to reach quiescence depends on the order in which the update-evidence-and-roles procedure (shown in Figure 4.13) processes the devices of the input cycle, since the evidence for or against a particular role may depend on belief in other role propositions. Although it is impossible to formulate a general algorithm that will produce the optimal order in which to process the devices of an arbitrary cycle, we have found that a heuristic of sorting the input list of devices to update-evidence-and-roles in increasing order of ambiguity significantly reduces the number of iterations required. We define ambiguity as the number of potential roles a device may play, so a turbine, which has two potential roles (*work-source* and *fluid-cooler*) has less inherent ambiguity in its actual role than a heater, which has five potential roles.

Figure 5.5 shows the number of iterations required to infer the roles of each of the thirty-six cycles in Test Set A. The gray bars indicate the number of iterations required if we sort the input to update-evidence-and-roles in increasing order of ambiguity, whereas the black bars show the additional iterations required if we sort the input in decreasing order of ambiguity. The absence of a black bar indicates no difference in the number of iterations.

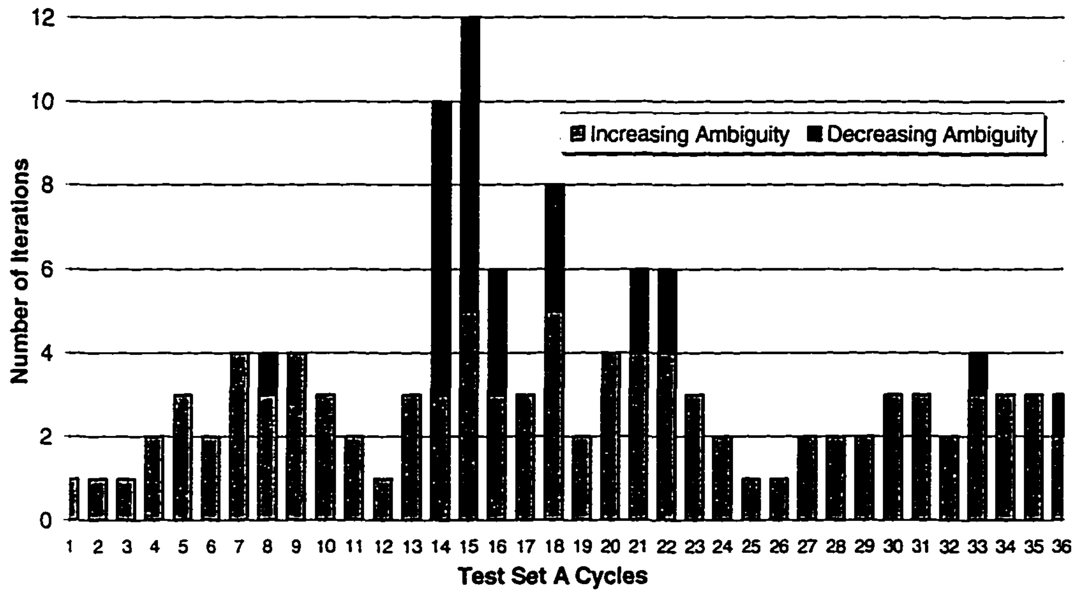


Figure 5.5 Effect of device order on number of iterations to reach quiescence

Although the majority of the cycles in Test Set A are unaffected by the order of input, in some cases the input order results in significant differences. For example, Cycles 14 and 15 are particularly sensitive to the input order. Extensive empirical analysis has yielded few generalizations. The number of pieces of evidence, not surprisingly, is highly correlated with N, the number of devices in the input cycle. However, the evidence count is not well correlated with the number of iterations. For example, Cycle 16 contains 99 pieces of evidence, but only requires three iterations, whereas Cycle 15 contains only 71 pieces of evidence but requires five iterations. The number of iterations required is not well correlated with cycle size. For example, Cycle

21, the largest cycle, requires four iterations, but Cycle 15, comprised of only slightly more than half the devices of Cycle 21, require at least five iterations.

Cycle type is not a strong predictor of iterations either. For example, Cycles 14 and 15 are both nuclear cycles, but so are Cycles 16 and 17, which each require only three iterations. Based on Test Set A, in the average case, we can expect a cycle of moderate complexity to require between two and four iterations.

5.2.4 Sensitivity to Likelihood Estimates

CARNOT uses both estimates of the prior probability for each role (that, is the probability of that role occurring in a randomly chosen cycle) and subjective likelihood assessments.

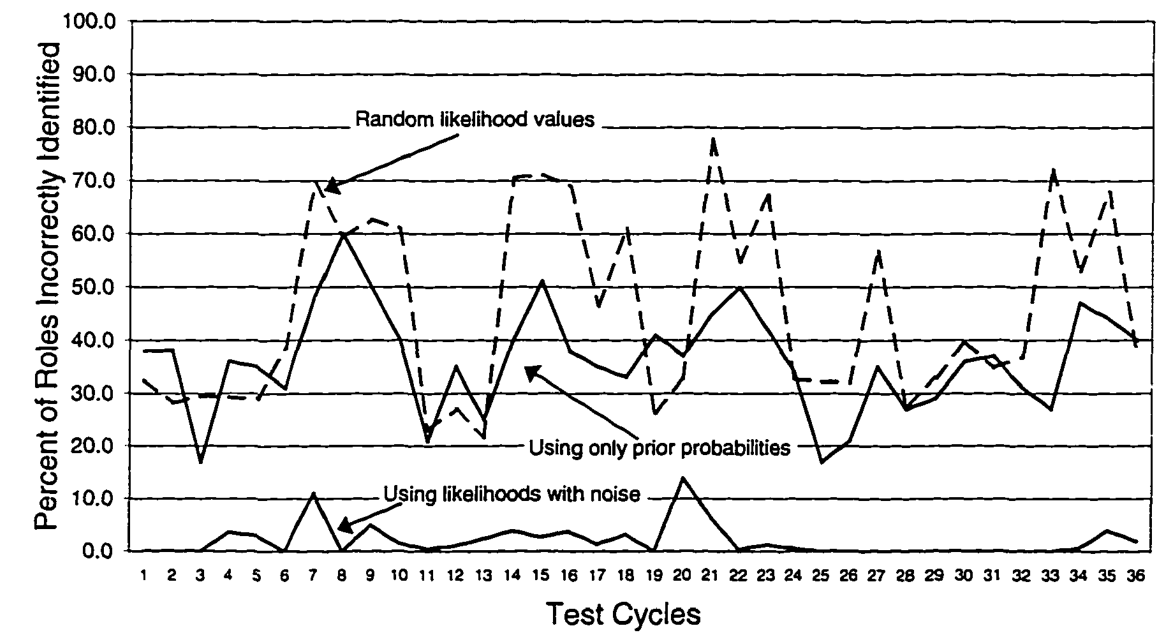


Figure 5.6 Effect of introducing noise into likelihood estimates

We considered using a qualitative scale here, but decided that the computational properties of numbers would afford the simplest and most efficient belief-updating algorithm. Nonetheless, the origin of the numbers is completely subjective and based on the knowledge of the domain that we acquired in the course of this research. The key question here is to what degree the particular numbers in CARNOT's knowledge base matter to its performance.

Our hypothesis was that the particular numbers do not matter, because the knowledge they encode resides in their relative magnitudes. To test this hypothesis, we introduced random noise into the likelihoods. We conducted thirty runs over the thirty-six cycles of Test Set A for random noise levels of 10%, 25%, 50%, 75%, and 100% and averaged the resulting scores for each noise level. A noise level of 10% means that each likelihood is perturbed either 10% above or below its actual value, the direction of perturbation being random. In addition, we also conducted a set of thirty runs in which the likelihoods were assigned completely random numbers ranging from 0.001 to 1000 and a run in which all tests were disabled, to establish the baseline for the a priori probability estimates.

The belief updating algorithm proved to be surprisingly robust to noise, exhibiting only minor degradation at 100% noise, as shown in the lowest line of Figure 5.6. Disabling all tests results in substantial performance degradation, as the system is simply guessing based on background knowledge of the domain. In this case, the system correctly infers about half the roles in the cycles, as shown in the middle line of Figure 5.6. The random run demonstrates that the probabilistic updating scheme does in fact carry its weight, as the results for this run are generally worse than if the tests were disabled, as shown by the topmost line in Figure 5.6.

6
CARNOT-BASED COACHING APPLICATIONS

We have characterized TNT as a performance theory of reasoning from structure to function. As such it is a means to an end, which in this case is the development of effective pedagogical software to help students develop deep conceptual models of physical domains, such as thermodynamics. Although much work toward this end remains to be done, we have implemented and fielded an experimental system called CyclePad that is now in active use at the U.S. Naval Academy, Northwestern University, and Oxford University. CyclePad is an *articulate virtual laboratory* for learning thermodynamics through the design of thermodynamic cycles. It is publicly available on the World Wide Web[1].

An articulate virtual laboratory (AVL), as described in Section 2.2, is *virtual* because it enables the user to design and construction of physical systems, such as power plants without resorting to the use of potentially hazardous, expensive, and inconvenient physical components. Eliminating the time required to configure such systems, even if it were practical to have students work with devices such as turbines, compressors, and boilers, results in more time available for thinking about conceptual issues. An AVL is *articulate* because the user can determine, via a hypertext system, the derivation of any conclusion the system generates.

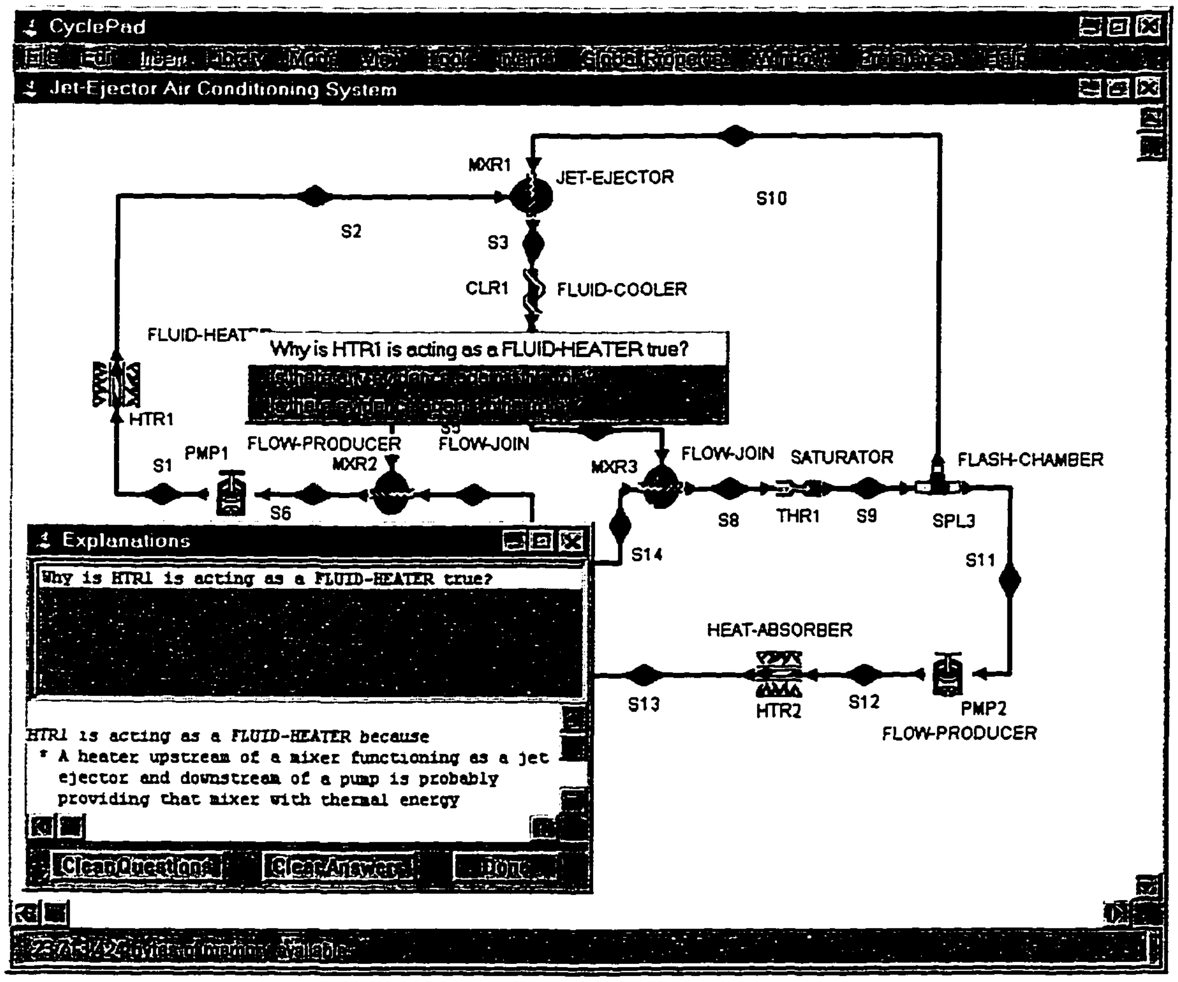
[1] http://www.qrg.ils.nwu.edu/software.htm.

In CyclePad, the user first designs a cycle (or chooses a design from an included library) and then makes modeling and parametric value assumptions. A modeling assumption might be, for example, to assume that a heater operates at constant pressure, whereas the most common parametric values assumed are those for pressure and temperature at various points in the system. Because CyclePad propagates values based on these assumptions, the user need only make a few assumptions to completely solve a cycle in steady-state. The user is typically interested in the efficiency of the cycle, which is a function both of the cycle's design and of the assumed parametric values.

As of this writing, the public version of CyclePad incorporates TNT in two subsystems, a Role Annotator and an Analytical Coach. We have also constructed a research prototype of a Design Coach, which we intend to incorporate into CyclePad in the near future. We view each of these subsystems, which we describe below, as prototypes for further research into how students can most effectively use such software. The current interfaces to these systems should be viewed as first drafts intended to provide users access to the relevant features. We are just now launching an extensive research effort to investigate issues such as how and when to provide coaching information, what information to provide, and how to structure the interface to best support thermodynamics instructors interested in incorporating CyclePad into their curricula.

6.1 The Role Annotator

The most direct application of TNT is the CARNOT-based Role Annotator, which infers function from structure. The Role Annotator enables students to see the roles that CARNOT has inferred for each device, and to explore the reasoning underlying those inferences. It displays the role inferred for each device as a mouse-sensitive annotation



Why is HTR1 is acting as a FLUID-HEATER true?

HTR1 is acting as a *FLUID-HEATER* because
 * A heater upstream of a mixer functioning as a jet ejector and downstream of a pump is probably providing that mixer with thermal energy

Figure 6.1 Role annotations and hypertext explanations in CyclePad

next to the device; clicking on the annotation opens the hypertext system, initially presenting the user with a choice of three questions, as shown in Figure 6.1.

The user can ask why the system has inferred a particular role, and also if there is mitigating evidence or evidence in favor of other roles. These latter two questions help the user assess the likelihood that the inference is accurate, since CARNOT is a plausible reasoner and therefore is capable of making incorrect inferences.

Although this potential for inaccuracy may at first appear to pose a problem for this application, we believe that encouraging students to question computed results is in general essential for the developing of critical thinking. The hypertext system provides the student with the information necessary to form an opinion of the role inference's accuracy. In Figure 6.1 the user has asked for an explanation of HTR1's role as a fluid-heater, and the system is displaying, in the hypertext Explanations window, its rationale, which is that jet-ejectors (i.e., mixers that pump the working fluid by entraining a relatively low-energy flow in a high-energy jet) require energy sources, and the heater is correctly positioned in the cycle's topology to act as such an energy source.

In this case (as in most) the inference is correct. It rests on the one piece of evidence shown in the Explanations window; a user asking the other questions would find that there is evidence that HTR1 is not a preheater nor a reheater, due to topological constraints, but there is evidence that it might be a heat-absorber, because refrigerators are more likely to employ heaters as heat-absorbers than as heat-injectors. In other situations there may be as many as five pieces of evidence in favor, and/or one or two pieces of mitigating evidence or evidence for other roles.

The Role Annotator, as it currently stands, primarily extends the exploratory dimension of CyclePad, rather than affording a different mode of interaction with the user. Alternative implementations could, however, engage the student in different tasks; for example, a user might be presented with a cycle and asked to describe its function, the system then engaging in a Socratic dialog with the user concerning any misidentifications of function.

6.2 The Analytical Coach

The Analytical Coach helps students fine-tune their parametric assumptions. It does so by first running Carnot on the cycle and then referring to a knowledge base of *norms*, which are defined in terms of the roles of each component, inferred plans for the cycle, and other preconditions relevant preconditions, such as the substance of the working fluid.

The results of the Analytical Coach are not deductively sound, as they are based on norms for particular parameters, and any given design may deliberately violate a particular norm—for example, a particular application may require a small but powerful engine, and the designer might choose to achieve these objectives by specifying more expensive materials for the turbine, such as molybdenum, which can withstand higher inlet temperatures.

```
(defNorm {phase|<parameter>} <device-proposition>
 <comment>
 :units <units-specifier>
 :preconditions {<trigger-pattern>|<trigger-pattern-with-test-form>}*
 :norms {<phase-norm>|<parameter-norm>|<qualitative-relation>}+
```

Figure 6.2 Syntax of norm knowledge representation

The syntax for the representation of norms is shown in Figure 6.2. The *device-proposition* is a pattern specifying the device that the norm applies to, such as (turbine ?tur ?in ?out). The *comment* is a natural language explanation of the norm that is incorporated into the explanation presented to the user of the system when the norm is violated. The *units-specifier* denotes the units of any numerical values mentioned in the norm. The :preconditions of the norm comprise a conjunct of facts that must be true if the norm is to apply to the cycle at hand. Preconditions may be either patterns, such as (cycle-type :heat-engine ?reason), or they may include a test form, as in:

```
((substance-of ?in ?subst)
    :test (member ?subst '(refrigerant-12 refrigerant-22))
```

which in this case is satisfied if the working fluid is either refrigerant-12 or refrigerant-22. This test clause may also be used to verify facts about the cycle as a whole, which greatly increases the expressiveness of the defNorm form. For example, one can specify that a precondition for a norm be the absence of pumps in the cycle:

```
(((devs :CYCLE) members ?cycle-devs
    :test (notany #'pump? ?cycle-devs))
```

Norms may either be phase norms, parametric norms, or qualitative relations. A phase norm may be a simple specification of a phase (e.g., liquid, saturated, or vapor), or, for saturated substances, it may include a specification of the quality of the substance, which enables one to specify saturated liquid or saturated vapor. A saturated liquid is a liquid right at the point of changing phase into a gas, whereas a saturated vapor is a gas right at the point of condensing into liquid.

A parametric norm enables one to specify a minimum and/or a maximum numerical value for a given parameter. For example, a norm for the inlet temperature of a turbine is that it not exceed 1100 degrees Kelvin, at which point most metals (with the notable exception of molybdenum) will fail. A minimum temperature has little meaning in this case, so one need not specify one. However, in the case of a Freon-based refrigerator, the heat-absorbing element of the device has both minimum and maximum norms, since the most common application for such devices is the preservation of food; a temperature below the freezing point of water at ambient conditions would be problematic, causing liquids such as milk to solidify, whereas a temperature more than about 7°C would fail to preserve the food.

Finally, norms may be expressed as qualitative relations constraining the values that specified state points may take. For example, a mixer identified as an open heat-exchanger must have a higher temperature at its hot-in inlet than at its cold-in inlet if it is to function as a heat-exchanger; a norm of this type is shown in Figure 6.3. At the same time the difference in these two inlet temperatures is also subject to norms; too small a difference will produce inadequate heat-transfer, whereas too large a difference will result

```
(defNorm T (mixer ?mxr ?in1 ?in2 ?out)
 "The structure of this cycle indicates that this mixer is intended to be used
  as an open heat-exchanger.  Effective heat exchange requires that the hot
  inlet have a higher temperature than the cold inlet, which is not the case
  in this instance"
 :preconditions
 ((role ?mxr mxr-heater ?prob)
  (cycle-type :heat-engine ?reason)
  (effect cooling ?mxr ?inh ?out)
  (effect heating ?mxr ?inc ?out))
 :norms
 ((< (T ?inh) (T ?inc))))
```

Figure 6.3 Example of a qualitative relation norm

in excessive thermodynamic irreversibilities that impair the efficiency of the device. In practice a difference of less than 15 degrees Celsius is inadequate, but more than about 50 degrees Celsius is inefficient; these constraints would be expressed as two norms, of the form:

```
(> 15 (- (T ?inh) (T ?inc))
(< 50 (- (T ?inh) (T ?inc))
```

where (T ?inh) is the temperature of the hot inlet and (T ?inc) is the temperature of the cold inlet.

To illustrate the use of norms, let's consider an example of a simple vapor-power cycle, which completely condenses its working fluid in order to utilize a pump, because pumps, which handle liquids are more efficient than compressors, which handle gases. Completely condensing the working fluid entails some loss in thermodynamic efficiency, because the heat ejected in order to condense the fluid must be re-injected in the boiler.

```
(defNorm phase (cooler ?clr ?in ?out)
 "Vapor cycles that condense their working fluid do so because pumps cannot
  handle a saturated mixture. This condensing process entails a loss in cycle
  efficiency, which should be minimized by avoiding subcooling of the working
  fluid at the outlet to the cooler"
 :preconditions
 ((role ?clr fluid-cooler ?bp)
  (substance-of ?in water)
  (cycle-type :heat-engine ?reason)
  (plan ?instance vapor-power-cycle))
 :norms
 ((:outlet . (?out (saturated 0)))))
```

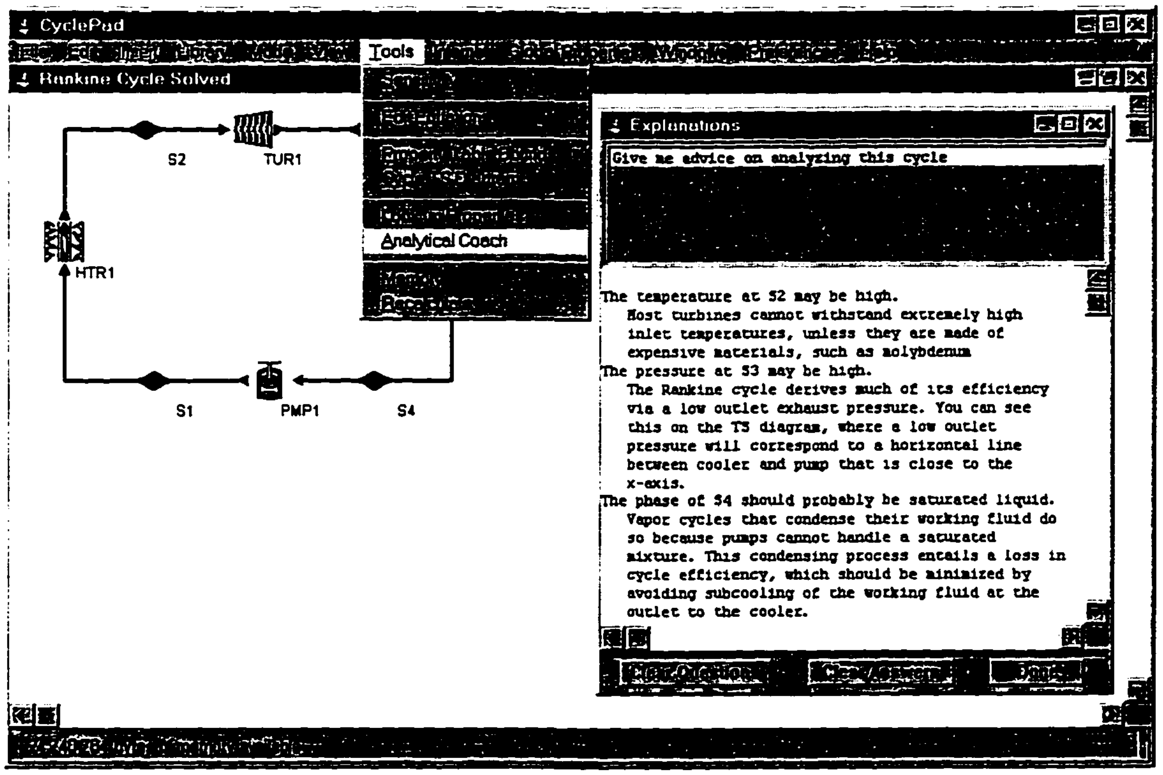
Figure 6.4 An example of an analytical coaching norm

Therefore, in order to minimize this loss, one wants to avoid subcooling the working fluid, that is, ejecting more heat than necessary to condense it.

Figure 6.4 illustrates the norm that one should avoid subcooling the working fluid of a vapor-power cycle. Norms are defined with reference to roles and, as shown in this case, potentially to plans, such as the vapor-power cycle plan.

When a user asks for analytical coaching, CARNOT is first run on the cycle to determine the roles of each device and which plans are in evidence. A set of forward-chaining rules then instantiates those norms whose preconditions are found in the database. A procedure show-norm-violations then gathers up all instantiated norms and checks each against the cycle's parametric data for potential violations. In Figure 6.4 the norm is for the outlet to be saturated with quality equal to zero, that is, a saturated liquid. Should the parametric data indicate that the working fluid at this point is not a saturated liquid (e.g., its quality is greater than zero), then the Analytical Coach reports this violation to the user. The Analytical Coach currently contains seventeen norms; we anticipate that thorough coverage of the domain will require approximately fifty norms. Advice is displayed within the hypertext system, as shown in Figure 6.5.

In the case of the cycle in Figure 6.5, the Analytical Coach has found three potential inefficiencies. The first is that the inlet temperature to the turbine is high compared to the norm for vapor-power cycles. Although this may be a deliberate choice



Give me advice on analyzing this cycle

The temperature at S2 may be high.
Most turbines cannot withstand extremely high inlet temperatures, unless they are made of expensive materials, such as molybdenum
The pressure at S3 may be high.
The Rankine cycle derives much of its efficiency via a low outlet exhaust pressure. You can see this on the TS diagram, where a low outlet pressure will correspond to a horizontal line between cooler and pump that is close to the x-axis.
The phase of S4 should probably be saturated liquid.
Vapor cycles that condense their working fluid do so because pumps cannot handle a saturated mixture. This condensing process entails a loss in cycle efficiency, which should be minimized by avoiding subcooling of the working fluid at the outlet to the cooler.

Figure 6.5 Analytical coaching output in CyclePad

on the part of the student, as discussed above, we believe that bringing this to the student's attention helps to ground the parametric values of the cycle manipulates in reality; in this case, the turbine may deform or melt during operation. This norm violation therefore relates directly to the third rational designer goal, that of preserving the system.

The second norm violation notes that the pressure at S3 (which is the outlet of the turbine, and is hidden in the figure behind the pull-down menu) may be too high, potentially resulting in inefficiencies. In this case the coaching output makes reference to the TS diagram, a widely used conceptual tool in thermodynamics which plots temperature versus entropy (s is the symbol for entropy). Developing sound intuitions about this diagram and how it relates to system performance is an essential part of understanding thermodynamics. (CyclePad can display the TS diagram for a particular cycle).

The third norm violation shown occurs at S4, the inlet to the pump, and, like the second one, points out a potential inefficiency. In this case the liquid is being cooled more than necessary. This particular coaching advice is based on a practical limitation, that pumps cannot handle mixtures of gas and liquids. A saturated liquid is a liquid on the verge of boiling, which is more desirable because it is the hottest liquid that we can run through the pump at a given inlet pressure.

6.3 The Design Coach

Students working with CyclePad first determine the structure of the cycle they are designing, deciding the number, type, and topological configuration of its components. Once this structure is established, the student then makes numerical and modeling assumptions to analyze the performance of the cycle. The Analytical Coach described above is designed to provide help to students in this latter task, but we would also like to provide coaching support for the design task.

The Design Coach currently exists as a research prototype and has not yet been integrated into the publicly-available version of CyclePad. This prototype contains twelve design cases and has successfully retrieved the most appropriate case in a preliminary set of experiments. In this section we describe the architecture of the prototype, while in Chapter 8 we will discuss directions for future research in this area.

The Design Coach is a case-based reasoning system that will ultimately guide students in exploring the vast space of possible cycle designs. Cases consist of a cycle design and a transformation to that design, as shown in Figure 6.6. The representations of the cycle include an automatically-generated TNT functional specification; without knowledge of device function, we have found that case retrieval becomes less accurate.

We use the MAC/FAC model of analogical case retrieval (Forbus, Gentner, and Law 1995), augmented with recent work on evaluating candidate inferences (Forbus et al. 1997). This model posits a two-stage retrieval process consisting of an efficient but overly inclusive initial stage (the MAC, or "many are called" stage) and a second

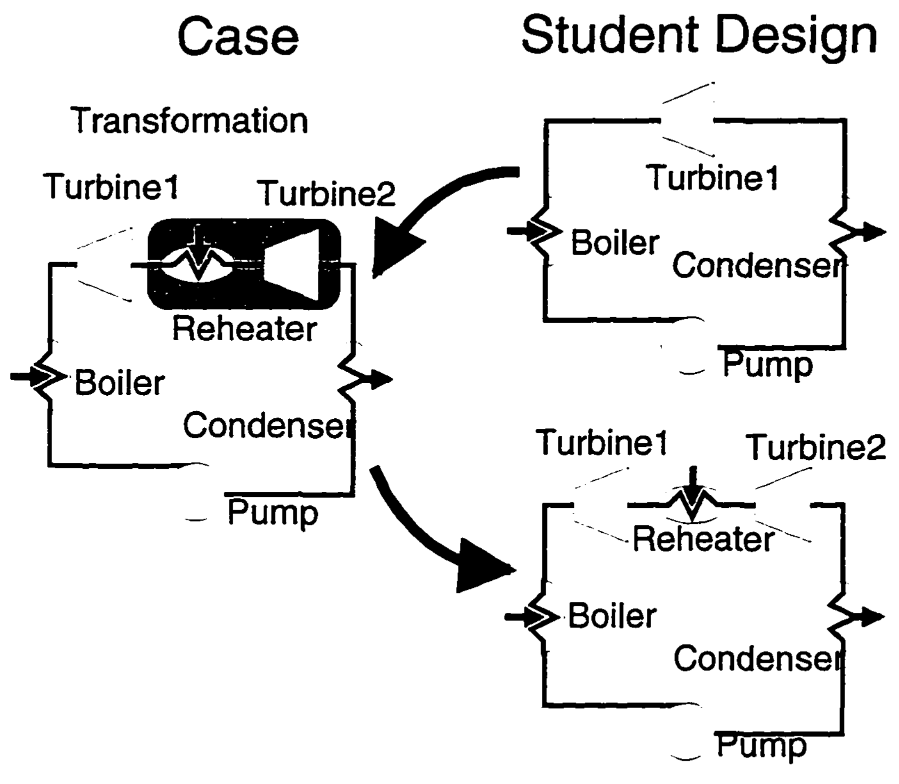


Figure 6.6 Diagrammatic example of a Design Coach case

structural evaluation stage (the FAC, or “few are chosen” stage). The FAC stage employs the Structure Mapping Engine (SME) (Falkenhainer, Forbus, and Gentner 1989), which is a matcher based on Gentner’s Structure Mapping theory of analogy (Gentner 1983).

SME proposes analogical inferences, but this in itself is not sufficient; we also need a means for evaluating the plausibility of the inference, if we are to choose the best case to suit the situation at hand. We use a method for structural evaluation of analogical inferences, which estimates how promising an inferences is, based on its form and the mapping that generated it. For candidate inferences we postulate two distinct dimensions: (1) *support*: how much structural support does an analogical inference derive from the mapping that generated it? (2) *extrapolation:* how far does an analogical inference go beyond the support lent by the mapping? We believe these two measures have significantly different functional roles. Support is like the structural evaluation of mappings, in that more is always better. Extrapolation is more complex; high extrapolation seems desirable in tasks like brainstorming or theory generation, but low extrapolation may be preferable for within-domain comparisons involving highly familiar situations. In the Design Coach, we therefore bias the score produced by this method to favor less extrapolation.

To illustrate a potential use of the Design Coach, let us take as a starting point a simple vapor power cycle, as illustrated in the upper right corner of Figure 6.6. Let's suppose further that a student has analyzed this cycle but cannot achieve the requisite efficiency to satisfy the requirements for a design problem. At this point the student should consider (or be prompted to consider) changes to the structure of the cycle. One possible transformation to the cycle, as shown in Figure 6.6, is the addition of a reheater to the turbine. The net effect of this change is to increase the average temperature of heat injection, which results in an increase in the thermal efficiency of the cycle. The transformation presented to the student is explained via the CARNOT explanation facility, possibly augmented with information stored in the case.

The current prototype of the Design Coach contains a selection of twelve cases, which are created in the following manner. A domain expert uses CyclePad to construct a cycle that illustrates a particular problem. In the course of building this cycle, the expert develops a rich context, which includes the type of working substance, the modeling assumptions for each component, and the assumed parametric values. In "watch me" mode, the expert then modifies the design in a way that fixes the problem. CARNOT is run over each cycle and its output is included in the case, to facilitate analogical retrieval and to enable automated explanations of function for the student. Thus the structural description of the cycle, CARNOT's teleological analysis of what the cycle does and how each part of the cycle contributes to this function, and a formal representation of the expert's transformation are all automatically generated for the case. The only hand-input part of the representation is the expert's specification of the exact nature of the problem, (e.g., low thermal efficiency or high operating cost), which is stated in a tightly constrained formal representation language (a "wizard" style user interface facilitates this step). We based the selection of the twelve initial cases on the likely needs of intermediate thermodynamics students. The average case contains 77 expressions and 19 entities.

In our experiments so far, we have found that when multiple cases were retrieved, choosing the analogical inference with the highest support score always provides the optimal advice. This result should be viewed with caution, since the number of problems tried has been small, the case base is only about one-fourth of what we believe is needed for broad coverage, and, most importantly, it has not been field-tested with students.

Although the computational infrastructure for retrieving appropriate cases has been implemented in prototype, there is much work to be done in order to understand how best to integrate the Design Coach into CyclePad. We discuss directions for this work in Chapter 8.

7 DISCUSSION AND RELATED WORK

In this chapter we first discuss the plausible inference mechanism that supports the reasoning algorithms, and then we consider how the teleological research relates to other work in qualitative reasoning.

7.1 Plausible Inference

The plausible inference mechanism in CARNOT replaces a more conventional qualitative reasoning approach: dependency-directed search (Stallman and Sussman 1977) for a set of intermediate behavioral representations we called *views* (Everett 1995). We found dependency-directed search to be extremely inefficient, because it operates by ruling out interpretations. The final result of such a search was a set of equally likely construals that were indistinguishable on the basis of purely qualitative criteria. A typical set of construals for a heat-engine with five mixers would contain all the permutations of those mixers being assigned open heat-exchangers or flow-join roles, and there would be no principled means for preferring one construal over another. A person with some knowledge of the domain, however, would have no trouble determining which mixers were flow-joins and which were open heat-exchangers.

Reflecting on the differences in CARNOT'S original algorithm and a person performing the same task, it became evident that domain experts bring to bear knowledge of designer intentionality. Recasting our domain knowledge base in these terms caused us to think in terms of ruling *in* the most likely construal rather than ruling out unlikely interpretations, and this resulted in a far more efficient reasoning algorithm. In addition, the result of the reasoning process is a single, most probable construal.

Plausible reasoning, however, is also subject to exponential behavior. The most obvious example is the joint probability distribution, which might be thought of as the naive approach to plausible reasoning; simply enumerate the probability of every possible state of the world in a table, which as we saw in Section 4.2.1, is an exponential undertaking. Bayesian belief networks (BBNs) (Pearl 1988) have evolved as a means for managing this intractability. They are based on the insight that domains tend to have an underlying causal structure. Carefully crafted representations of this structure via directed acyclic graphs can dramatically reduce the number of probability numbers one must estimate.

Although BBNs afford a powerful means for reasoning with uncertainty, we decided against using them in our implementation of TNT, for several reasons. First, BBNs require careful hand-crafting, which works against our goal of making the knowledge base readily modifiable. Any domain may be represented by many networks of differing topologies, but computational efficiency depends critically on crafting the *right* topology; in the limit, a poorly built topology will require as many probability assessments as the joint distribution.

A guiding heuristic for network construction has one identify independent random variable nodes and draw causal links from them to dependent random variable nodes.

Russell and Norvig (1995) elucidate this heuristic, and provide a telling example (drawn in turn from Pearl 1988) of the cost of violating it; suppose that Fred, who lives near the San Andreas fault, has a security alarm installed in his house, and two neighbors, John and Mary, who tend to be home during the day, and will call him at work if they think they hear the alarm sounding. Sometimes earthquakes cause the alarm to go off.

If we are to construct a BBN to figure out the probability that a burglary has occurred given that John has called, then we should start with independent variables, in this case burglary and earthquake. As an aside, even in this trivial example, we find that a well-constructed network requires ten probabilities, whereas the joint distribution requires 31 distinct probabilities.

Should we not construct the network properly, we will have to specify additional probabilities, some of which will be quite hard to estimate. For example, if we started the network with the MaryCalls node, we would have to specify three extra probabilities, and we would have to assess the probability of Earthquake given Burglary and Alarm.

Secondly, the nodes of BBNs typically represent particular random variables that describes possible partial states of the world. For example, in medical diagnosis, each symptom would be associated with a node representing a discrete random variable, perhaps taking on the values present/absent or absent/mild/severe. The analog of symptoms in our teleological theory are not so succinctly described, as they are topological patterns. As such they are best described as a conjunct of conditions. While it would certainly be possible to bind the value of a random variable to the truth label of a conjunct, we felt this approach would require undue complexity in the implementation.

Finally, our theory allows for roles to act as evidence for or against other roles. This would result in a multiply-connected network topology, which requires substantially more complex methods for inference, such as clustering (Spiegelhalter 1986), cutset conditioning (Horvitz, Suermondt, and Cooper 1989; Jensen, Lauritzen, and Olesen 1990; Pearl 1986), or stochastic simulation (Henrion 1988).

TNT's inference mechanism therefore adds an evidential reasoning algorithm to a forward-chaining rule engine that itself is coupled to a logic-based truth maintenance system, or LTMS (Forbus and de Kleer 1993; McAllester 1978; McAllester 1990). Several other researchers have combined truth maintenance and probabilistic reasoning systems. Falkenhainer has added the Dempster-Shafer belief functions to a justification-based TMS (JTMS) to develop a belief maintenance system (Falkenhainer 1986). For our purposes, the JTMS, which is limited to manipulating Horn clauses, lacks the expressive power of the LTMS. De Kleer and Williams have combined an assumption-based TMS (ATMS) with Bayesian probability theory (de Kleer and Williams 1987) to sequentially diagnose multiple faults in digital circuits via the General Diagnostic Engine (GDE). GDE employs model-based reasoning, in which inferences are based on observed deviations in behavior from a model of the correct behavior of the input circuit. The application of probability theory remains tractable in this case because the models of behavior are fairly simple (e.g., Kirchoff's laws for voltage and current) and accurate. Taking as input only the probabilities of individual component failure, GDE can directly compute the conditional probabilities of interest based on the circuit's model.

The ATMS, which enables rapid switching among many contexts, is ill-suited for our purposes. One can think of the progression of the TNT inference algorithm as the evolution of a particular context, as more information becomes known or believed, but there is no reason to compare one context to another during the inference of function.

Ramoni and Riva (1993) have augmented the LTMS to propagate probabilistic intervals, and they use this logic-based belief maintenance system (LBMS) as a substrate for a class of Bayesian belief networks they call ignorant belief networks, or IBNs (Ramoni and Riva 1994a; Ramoni and Riva 1994b). Their LBMS uses a variant of Nilsson's probabilistic logic (Nilsson 1986), which generalizes propositional truth values to range continuously over the interval [0 1]. The LBMS assigns probabilistic intervals to both clauses and propositions and extends the Boolean constraint propagation algorithm to incrementally and monotonically reduce these intervals as more information is added to the database.

An interval of [0 1] assigned to a proposition P represents complete ignorance about the probability of that proposition, whereas an interval of [0.4 0.6] represents partial ignorance, which might occur if some but not all of the conditional probabilities in the relevant joint distribution are unknown. IBNs, therefore, enable the construction of incremental Bayesian belief nets, which both maintain the ability to explain conclusions, via the traditional TMS mechanism, and will at any point in the processing produce probability intervals derived from all information input up to that point. These intervals encode both the probability of the propositions in the database and the degree of ignorance associated with each proposition. This approach is in contrast to the conventional approach to BBNs, which requires that each node have a complete conditional probability table specified prior to propagation.

Ramoni, Riva, Stefanelli, and Patel have applied IBNs to the problem of forecasting glucose concentrations in diabetic patients (Ramoni et al. 1994). In contrast to the domain of digital circuits, where explicit models of behavior exist, the biological systems that regulate glucose levels are sufficiently complex that they relied on a database containing the records of 70 insulin-dependent diabetic patients for their input data. These records contained information on insulin dose, meals, and blood glucose concentrations. Although they do not present extensive results, their system does show promise, and it does so with a small fraction of the conditionals needed to specify a conventional BBN (2262 vs. 19200).

We considered using IBNs in CARNOT, and in fact Ramoni provided us with the source code. However, we found that the distinction between uncertainty and ignorance (of the actual probability distribution) is unnecessary for our purposes, and so we opted for the relative simplicity of a conventional LTMS, in combination with the evidential reasoning algorithm described above.

Goldman and Charniak (1993) have developed what they call a "network construction language" for incrementally constructing belief networks to facilitate natural language understanding. This language, FRAIL3, enables the user to specify canonical models for the probability distributions associated with various types of random variables. These models are similar to the noisy-OR and noisy-AND logical operators described in (Russell and Norvig 1995, p444); these operators reduce the description of a random variable of k parents to $O(k)$ parameters instead of $O(2^k)$, which the full conditional probability table would require, and they enable the dynamic construction of networks, since they, and the rules for their application, may be pre-specified.

Although there are striking parallels between TNT's incremental inference of function for a given cycle and the Bayesian method for story understanding that Goldman and Charniak describe in (Goldman and Charniak 1993), to the extent that one might construe TNT's process as "reading" a cycle, in the natural language understanding sense of the word, we again opted for the relative simplicity of our approach over FRAIL3. For one thing, as Goldman and Charniak note, in FRAIL3 "the networks built are not used for truth/reason maintenance" (Goldman and Charniak 1993, p198). Since we rely extensively on the LTMS to provide explanations of TNT's inferences, an approach based on modifying FRAIL3 to provide the relevant explanations would be needlessly complex.

Finally, our approach obviates the need for pre-specified probability distributions, since we apply Bayes' rule directly and incrementally to the information available in the database.

Making evidence a fundamental unit of knowledge provides three benefits. First, representing teleological knowledge in terms of evidence turns out to be quite intuitive. Second, units of knowledge can be small, as no one unit carries the full burden of anticipating and reacting to every possible context. This modularity of knowledge facilitates changes to the knowledge base. Finally, there are well-known algorithms (Pearl 1988) for incrementally propagating evidence (see Appendix A).

Representing knowledge in this manner imposes a constraint of conditional independence on each piece of evidence. In other words, when an evidential test fires and instantiates a piece of evidence, that evidence should be independent of all other instantiated evidence, or else its associated likelihood will inflate the probability of the role the evidence indicates.

This is not quite so onerous a constraint as it may at first appear. The key is to arrange that no two tests install separate evidence propositions based on the same evidence. Since tests are specialized on a particular type of device performing a particular role, the knowledge engineer need only ensure that all tests for a particular device type are mutually independent. The CARNOT test database, which suffices for the accurate identification of all thirty-six cycles in the initial set of cycles, consists of 107 tests, of which no single device and role combination accounts for more than five tests.

To illustrate this constraint, suppose that we have a test for a splitter playing the role of a bleed-valve that looks for a turbine at the splitter's inlet and another turbine at the splitter's outlet. A second test for a splitter-as-bleed-valve that also requires a turbine at the splitter's inlet must also specify that the outlet device of the splitter is not a turbine. Without this additional specification, both tests would fire and install separate evidence propositions, in effect double-counting the evidence for this particular role.

Finally, we would like to note the expressiveness that our theory affords, due to the ability to encode suppressive evidence. In a conventional forward-chaining inference system, reasoning about negation is often difficult. As an example, we will consider the intricacies inherent in the inference of the flash-preventer role for a pump. In general, this role applies when a pump is raising the pressure on the working fluid so that it a downstream heat-injection will not cause it to prematurely vaporize, or "flash." This heat-injection may occur via a mixer playing the role of an open heat-exchanger, and we have an evidential test in CARNOT's knowledge base for this case. Premature flashing is only a problem if there is a pump downstream of the heat-injection, so our tests checks for this condition as well.

There is a case in which such a topology is not indicative of flash-prevention, however. Many heat-engines use steam-drums, which one represents as a mixer coupled to a splitter. One splitter outlet delivers dry saturated vapor either to a turbine or a superheater, and the other delivers saturated liquid to a loop that flows through a heater. This loop often contains a recirculating pump acting as a flow-producer. Should there be a pump and a mixer acting as an open heat-exchanger upstream, the recirculating pump will be construed as the downstream pump that must be protected from premature flashing, and an incorrect piece of evidence will be instantiated.

Suppressive evidential tests provide a means to rectify this error; we simply write another test for the same topological pattern, and also for the presence of a steam-drum, which is identified as a multi-port-chamber. Should this test fire, it instantiates a piece of evidence against the flash-preventing role. Without this ability to write evidence to address special cases, it would be difficult if not impossible to write tests that discriminated sufficiently.

7.2 Qualitative Reasoning about Function

De Kleer was the first to investigate the inference of function from structure, which he did in the domain of electronic circuits (de Kleer 1979). In his theory, a causal analysis describes in qualitative terms the behavior of a given circuit topology, and a separate teleological analysis determines the purpose of the system by parsing via a grammar of circuit functions. This work established what has become the conventional approach to reasoning about function, in which a two-stage reasoning process first does a qualitative behavioral simulation of the input then uses the resulting information to infer function.

In contrast to this approach, TNT posits a process of topological parsing that enables the reasoner to proceed directly from structure to a functional construal of the input. We consider this to be one of the main contributions of this work. TNT's multiple representations of locality are similar in spirit to the view of locality in Sussman's *slice* concept (Sussman and Steele 1980). With respect to slices, Sussman and Steele write that "what we have done here is to introduce an *alternative point of view* of the circuit. While in principle it contains no extra information, the new viewpoint is better organized for certain purposes...The constraint language permits us to introduce many such redundant viewpoints so that they can cooperate in solving a problem." (Sussman and Steele 1980 p21, emphasis in original).

Whereas the TNT reasoning process is purely topological, disregarding the geometry of the input system, Davis (1983) argues that a useful definition of locality arises from an explicit representation of physical organization. For diagnostic tasks, he notes that aspects of a system that are physically adjacent (perhaps in order to achieve a compact packaging of the device) may be behaviorally disparate. A reasoner operating solely on a behavioral description may not be capable of diagnosing problems, such as bridge faults in electronic circuits, that arise from interactions among physically adjacent components; "changes small and local in one representation are not necessarily small and local in another" (Davis 1983, p88). Although TNT does not reason about geometry, its multiple representations of locality are consistent with this insight.

De Kleer specifically rejects topological analysis, for two reasons. First, he argues that a component with multiple functions will result in a tangled and complicated hierarchical description that will impede reasoning. Second, he concedes that a performance system would be well-served in the short-run by resorting to topological and geometric reasoning, but at the expense of confounding what might otherwise be learned about the limits of causal reasoning (de Kleer 1984, p207). In particular, he argues that "[e]xpert systems are aimed at producing what performance is possible in the short term without consideration of the longer term. Typically this is achieved by recording as many of the heuristics and rules of thumb that experts actually use in practice, as possible. This is misguided (de Kleer 1984, p208).

With respect to the first objection, TNT's topological reasoning differs significantly from de Kleer's conception thereof; "topological analysis attempts to produce a hierarchy of structures, each of which is a known behavior" (de Kleer 1979, p20). Because TNT reasons directly from structure to function, there is no inference of behavior, and therefore no construction of such a hierarchy. Avoiding this hierarchy precludes the concomitant problems with identifying multiple functions for a given component.

The second objection is in essence a knowledge engineering argument, specifically about what knowledge to represent and how it should be organized. This concern motivates what de Kleer terms the "no function in structure" (NFIS) principle, which in its original form holds that "the laws of the parts of a component may not presume the functioning of the whole." (de Kleer 1984, p239). Although other researchers have taken issue with this principle (e.g., Iwasaki and Simon 1986; Keuneke and Allemang 1989), and de Kleer has acknowledged that some of these criticisms are well-founded, he contends that "some principle like no-function-in-structure is critically needed. How else can we be assured that the predictions of our models do not simply regurgitate assumptions we have built into them?" (de Kleer and Brown 1986, pp56-7).

This point deserves careful consideration if we are to construct robust performance systems that address real-world tasks. Such systems could be constructed from scratch, via the short-term expert system approach that de Kleer rejects as misguided, but only at exorbitant and recurring expense in terms of programmer time. A theory of causal or teleological reasoning, on the other hand, gives us leverage via its generality. Obtaining this leverage was the underlying motivation for both de Kleer's work and TNT. Now that we better understand qualitative models of behavior, from de Kleer's and other work, such as Forbus' Qualitative Process Theory, (Forbus 1984), we can focus on building robust performance-oriented architectures that enable the modular application of different types of reasoning.

De Kleer's work spanned both behavioral and teleological reasoning because at the time little was known, from the perspective of artificial intelligence theory and practice, about either. TNT on the other hand, focuses exclusively on making functional inferences from topological knowledge, in order to determine how much we can infer from the least possible input information. So long as the knowledge-base required for this task remains tractable in terms of maintenance and extension, and so long as the underlying reasoning process remains transparent to the user, such an approach, although quite different from de Kleer's remains consistent with the underlying motivations for his work.

Subsequent research in this area has been conducted in three disparate communities, the Qualitative Reasoning community, the Functional Reasoning community, and the Functional Modeling community. The first two consist of artificial intelligence researchers interested in engineering domains, whereas the latter consists of systems engineers interested in developing computational tools for their work.

Work in Qualitative Reasoning attempts to derive function from causal models of devices. Research in the Functional Reasoning community is based on the work of Chandrasekaran, and has taken a different approach, of directly incorporating functional knowledge about physical systems into causal accounts. The Functional Modeling community is primarily concerned with diagnostic reasoning applied to large industrial processes. Research in this community is based on the multilevel flow modeling formalism developed by Morten Lind (1990). This formalism, which models systems in terms of energy, matter, and information flows, is of interest because it directly incorporates the goals of the system into the representation.

The Functional Reasoning community is concerned with what it means to understand how a device works. This line of inquiry, driven by Chandrasekaran, grew out of a dissatisfaction with the opacity of knowledge in the expert systems developed in the late 1970s. In particular, Chandrasekaran makes a distinction between the "compiled" knowledge of systems such as MYCIN, "whose knowledge base contains the evidentiary relationships between disease hypotheses and symptoms directly, without specifying the causal pathways between them" (Sembugamoorthy and Chandrasekaran 1986, p48) and "deep" models, which make explicit causal interactions. The goal of this research is to specify a language (FR) for the representation of the function of devices, primarily for the purpose of diagnostic reasoning.

This language distinguishes five aspects of functional knowledge: (1) structure, (2) function, (3) behavior, (4) generic knowledge, and (5) assumptions. The conception of structure is quite similar to that used in TNT, although FR is more hierarchic, and supports descriptions at multiple levels of abstraction. Generic knowledge consists of "chunks of deeper causal knowledge that have been compiled from various domains to enable the specification of behavior," and assumptions enable the reasoning agent to choose among behavioral alternatives. We will have more to say about these two aspects later, when we discuss the inferential mechanisms of FR.

The terms *function* and *behavior*, however, are defined somewhat differently; in particular function in FR "specifies WHAT is the response of a device or a component to an external or internal stimulus" and behavior "specifies HOW, given a stimulus, the response is accomplished" (Sembugamoorthy and Chandrasekaran 1986, p50). These definitions arise from and support the hierarchic nature of FR; in this view, a function is something on the order of "make an audible alarm noise," whereas a behavior might be "repeated hit of clapper on bell." The conception of function in FR is therefore neutral with respect to implementation, whereas behavior is not.

In contrast, TNT defines function to be the intended behavior of a device, and in fact construes the task of teleological reasoning to be the disambiguation of function from behaviors, of which there is generally more than one. For example, regardless of context, the physics of a turbine operating in steady state dictate that there will be a pressure drop across the device, a concomitant drop in the temperature of the working fluid, and a flow of shaft-work from the device to some other device outside the thermodynamic system (e.g., a generator). The context in which the turbine exists determines which of these behaviors is intentional, and which are simply side-effects, which may be beneficial, neutral, or detrimental with respect to the designer's goals.

These differences in definition arise from differences in opinion about how to obtain the greatest inferential leverage from the reasoning system. In TNT we assume that one must explicitly build the knowledge base to support reasoning within a particular domain, and therefore require that specific device instantiations be first-class objects which have as attributes all relevant behaviors (i.e., roles). These objects, however, may exist at the most convenient level of abstraction to support the reasoning task; for example, CARNOT defines pumps to be devices that increase the pressure of a liquid working fluid, but makes no commitment to their internal structure; they might be Roots blowers, piston compressors, or axial centrifuges. However, TNT is quite capable of representing these subtypes and their relevant behaviors if this is the level at which reasoning must take place.

The leverage we obtain from TNT arises from its modularity; because TNT enables teleological inferences from exclusively topological information, it can be used in interactive combination with other reasoning engines. For example, qualitative behavioral envisioning is often exponential due to ambiguities in alternative behaviors at state transition points; however, one might be able to significantly reduce or eliminate such ambiguities by using the functional inferences of a TNT-based system to choose particular branches at these transition points. In our coaching applications, we have found that CARNOT interacts readily with the existing CyclePad system, yet it remains a separate part, so maintenance and extension of the teleological knowledge base is independent of CyclePad's knowledge base.

The leverage Sembugamoorthy and Chandrasekaran hope to obtain from their approach is the organization of knowledge:

> What the functional representation really does is organize the agent's understanding of how the device's *functions* result from *behaviors* made possible by the *structure* of the device, and contains explicit pointers to *generic domain knowledge* and *assumptions about behavioral alternatives* used by the agent in this process. Thus this representation is a piece in the

> total understanding structure, and is responsible for elucidating the role of the structure in the functioning of the device. (Sembugamoorthy and Chandrasekaran 1986, p59, emphasis in original).

Organizing frameworks have proven to be particularly valuable tools in artificial intelligence, especially as the emphasis in the field has shifted from smart algorithms operating on small data sets to the representation of large amounts of knowledge about the world. The ultimate value of such frameworks rests on the inferences that they sanction, and it is in this area that we have some reservations about this work.

The literature on FR in general tends not to formally specify the semantics of the representation. In (Sembugamoorthy and Chandrasekaran 1986), the authors present an FR representation of a doorbell buzzer, in which the FUNCTION of the DEVICE buzzer is "TOMAKE buzzing(buzzer)." The authors state that the representation "does not understand *buzzing* or *electrically connected* or any of the terminals that are treated as strings of symbols" (Sembugamoorthy and Chandrasekaran 1986, p59, emphasis in original). The ontological commitments of this theory are therefore hard to discern. For example, the predicates *elect-connected* and *buzzing* appear to denote states, yet what are we to make of a predicate such as *repeated-hit*? It would seem that this represents an iteration of states, so at the least it bundles up implicitly a notion of time.

The clapper of the doorbell in this representation has FUNCTIONS "mechanical, acoustic, magnetic." The semantics of these functions are not made clear; (Sembugamoorthy and Chandrasekaran 1986) lacks the DEVICE representation for the clapper, so we must speculate, but it would seem that *magnetic* and *acoustic* are attributes of the material of the clapper, whereas *mechanical* would refer to some aspect of the clapper's structural attachment, such as a hinge-point, unless it denotes an attribute of structural rigidity. The relation *serially-connected,* which applies to "manual-switch, battery, coil, clapper" as part of the FR structural description of the doorbell, is troubling in what it apparently subsumes; for example, the electrical behavior of a circuit depends on which components are connected in parallel and which in series, yet there is no explicit representation of this.

Representations of behavior are in terms of states and transitions, in which the transitions are represented as annotated arcs. For example, the state *elect-connected* (t1, t2) results in a state voltage-applied via an arc annotated with "AS-PER knowledge1 IN-THE-CONTEXT-OF FUNCTION voltage OF battery, *serially-connected*(battery, coil, clapper, manual-switch)." Knowledge1 is a state transition *from voltage-applied*(t1, t2) to *voltage-applied*(t3, t4) via an arc annotated by AS-PER Kirchoff's Law. This is where the representation grounds out, yet there is no account of how Kirchoff's Law could be applied in making inferences about a particular situation.

In a more recent collaborative effort, Vescovi, Iwasaki, Fikes, and Chandrasekaran (1993) have produced a specification for a language that combines aspects of FR and qualitative behavioral reasoning, which they call Causal Functional Representation Language (CFRL). The semantics of CFRL are clearly specified in (Vescovi et al. 1993); in particular, a function F is defined as a triplet (D_F, C_F, G_F), where

D_F denotes the device of which F is a function, C_F denotes the context in which the device is to function, and G_F denotes the goal to be achieved by the function.

Behaviors in CFRL are represented as *causal process descriptions* (CPDs), which are directed graphs in which each node is a partial state description and each arc represents either a causal or temporal relationship. The semantics of CFRL is defined in terms of matching functional representations to behavioral ones; in particular the matching is between sets of CPDs and a trajectory through a behavioral envisionment of the system being modeled. A trajectory is defined as a possible path through the graph of states produced by a qualitative simulation system, such as the Qualitative Process Engine (QPE) (Forbus 1990) or the Device Modeling Environment (DME) (Iwasaki and Low 1993).

Although the authors indicate that extensive testing, including complexity analysis of the algorithm have yet to be done, CFRL appears to define an expressive means for combining FR with qualitative reasoning and for clarifying the semantics of functional representations. How well a CFRL-bases system would scale remains to be seen, as the qualitative envisionment mechanism at the heart of QPE and DME often exhibits exponential behavior in situations of practical interest.

There is reason to believe that FR-based systems will in fact scale to tackle problems of practical interest. In particular, Pegah, Sticklen and Bond (1993) describe a model of the fuel system for an F/A-18 aircraft that contains 89 component devices, 92 functions, 118 behaviors, and 181 state variables. The authors use a variant of FR that they call Functional Modeling[1] (FM), which differs from FR in that it relies on simulation as a core reasoning strategy. The goal of FM is to determine the consequences of a particular set of boundary conditions. The FM reasoning algorithm generates a *particularized state diagram* (PSD), which is a directed acyclic graph in which each node is a partial state description with respect to a particular variable of the device being modeled, along with a statement of how the variable is changed. The PSD is constructed by traversing this graph and adding new nodes whenever the functional representation of the device can be expanded (recall that FR represents devices hierarchically, so the expansion follows the chain device → function → behavior → subdevice → function → behavior... until the representation grounds out). The reasoner determines the effect of the initial boundary conditions on the device by traversing the PSD and accumulating the changes in the state description variables associated with each node.

Although the scale of this effort is encouraging, the authors do not discuss the complexity characteristics of their algorithm. Given its apparent similarity to the process of envisioning (de Kleer 1975), which is often exponential in situations of practical interest, the issue of scalability remains open. The actual representation of the fuel system required two years of a graduate student's time to construct, and is quite

1 Note the overloading of this term, which has also been used by Modarres and others to describe work based on the multilevel-flow modeling theory of Morten Lind, which we discuss later in this chapter.

specialized to this particular system; for example, there are *TransferFueltoFuelTanks* and *TransferViaTransferLines* representations of function in the model, which would seem to limit the generality of this work. However, as part of this work the researchers constructed an extensive library of functional representations of parts, including more than 90 different types of valves. They note that properly integrating such model fragments into a design is not straightforward, and in their implementation is done manually, although work in the area of compositional modeling might well be brought to bear on this issue (Falkenhainer and Forbus 1991; Nayak and Joskowicz 1996).

Other work in the Functional Reasoning community also bears on this research. The closest in spirit is Thadani's work on device understanding (Thadani 1994). He presents an account of how to produce causal explanations of devices given a structural description and a library of structure-function-FR (SFF) templates which associate the functions and FR description of a device with an abstract structural description. A matching algorithm attempts to fit SFF templates to the input structural description. Where there is no exact match, the system attempts to formulate a new structural description (e.g., by consolidating two parallel electrical loads into a single load) and find a match for that. Thadani claims that careful organization of the system's knowledge produces polynomial performance.

This work presents a clear algorithmic account of how a reasoner could utilize FR representations to make inferences from structure to function and in the process clarifies some aspects of FR that are left implicit elsewhere in the literature. In particular, he makes explicit the semantics of such relations as *electrically-connected* (which sanctions the inferences that all voltages at the connected ports are equal and the sum of the currents at all ports is zero). Like TNT, it relies on a large knowledge base to make its inferences. However, TNT differs from this approach in its organization of the knowledge base around evidential tests, which we believe afford greater maintainability than the template-based approach to knowledge organization.

Although the complexity analysis is encouraging, Thadani presents only two examples, each consisting of four components. How maintainable the knowledge-base will remain as it is scaled up to address larger examples is unclear. The representation of behavior relies on causal process descriptions, but there is no account of how CPDs are constructed; in particular there is no discussion of a mechanism which would ensure that independent knowledge engineers to arrive at the same CPD for a given behavior (e.g., the same choices for parameters to specify partial states). Lack of uniformity at this level would be a serious impediment to the development of large-scale SFF based knowledge bases. These issues notwithstanding, this work provides a lucid account of practical FR-based reasoning.

Bylander's method of reasoning by consolidation (Bylander and Chandrasekaran 1985) bears some resemblance to the aggregation step in TNT, although this work is an attempt to infer behavior from structural input, and is presented as a more computationally-efficient alternative to qualitative simulation. Bylander's consolidations are far more general than the aggregation of devices that TNT posits. The goals of these two processes, however are similar; to gain inferential leverage via abstraction.

Keuneke (1989) extends FR work to the problem of device understanding and explanation by adding the teleological predicates ToMaintain, ToPrevent, and ToControl to the representation. This work focuses on developing a causal explanation of malfunction diagnoses. Keuneke's representation of a chemical processing plant is also one of the first large-scale applications of the functional reasoning theory. Note that the semantics of each of these predicates embodies a lack of change, that is, the preservation of a given state over time. The ability to explicitly represent and reason about such maintenance functions considerably enriches the expressive power of the representation. In TNT we achieve this expressiveness in the definition of particular roles, such as the *flash-preventer* role for a pump, and in the achievement of the *system-preservation* goal by a particular device or subcycle of a system.

Allemang (1990) has demonstrated that the concepts of FR extend to non-physical domains, in his case the domain of computer programs. His system, DUDU, takes as input Pascal-like pseudo-code and either verifies its correctness or produces an explanation of the problems it finds. The explicit representation of function in DUDU enables the program to focus its efforts on buggy parts of the input code rather than constructing a proof of the entire program to verify it. In some respects this domain is more difficult for an automated reasoner than a physical domain, because there is no notion of locality for a computer program, aside from the semantics that programmers impose on a particular language (e.g., object-oriented programming). In a poorly constructed or otherwise buggy program, one part may alter the state of any other part of the program at an arbitrary point in the processing.

Goel's KRITIK2 and IDEAL systems (Bhatta and Goel 1993; Goel 1991; Stroulia and Goel 1992) combine FR with case-based reasoning techniques to support design tasks. They perform the structure to function reasoning task that TNT addresses in reverse, taking as input a functional specification and returning a particular structure, in the form of a design case, along with a model that accounts for how the structure realizes the input function. We have used TNT's representations to construct pedagogical cases for thermodynamics which we have then retrieved using MAC/FAC, a model of analogical retrieval.

In the Qualitative Reasoning community, Franke (1993) has proposed a formal representation language, TeD, for teleological descriptions. TeD specifies desired and undesired behaviors in terms of a specific design modification, using two primitive operators, *Guarantees* and *unGuarantees*. This language is domain independent and enables descriptions to be acquired during the design process. Unlike TNT, it requires an envisionment of all possible behaviors of the system, which it achieves via QSIM (Kuipers 1986), and also a specific design change.

Narayanan has investigated the inference of behavior from the geometric information contained in diagrams (Narayanan, Suwa, and Motoda 1995). This work is complementary to TNT, in that it formalizes another mode of reasoning. It is likely that CARNOT could directly act on behavioral evidence generated such a reasoner from the visual depiction of the input, which CARNOT currently ignores. People tend to adhere to visual conventions in the graphical layout of thermodynamic system designs that could provide valuable evidence for or against particular roles. For example, the primary boiler of a steam engine is most often placed on the left side of the diagram, with the flow of the cycle moving clockwise.

Tomiyama, Umeda and Kiriyama have proposed an architecture for supporting what they term "knowledge intensive engineering" (Tomiyama, Umeda, and Kiriyama 1994). They posit a "function behavior state" (FBS) modeler (Umeda et al. 1990) that takes as inputs the behaviors and state transitions from a qualitative simulation and also a functional knowledge base and produces as output a critique of the system. Function is represented in general as "to do something," for example, "to move book." The representation of function is hierarchical, organized either as a taxonomy or a partonomy. The goal of this research is to make more knowledge about the engineering domain available to engineers during design tasks. Functional knowledge can be used to improve designs; for example, the system might suggest a particular design strategy to achieve functional redundancy. The knowledge intensive engineering architecture is similar in spirit to TNT in that they propose that it be comprised of "plug and play" reasoning agents.

There has also been some related work in the Functional Modeling Community, which consists of systems engineers interested in developing modeling methods, primarily to support diagnostic reasoning tasks. Lind (1982; 1990) presents this approach to the representation of device and system function in his Multilevel Flow Modeling (MFM) theory, which posits that systems have an intended purpose or purposes, and models them in terms of such purposes. The flows of the theory refer to processes of material, energy, and information flow. MFM defines a function as a "useful behavior" (Lind 1990, p19), which is consistent with our definition. MFM also makes explicit the goals of the system, more specifically than does TNT; whereas in TNT we posit three design goals (produce a change in the environment, do so efficiently, and preserve the system), the goals of an MFM model are specific to the system being modeled. For example, the goals of an MFM model of the water circulation system of a central heating system are *maintain water level within safe limits*, *maintain condition for energy transport*, and *keep room temperature within limits* (Lind 1990, p16). In contrast, the functions that MFM defines are more general than the roles defined in TNT. For example, MFM specifies mass and energy flow functions that include storage, balance, transport, barrier, sink, and source (Lind 1990, p44). Despite these differences in representation, the overall structure of MFM is largely consistent with TNT. Lind, however, offers no algorithmic account of reasoning via MFM.

Larsson (1996) describes a quantitative algorithm for reasoning with MFM models, where the various state parameter values are derived from measurements. Where measurements are not available, other measurement values are propagated. A set of rules governs this propagation with the goal of producing the most probable set of values. The MFM model is first divided into consistent subgroups. For example, if the absolute difference in the measured flows F_1 and F_2 of two connected flow-carriers (a class of entities that includes sources, transports, non-forking balances, and sinks) is less than a specified error threshold, then those two flows are deemed to be within the same subgroup. The propagation rules give precedence to a flow value from a subgroup of more than one member (such a flow value is said to be *validated*). Values propagated downstream are given priority over values propagated upstream, in order to make flow propagation behave consistently.

Where conflicts in measured values arise, inconsistent subgroups are marked in color; in particular, subgroups of a single member are highlighted in red. Conflict resolution, therefore, is delegated to human operators.

Larsson also presents an algorithm for fault diagnosis that traverses the MFM graph; when it comes to single flow functions it uses pre-specified questions, such as "Is the power switch on?" or sensor readings to determine the state of the function. The algorithm can also generate a limited natural language explanation of its diagnosis as it proceeds depth-first through the MFM graph. Larsson claims that the complexity of these algorithms is worst-case linear because they only traverse static links.

Such MFM models appear to scale; Larsson reports that a large-scale MFM model for monitoring and diagnosis of post-operative intensive care patients called Guardian (Hayes-Roth et al. 1992), which contains 544 rules exhibits a worst-case execution time for a fault diagnosis of 1100 µseconds, and executes 500,000 rules per second. However, the MFM approach differs significantly from ours in that the modeler must make explicit the goals and functions of the system, whereas in TNT the inference of goals and functions based on structure is the output of the process.

8
CONCLUSION AND FUTURE WORK

We have presented Topological iNference of Teleology (TNT), a theory of reasoning from structure to function in the domain of thermodynamic cycles. This theory describes a knowledge representation that enables efficient evidential reasoning from structure to function. The inference step relies on a Bayesian updating mechanism which rules in the most likely role for each device; from roles we infer plans that describe the intention of the cycle as a whole. The contributions of this work include:

- A representation language for structural locality, which enables the inference of function directly from structure, without an intermediate behavioral simulation of the system.
- An account of evidential reasoning from structure to function that operates in quadratic time.
- A knowledge representation based on evidential test for or against particular roles that enables our approach to scale, both algorithmically and from the point of view of knowledge base management.
- Proofs of concept for self-explanatory coaching systems for use in design-based exploratory learning environments.

Plausible reasoning is efficient and flexible. Representing knowledge in terms of evidential tests for or against particular roles enables a domain expert to rapidly construct, edit, and extend a knowledge base. Prior probabilities provide a base level of competence, to which the domain expert can quickly add. The ability to provide suppressive evidence facilitates the description of special cases. The knowledge base remains modular, so changes to one part of the knowledge base do not affect other parts.

In coaching applications, the evidential reasoner's lack of deductive soundness may be turned to our advantage, by forcing the student to reflect on the validity of the system's inferences, based on the evidence for and against that the system presents for each inference.

TNT provides us with the ability to realize a class of coaching software tools, but it tells us nothing about how to apply these tools, and this is the direction in which we intend to direct future research. Preliminary results with collaborators at Northwestern, the U.S. Naval Academy, and Oxford have made clear that integrating CyclePad, with its design-centric approach to thermodynamics, into existing thermodynamic curricula will require more work.

Simply making CyclePad available to a conventional thermodynamics class is tantamount to providing an arithmetic class with calculators. Most problems in thermodynamic textbooks ask the student to analyze a cycle in order to calculate a particular number, generally the cycle's efficiency. In contrast, a CyclePad problem might ask a student to compare the efficiencies of several different cycles, or the same cycle with several different working fluids and to explain observed differences. Without CyclePad, such an assignment would be impractical, given the amount of computation required, but with CyclePad we can focus the student's energies on the important conceptual issues.

At this time, open research questions include:

- What form should CyclePad problems take?
- How can we best integrate CyclePad into a conventional thermodynamics class?
- How and when should advice be presented to students to maximize the pedagogical impact of CyclePad?

To understand these issues, we are actively seeking engineering professors interested in developing design-centric curricula for thermodynamics. We anticipate that CyclePad and the TNT-based coaching systems will have to change significantly if they are to be integrated into such a curriculum; what we don't yet know is how they will change.

We are in the process of developing a pedagogical environment to support the use

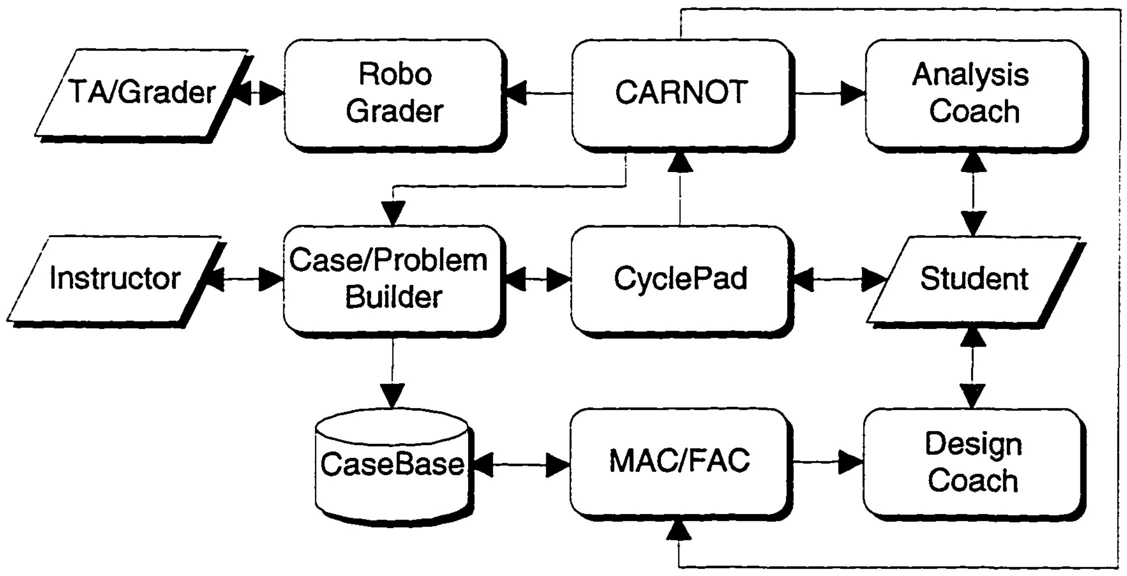


Figure 8.1 Architecture of CyclePad pedagogical support environment

of CyclePad in the field. This environment, depicted in Figure 8.1, will enable the creation of problems for use with CyclePad and provide assistance in their grading. CyclePad, shown in the center of the diagram, provides a common interface to the environment. The instructor interacts with an enhanced version of CyclePad that includes the Case/Problem Builder, a wizard-style set of dialog boxes for the step-by-step creation of a problem or case. Problems generated in this manner may be distributed electronically to students.

A student interacts with a version of CyclePad that includes the Design and Analysis Coaching subsystems. Loading a homework problem from the Case/Problem Builder causes CyclePad to set up the environment to suit the problem, for example setting the units in effect appropriately. The problem file contains both a machine-readable and a natural-language description of the situation that the student may view at any time and may, at the instructor's discretion contain either a fragment or a complete design for the student to work with. We intend to add a facility for the student to view how well her current state of analysis addresses the requirements set forth in the problem.

CyclePad contains an email facility that allows the student to send questions to a human or automated TA (e.g., RoboTA, a project that we are currently prototyping, based on the Design Coach) and to submit completed homework. Such a submission is first annotated with CARNOT's inferences and also with the RoboGrader's analysis, which compares the student's work to the problem requirements. The TA interacts with the system via a RoboGrader browser that summarizes and displays student results and automates basic grading calculations (e.g., score mean and distribution), given instructor- or TA-assigned point values for each part of the problem.

As we have found that a rich context is essential in the Design Coach, the Case/Problem Builder enables the creation of cases via a wizard-style interface that guides the instructor in specifying extra information, such as a design change and rationale for the change. The Design Coach will utilize MAC/FAC, a model of analogical retrieval, as described in Section 6.3, to retrieve case for presentation to the student. We are currently working on an email-based system, RoboTA, that will offer design and analytical advice in response to emailed queries. Utilizing email for RoboTA will enable us to maintain and improve the Design Coach on a centralized server, rather than requiring frequent updates to the CyclePad distribution. It will also enable us to gather data on how students are using CyclePad, which we anticipate will aid us in future development work.

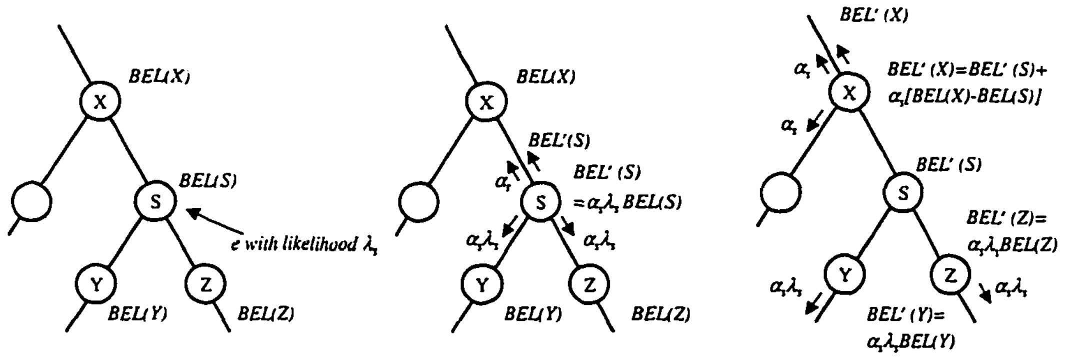
To conclude, we believe that the pedagogical tools TNT makes possible will enable us, in collaboration with thermodynamics teachers, to construct new curricula that have design as a central focus and demonstrably improve understanding of domain concepts. Finally, we believe there are other applications for TNT that we have yet to explore. For example, a TNT-based subsystem might be embedded CAD system for automating the function-based indexing and retrieval of designs. We also expect that our current applications of TNT will extend to other physical domains, such as electronics.

Appendix A
Hierarchical Belief Updating

CARNOT implements Pearl's propagation-based hierarchical belief-updating algorithm (Pearl 1988). This appendix summarizes the key points of this algorithm; for further information, see (Pearl 1988). The essence of the algorithm is captured in Figure A.1.

In part (a) of this diagram, a new piece of evidence with likelihood λ_s impacts belief in S. *BEL*(x) is a function that returns the current level of belief in a proposition x, that is, the probability that x is true given all evidence currently available. In part (b), messages transmitted to the parent and children of S update their probabilities, while in part (c) these messages complete their propagation to the root and leaves of the tree.

Alpha, defined in Equation A.1, is a normalizing factor that ensures that the



BEL(X)
X
BEL(S)
S
e with likelihood λ_s
Y
Z
BEL(Y)
BEL(Z)
BEL(X)
BEL'(S)
α_s
BEL' (S) $=\alpha_s\lambda_s$ BEL(S)
$\alpha_s\lambda_s$
$\alpha_s\lambda_s$
BEL(Y)
BEL(Z)
BEL' (X)
α_s
BEL' (X)=BEL' (S)+ α_s[BEL(X)-BEL(S)]
α_s
BEL' (S)
BEL' (Z)= $\alpha_s\lambda_s$BEL(Z)
$\alpha_s\lambda_s$
$\alpha_s\lambda_s$
BEL' (Y)= $\alpha_s\lambda_s$BEL(Y)

(a) New evidence impacts node S

(b) Messages transmitted to parent and children of S

(c) Messages transmitted to siblings, cousins and descendents of S

Source: (Pearl 1988, p339)

Figure A.1 Hierarchical updating of belief

probabilities at any given level of the hierarchy sum to the probability of the immediate parent node (and hence the probabilities across the entire hierarchy sum to 1).

$$\alpha_s = [\lambda_s BEL(R) + 1 - BEL(R)]^{-1}$$ **Equation A.1**

The updating process takes place in three steps. In the first step, the prior probability of the role for which the evidence applies is updated by multiplying it by the current likelihood and the normalizing factor:

$$BEL'(R) = \alpha_s \lambda_s BEL(R)$$ **Equation A.2**

or, in other words,

$$BEL'(R) = \frac{\lambda_S BEL(R)}{\lambda_S BEL(R) + 1 - BEL(R)}$$ **Equation A.3**

Messages are then transmitted up and down the hierarchy. The parent of R (i.e., its role abstraction) receives a message of three parts, the prior probability of R, the new probability of R as calculated in Equation A.3, and the normalizing factor alpha. All children of R (i.e. specializations of the role) receive a message of alpha times the likelihood.

In the second step, all children of R update their beliefs by multiplying them by the message received. They then pass this message down to all of their successors and this process repeats until it terminates at the leaves of the tree.

In the third step, a parent node P that receives the three part message from its child updates its belief via

$$BEL'(P) = BEL'(R) + \alpha_S[BEL(P) - BEL(R)] \qquad \textbf{Equation A.4}$$

The parent node then transmits to its parent a three-part message consisting of its prior belief, its newly-revised belief and alpha. It also transmits to its children a message alpha. Each child *C* that receives this message updates its belief by multiplying its current belief times alpha, or, in effect via

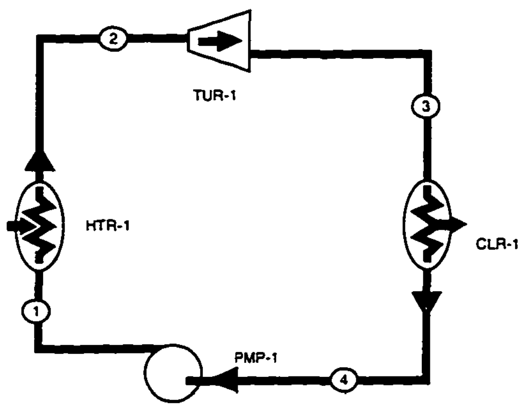
$$BEL'(C) = \frac{BEL(C)}{\lambda_S BEL(C) + 1 - BEL(C)} \qquad \textbf{Equation A.5}$$

Note that the difference between this equation and Equation A.3 is the lack of a λ_S term in the numerator. All nodes that are updated in this manner are siblings (or cousins) of the original node R, because a parent (or more distant ancestor) passed this message down to all of its other offspring. The λ_S is a weighting factor that encodes how much we should revise our belief in the original node R, so all nodes that are not a part of R's direct lineage should not be affected by this weighting. However, in order to ensure that the sum total of all probabilities remains 1, we must adjust all sibling probabilities either up or down depending on the adjustment to R. For example, if we find new evidence that a cooler CLR-1 is acting as an intercooler, then not only should our belief that CLR-1 has the role intercooler increase, but belief in its abstraction, heat-ejector should rise. At the same time, belief in its sibling fluid-cooler should decline, as should belief in the sibling to heat-ejector, heat-provider. As a consequence of this updating mechanism, each piece of evidence affects the probabilities of all roles. This results in swift convergence on particular role assignments for each device.

Appendix B
CARNOT-Generated Explanations for Test Set A

The following 36 cycles comprise the primary test set used in the development of CARNOT. The current version of CARNOT correctly identifies the function of each cycle and its component parts. The accompanying explanations are automatically generated based on plans instantiated during the processing of the cycle and have not been edited in any way. Note that this appendix only demonstrates one aspect of CARNOT's output; for examples of its other capabilities, see Chapters 2 and 6.

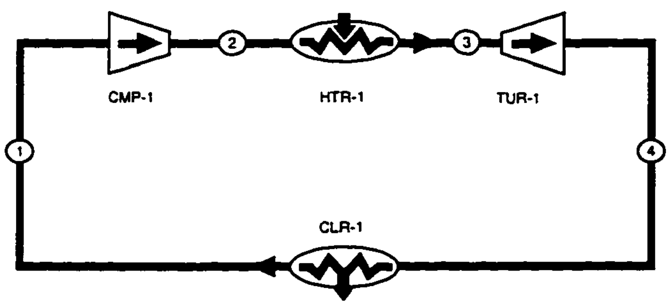
Cycle 1
Simple Steam Cycle



Source: Analysis of Engineering Cycles, Figure 1.1

This cycle is a heat-engine, so it is intended to produce a change in the environment, namely work. This heat engine fully condenses its working fluid in order to realize gains in pumping efficiency over compressing gas or saturated mixtures.

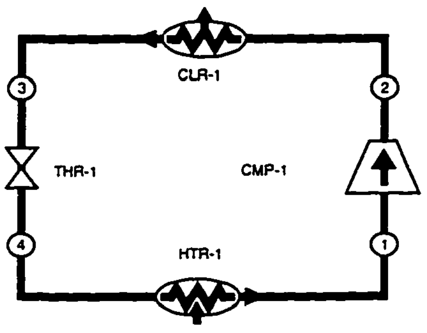
Cycle 2
Closed-Circuit Gas-Turbine



Source: Analysis of Engineering Cycles, Figure 1.4

This cycle is a heat-engine, so it is intended to produce a change in the environment, namely work. This heat engine utilizes gas working fluids throughout, which typically results in lower weight and a more compact design.

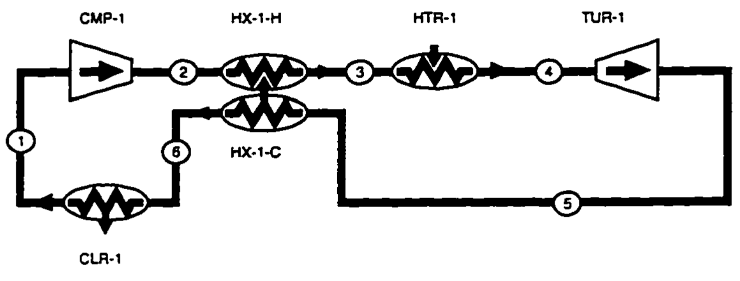
Cycle 3
Practical Vapor-Compression Plant



Source: Analysis of Engineering Cycles, Figure 5.3(a)

This is a refrigerator cycle, so it is intended to move heat from one location to another.

Cycle 4
CBTX Cycle with Exhaust-Gas Heat-Exchanger

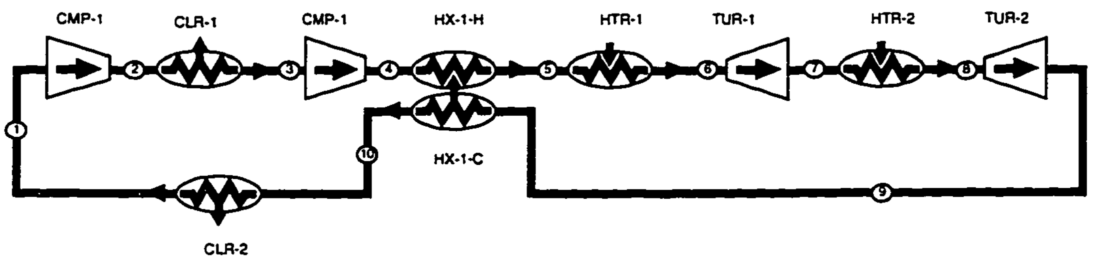


Source: Analysis of Engineering Cycles, Figure 6.2

This cycle is a heat-engine, so it is intended to produce a change in the environment, namely work. This heat engine utilizes gas working fluids throughout, which typically results in lower weight and a more compact design.

To increase this cycle's efficiency, the relatively hot exhaust gas is piped through a closed heat-exchanger in order to preheat the working fluid entering the combustion chamber. This takes place at HX-1H. This plan takes advantage of the high temperature exhaust of a gas-turbine to preheat the working fluid.

Cycle 5
CICBTRTX Cycle with Intercooler, Reheater, and Exhaust-Gas Heat-Exchanger

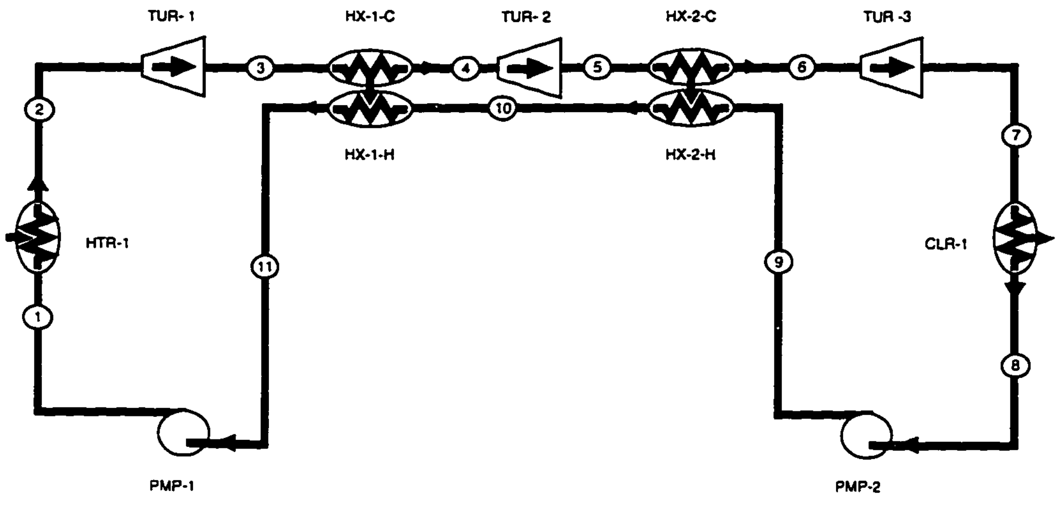


Source: Analysis of Engineering Cycles, Figure 6.6

This cycle is a heat-engine, so it is intended to produce a change in the environment, namely work. This heat engine utilizes gas working fluids throughout, which typically results in lower weight and a more compact design.

To increase this cycle's efficiency, energy is ejected from the compressor. This takes place at CLR-1. Intercooling reduces the amount of work necessary to compress a gas, thus increasing cycle efficiency. To increase this cycle's efficiency, the relatively hot exhaust gas is piped through a closed heat-exchanger in order to preheat the working fluid entering the combustion chamber. This takes place at HX-1H. This plan takes advantage of the high temperature exhaust of a gas-turbine to preheat the working fluid. To increase this cycle's efficiency, energy is injected into the working fluid as it flows through the turbine. This takes place at HTR-2. Reheating in a gas power cycle takes advantage of the fact that oxygen remains in the working fluid even after the initial combustion, so injection of more fuel enables further extraction of energy from the exhaust of the first turbine stages. In an airplane engine this would be called an afterburner.

Cycle 6
Reversible Regenerative Cycle for Saturated Steam, without Steam Extraction

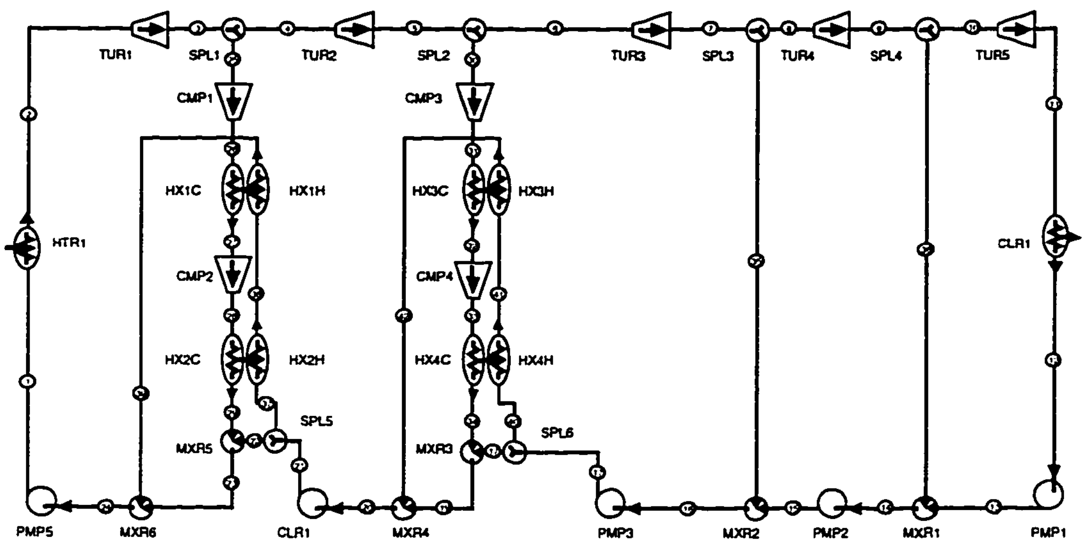


Source: Analysis of Engineering Cycles, Figure 7.3

This cycle is a heat-engine, so it is intended to produce a change in the environment, namely work. This heat engine fully condenses its working fluid in order to realize gains in pumping efficiency over compressing gas or saturated mixtures.

To increase this cycle's efficiency, there are 2 points (HX-2H and HX-1H) where the working fluid flowing through the turbine is used to preheat the feed fluid. Ideally we could extract heat from the turbine as the gas expands through it. In practice this causes unacceptable saturation of the fluid.

Cycle 7
Idealized Extraction Regenerative Superheated Cycle

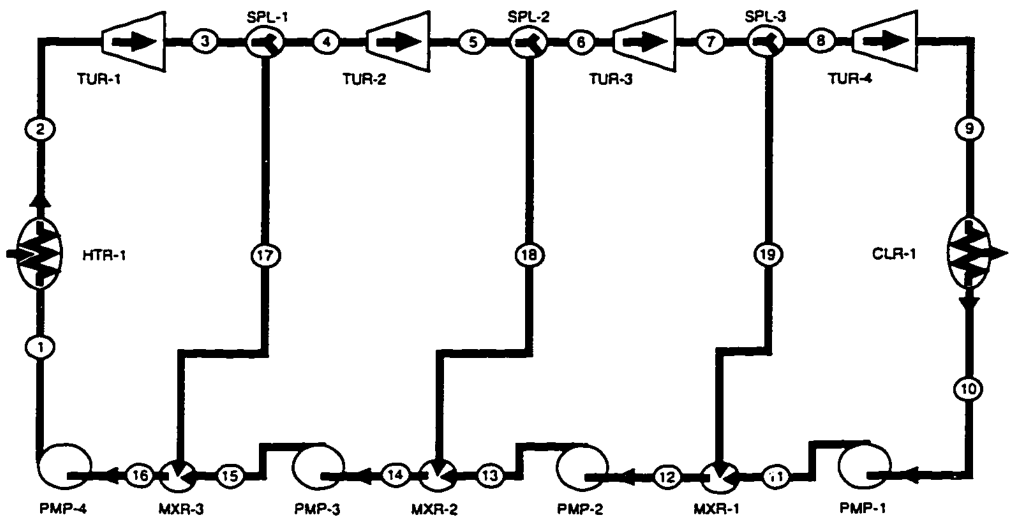


Source: Analysis of Engineering Cycles, Figure 7.4

This cycle is a heat-engine, so it is intended to produce a change in the environment, namely work. This heat engine combines the greater maximum temperatures of a gas cycle with the thermal advantage of the vapor cycle in ejecting all its heat at the lowest possible temperature.

To increase this cycle's efficiency, there are 2 points (MXR3 and MXR5) where superheated fluid bled from the turbine is saturated then is directly mixed with the boiler feed liquid. Direct contact is the most efficient form of heat exchange, but a large temperature difference in the streams will lead to irreversibilities. Increasing the pressure and intercooling a working-fluid bleed can convert it to a dry saturated gas at the same temperature as the wet saturated liquid to be preheated. There are also 2 points (MXR2 and MXR1) where fluid bled from the turbine is directly mixed with the boiler feed liquid. Heat-exchange via mixing is more efficient than indirect transfer in closed heat-exchangers, and it also enables deaeration of working fluid, but it requires more pumps because inlets must be maintained at the same pressure. There are also 2 points (HX3C and HX1C) where energy ejected from the compressor is used as a source of heat. Using the ejected heat from an intercooler increases the efficiency of intercooling.

Cycle 8
Plant with Train of DC Heaters

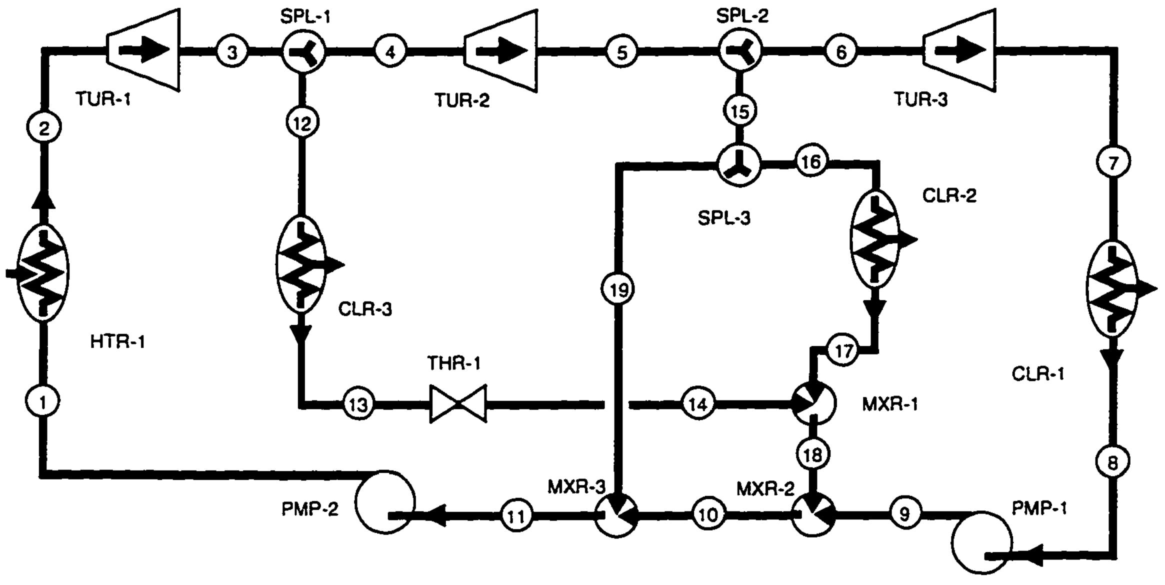


Source: Analysis of Engineering Cycles, Figure 7.8

This cycle is a heat-engine, so it is intended to produce a change in the environment, namely work. This heat engine fully condenses its working fluid in order to realize gains in pumping efficiency over compressing gas or saturated mixtures.

To increase this cycle's efficiency, there are 3 points (MXR-1, MXR-2, and MXR-3) where fluid bled from the turbine is directly mixed with the boiler feed liquid. Heat-exchange via mixing is more efficient than indirect transfer in closed heat-exchangers, and it also enables deaeration of working fluid, but it requires more pumps because inlets must be maintained at the same pressure.

Cycle 9
Steam-Turbine Plant for Combined Supply of Power and Process Steam

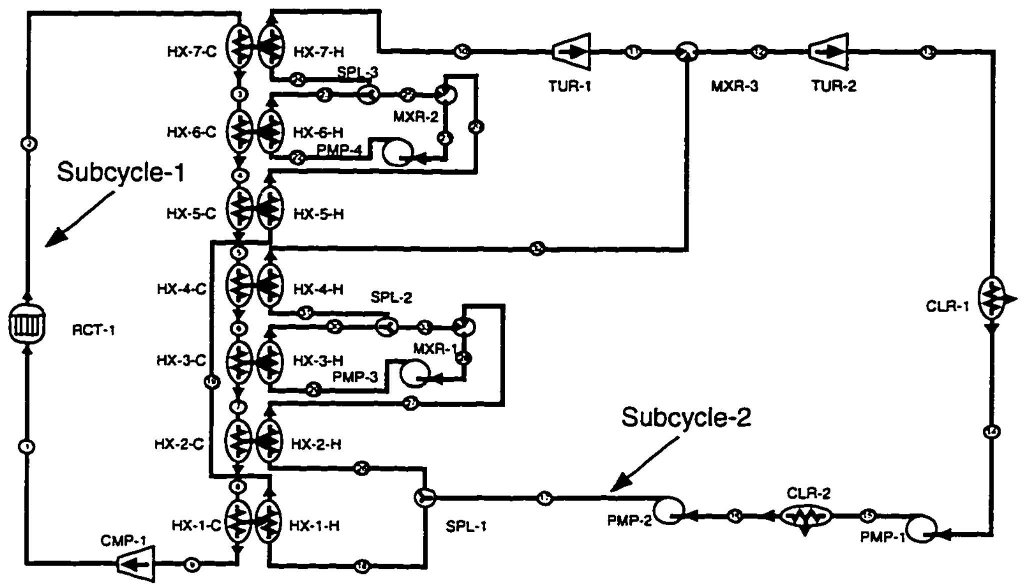


Source: Analysis of Engineering Cycles, Figure 7.11

This cycle is a heat-engine, so it is intended to produce a change in the environment, namely work. This heat engine fully condenses its working fluid in order to realize gains in pumping efficiency over compressing gas or saturated mixtures.

To increase this cycle's efficiency, there are 2 points (CLR3 and CLR2) where vapor is bled for use in an industrial process. Conversion of energy from one form to another inevitably entails irreversibility, so using vapor generated for a power cycle is more efficient than using the power generated to produce the vapor. To increase this cycle's efficiency, fluid bled from the turbine is directly mixed with the boiler feed liquid. This takes place at MXR3. Heat-exchange via mixing is more efficient than indirect transfer in closed heat-exchangers, and it also enables deaeration of working fluid, but it requires more pumps because inlets must be maintained at the same pressure.

Cycle 10
Gas-Cooled Reactor with Dual-Pressure Steam Cycle



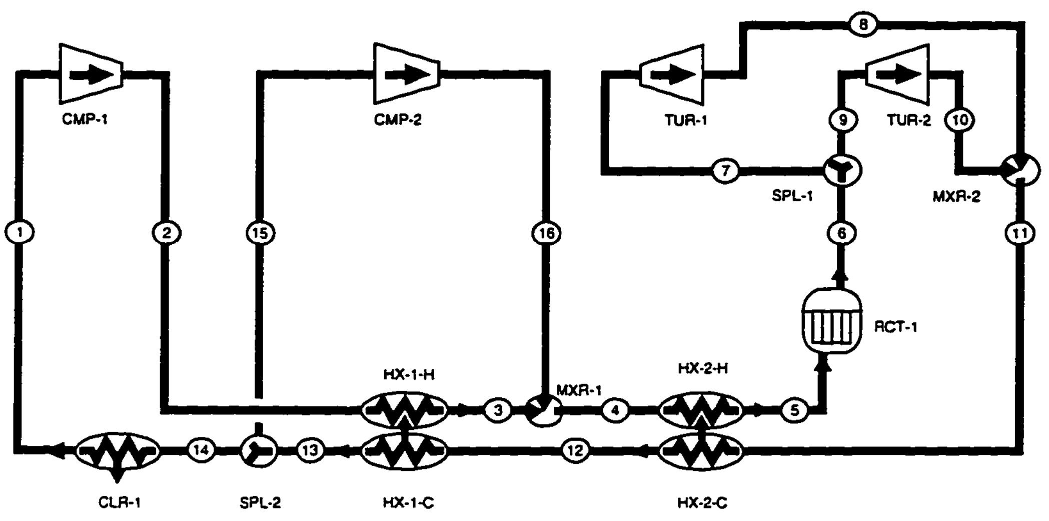
Source: Analysis of Engineering Cycles, Figure 8.1

This cycle is a heat-engine, so it is intended to produce a change in the environment, namely work.

To increase this cycle's efficiency, heat is injected into the work-generating subcycle at multiple pressures. This takes place at SUBCYCLE-1/SUBCYCLE-2. Heat-exchange at multiple-pressures minimizes the temperature difference between the two working fluids, thereby creating less irreversibility. A relatively cool, high-pressure saturated liquid can absorb heat from a high-temperature working fluid, cooling that fluid to the point where it can supply heat to the same saturated liquid at a lower temperature.

To preserve this cycle's integrity, a separate subcycle is used to contain the radiation of the reactor core. This takes place at SUBCYCLE-1. Nuclear plants often use a separate subcycle to contain radiation. There are also 2 points (MXR2/SPL3 and MXR1/SPL2) where a vapor-drum is used to maintain a ready supply of gaseous working fluid. A vapor-drum buffers the system against sudden changes in load.

Cycle 11
Hypercritical CO2 Cycle

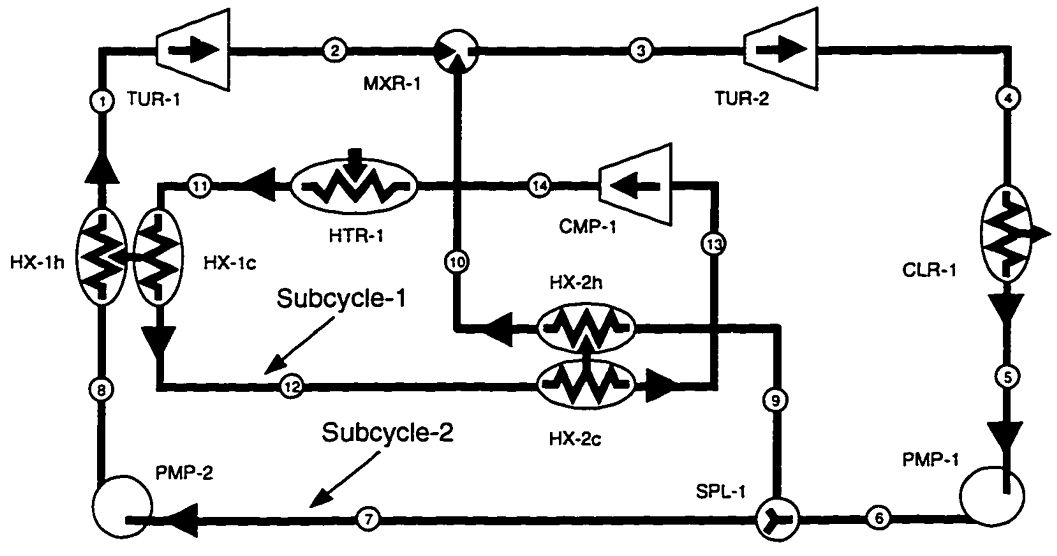


Source: Analysis of Engineering Cycles. Figure 8.6

This cycle is a heat-engine, so it is intended to produce a change in the environment, namely work. This heat engine utilizes gas working fluids throughout, which typically results in lower weight and a more compact design.

To increase this cycle's efficiency, the relatively hot exhaust gas is piped through a closed heat-exchanger in order to preheat the working fluid entering the combustion chamber. This takes place at HX-2H. This plan takes advantage of the high temperature exhaust of a gas-turbine to preheat the working fluid.

Cycle 12
Ideal Dual-Pressure Work-Producing Device

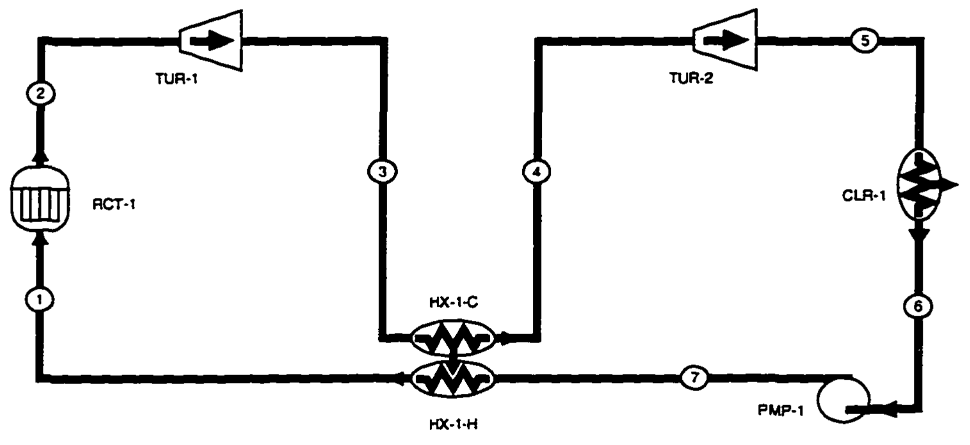


Source: Analysis of Engineering Cycles. Figure 8.3

This cycle is a heat-engine, so it is intended to produce a change in the environment, namely work.

To increase this cycle's efficiency, heat is injected into the work-generating subcycle at multiple pressures. This takes place at SUBCYCLE-1/SUBCYCLE-2. Heat-exchange at multiple-pressures minimizes the temperature difference between the two working fluids, thereby creating less irreversibility. A relatively cool, high-pressure saturated liquid can absorb heat from a high-temperature working fluid, cooling that fluid to the point where it can supply heat to the same saturated liquid at a lower temperature.

Cycle 13
Supercritical Cc:ıdensing N2O4 Cycle

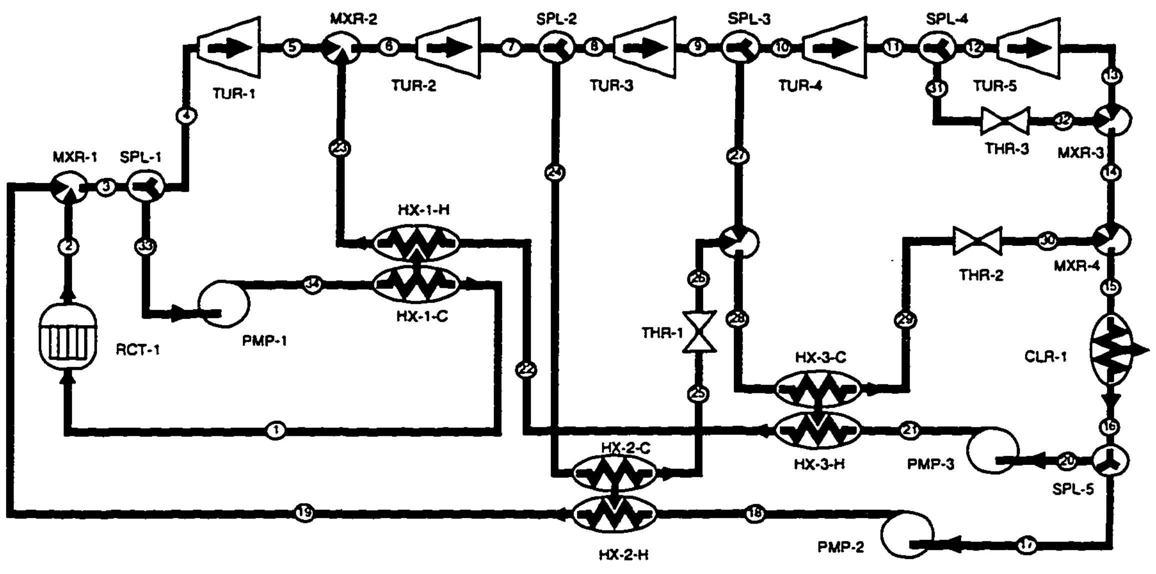


Source: Analysis of Engineering Cycles, Figure 8.7

This cycle is a heat-engine, so it is intended to produce a change in the environment, namely work. This heat engine fully condenses its working fluid in order to realize gains in pumping efficiency over compressing gas or saturated mixtures.

To increase this cycle's efficiency, the working fluid flowing through the turbine is used to preheat the feed fluid. This takes place at HX-1H. Ideally we could extract heat from the turbine as the gas expands through it. In practice this causes unacceptable saturation of the fluid.

Cycle 14
Direct-Cycle Dual-Pressure Boiling-Water Reactor without Superheat



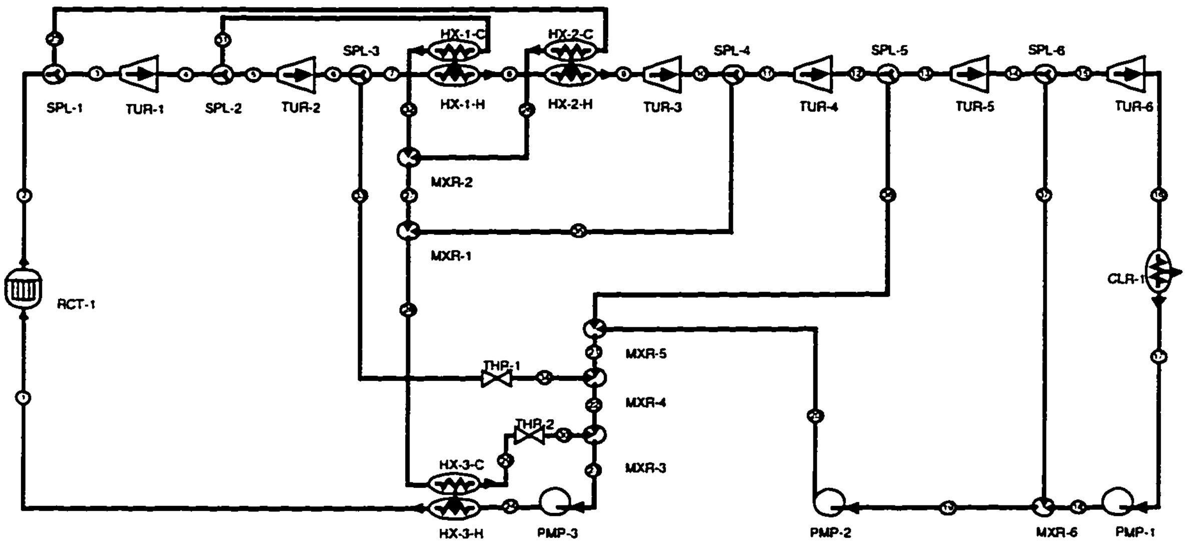
Source: Analysis of Engineering Cycles, Figure 8.8

This cycle is a heat-engine, so it is intended to produce a change in the environment, namely work.

To increase this cycle's efficiency, there are 2 points (HX-3H and HX-2H) where working fluid bled from the turbine is used to supply heat in a heat-exchanger. Heat-exchange via closed heat-exchangers is less efficient than direct mixing, but the two working fluids can be at different pressures, reducing the need for pumps. To increase this cycle's efficiency, working fluid is used to cool the fluid flowing through the reactor core. This takes place at HX-1C. Liquid-moderated reactors may be controlled by cooling their inlet working-fluid, which will result in the raising of more vapor.

To preserve this cycle's integrity, the working fluid flowing through the turbine is flashed to improve its quality. This takes place at SPL-4. Reactors often generate vapor of lower quality, and hence may need flash-chambers in the turbine series to dry the vapor entering the latter stages. To preserve this cycle's integrity, a vapor-drum is used to maintain a ready supply of gaseous working fluid. This takes place at MXR-1/SPL-1. A vapor-drum buffers the system against sudden changes in load.

Cycle 15
Direct-Cycle Boiling-Water Reactor with Live and Bled-Steam Reheating



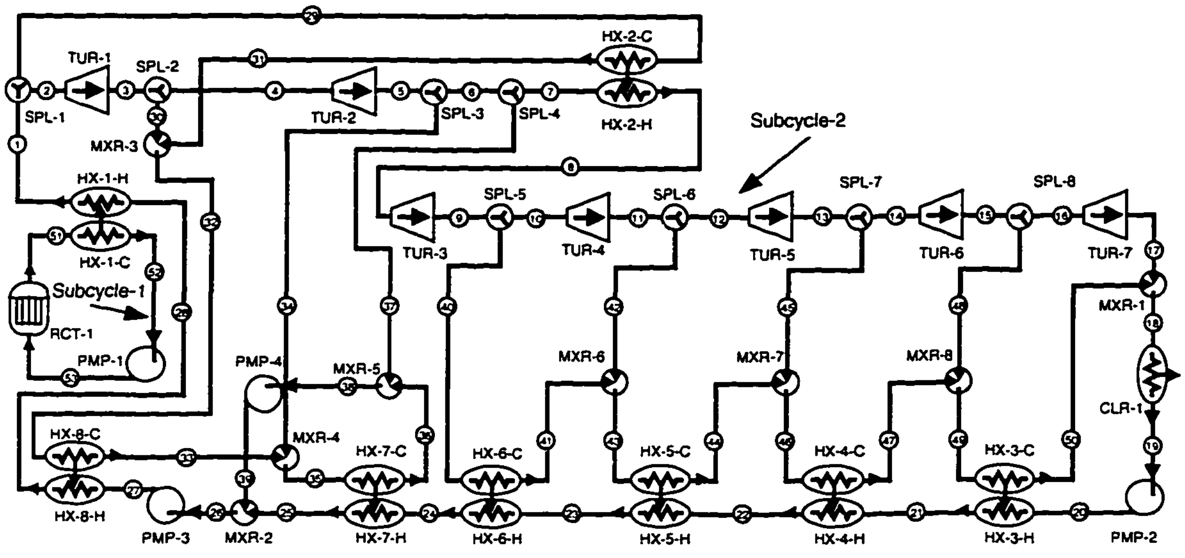
Source: Analysis of Engineering Cycles, Figure 8.9

This cycle is a heat-engine, so it is intended to produce a change in the environment, namely work. This heat engine fully condenses its working fluid in order to realize gains in pumping efficiency over compressing gas or saturated mixtures.

To increase this cycle's efficiency, working fluid bled from the turbine is used to supply heat in a heat-exchanger. This takes place at HX-3H. Heat-exchange via closed heat-exchangers is less efficient than direct mixing, but the two working fluids can be at different pressures, reducing the need for pumps. There are also 2 points (MXR-5 and MXR-6) where fluid bled from the turbine is directly mixed with the boiler feed liquid. Heat-exchange via mixing is more efficient than indirect transfer in closed heat-exchangers, and it also enables deaeration of working fluid, but it requires more pumps because inlets must be maintained at the same pressure. There are also 2 points (HX-2H and HX-1H) where energy is injected into the working fluid as it flows through the turbine. Reheating in a vapor power cycle enables the use of higher turbine inlet pressures, which increases efficiency. Reheat ensures that the exit vapor is of high quality. If turbine blades could withstand higher inlet temperatures then reheat would be unnecessary.

To preserve this cycle's integrity, the working fluid flowing through the turbine is flashed to improve its quality. This takes place at SPL-3. Reactors often generate vapor of lower quality, and hence may need flash-chambers in the turbine series to dry the vapor entering the latter stages.

Cycle 16
Indirect-Cycle Pressurized-Water Reactor with Live-Steam Reheating



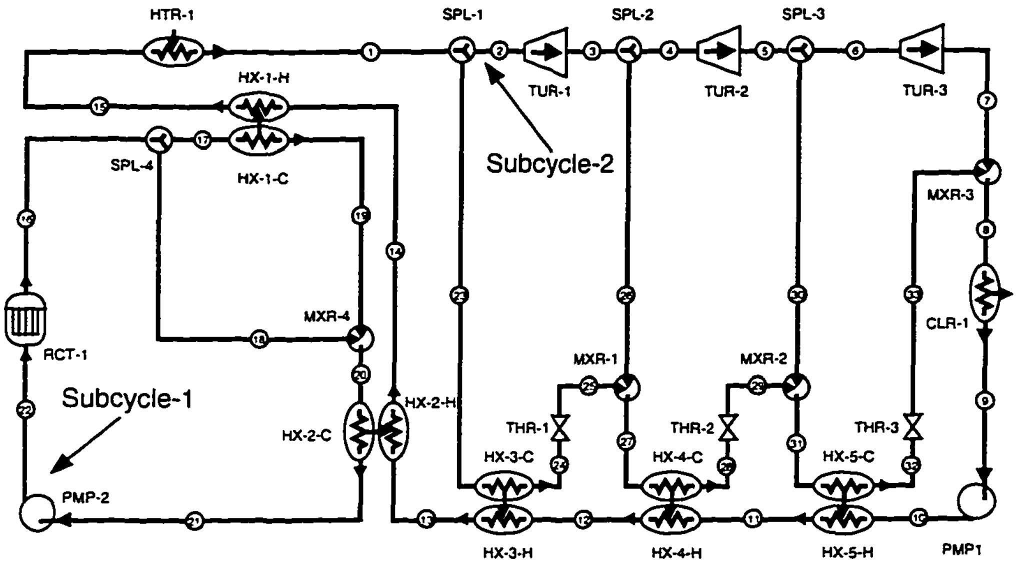
Source: Analysis of Engineering Cycles, Figure 8.10

This cycle is a heat-engine, so it is intended to produce a change in the environment, namely work. This heat engine fully condenses its working fluid in order to realize gains in pumping efficiency over compressing gas or saturated mixtures.

To increase this cycle's efficiency, there are 6 points (HX4H, HX3H, HX5H, HX6H, HX7H, and HX8H) where working fluid bled from the turbine is used to supply heat in a heat-exchanger. Heat-exchange via closed heat-exchangers is less efficient than direct mixing, but the two working fluids can be at different pressures, reducing the need for pumps. To increase this cycle's efficiency, energy is injected into the working fluid as it flows through the turbine. This takes place at HX2H. Reheating in a vapor power cycle enables the use of higher turbine inlet pressures, which increases efficiency. Reheat ensures that the exit vapor is of high quality. If turbine blades could withstand higher inlet temperatures then reheat would be unnecessary.

To preserve this cycle's integrity, the working fluid flowing through the turbine is flashed to improve its quality. This takes place at SPL4. Reactors often generate vapor of lower quality, and hence may need flash-chambers in the turbine series to dry the vapor entering the latter stages. To preserve this cycle's integrity, a separate subcycle is used to contain the radiation of the reactor core. This takes place at SUBCYCLE-1. Nuclear plants often use a separate subcycle to contain radiation.

Cycle 17
Indirect-Cycle Boiling-Water Reactor with Secondary Nuclear Superheater



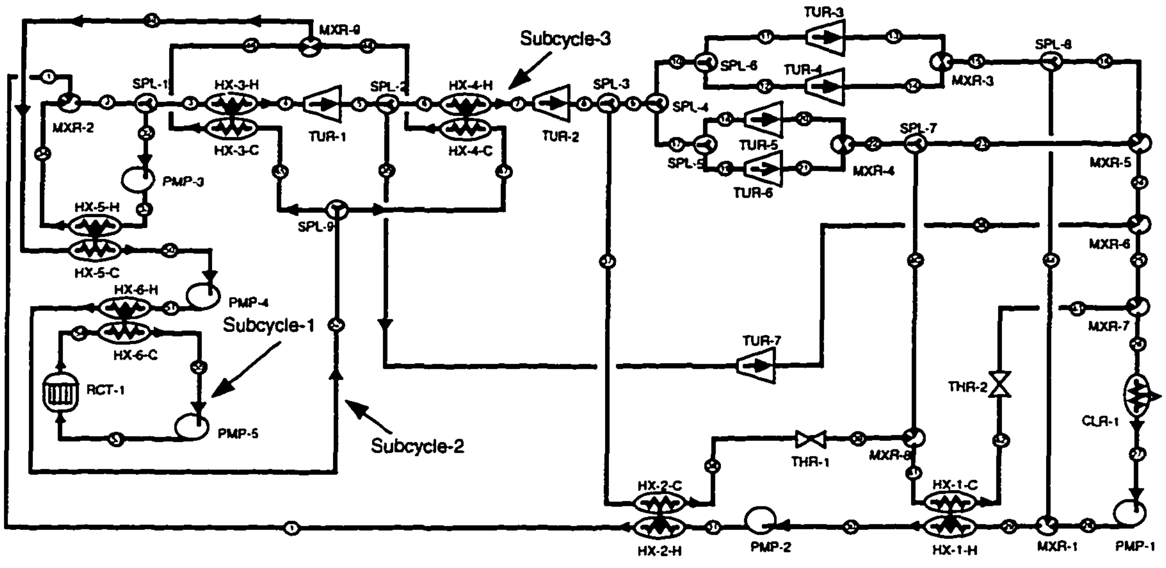
Source: Analysis of Engineering Cycles, Figure 8.11

This cycle is a heat-engine, so it is intended to produce a change in the environment, namely work. This heat engine fully condenses its working fluid in order to realize gains in pumping efficiency over compressing gas or saturated mixtures.

To increase this cycle's efficiency, there are 3 points (HX5H, HX4H, and HX3H) where working fluid bled from the turbine is used to supply heat in a heat-exchanger. Heat-exchange via closed heat-exchangers is less efficient than direct mixing, but the two working fluids can be at different pressures, reducing the need for pumps.

To preserve this cycle's integrity, a separate subcycle is used to contain the radiation of the reactor core. This takes place at SUBCYCLE-1. Nuclear plants often use a separate subcycle to contain radiation.

Cycle 18
Indirect-Cycle Sodium-Cooled Prototype Fast Reactor with Superheating and Reheating



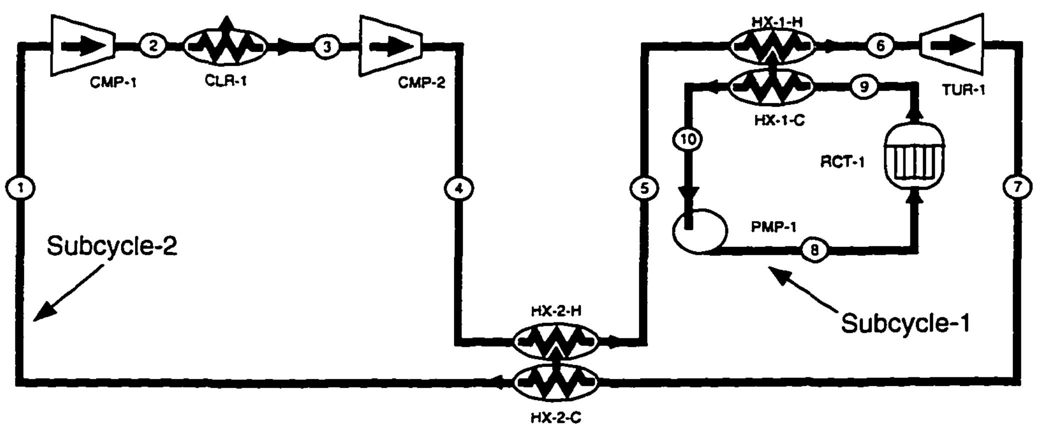
Source: Analysis of Engineering Cycles, Figure 8.12

This cycle is a heat-engine, so it is intended to produce a change in the environment, namely work. This heat engine fully condenses its working fluid in order to realize gains in pumping efficiency over compressing gas or saturated mixtures.

To increase this cycle's efficiency, heat is injected into the work-generating subcycle at multiple pressures. This takes place at SUBCYCLE-2/SUBCYCLE-3. Heat-exchange at multiple-pressures minimizes the temperature difference between the two working fluids, thereby creating less irreversibility. A relatively cool, high-pressure saturated liquid can absorb heat from a high-temperature working fluid, cooling that fluid to the point where it can supply heat to the same saturated liquid at a lower temperature. There are also 2 points (HX-2H and HX-1H) where working fluid bled from the turbine is used to supply heat in a heat-exchanger. Heat-exchange via closed heat-exchangers is less efficient than direct mixing, but the two working fluids can be at different pressures, reducing the need for pumps. To increase this cycle's efficiency, energy is injected into the working fluid as it flows through the turbine. This takes place at HX-4H. Reheating in a vapor power cycle enables the use of higher turbine inlet pressures, which increases efficiency. Reheat ensures that the exit vapor is of high quality. If turbine blades could withstand higher inlet temperatures then reheat would be unnecessary.

To preserve this cycle's integrity, a separate subcycle is used to contain the radiation of the reactor core. This takes place at SUBCYCLE-1. Nuclear plants often use a separate subcycle to contain radiation. To preserve this cycle's integrity, a vapor-drum is used to maintain a ready supply of gaseous working fluid. This takes place at MXR-2/SPL-1. A vapor-drum buffers the system against sudden changes in load.

Cycle 19
Closed-Circuit Gas-Turbine for Use with Nuclear Power Plant



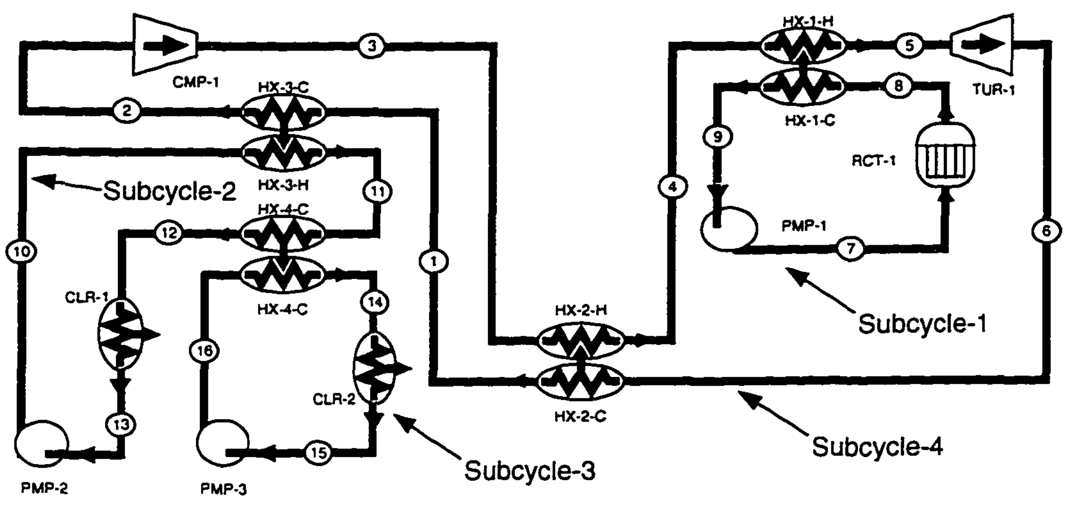
Source: Analysis of Engineering Cycles, Figure 8.13

This cycle is a heat-engine, so it is intended to produce a change in the environment, namely work.

To increase this cycle's efficiency, the relatively hot exhaust gas is piped through a closed heat-exchanger in order to preheat the working fluid entering the combustion chamber. This takes place at HX1H. This plan takes advantage of the high temperature exhaust of a gas-turbine to preheat the working fluid. To increase this cycle's efficiency, energy is ejected from the compressor. This takes place at CLR1. Intercooling reduces the amount of work necessary to compress a gas, thus increasing cycle efficiency.

To preserve this cycle's integrity, a separate subcycle is used to contain the radiation of the reactor core. This takes place at SUBCYCLE-1. Nuclear plants often use a separate subcycle to contain radiation.

Cycle 20
HTGR Gas-Turbine, Direct-Cycle Plant with Reject Heat Utilized for District Heating



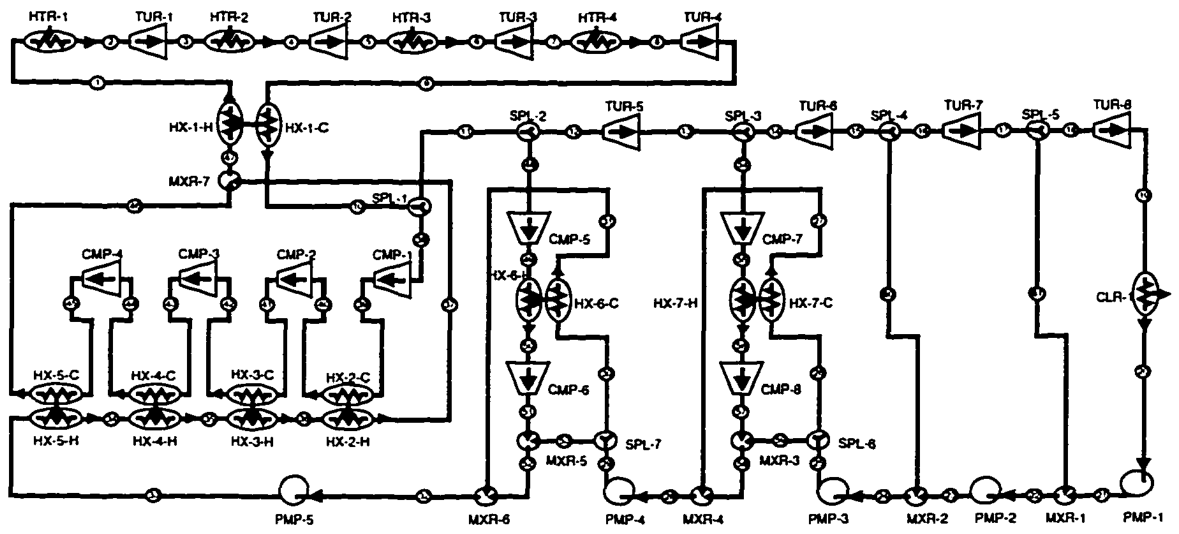
Source: Analysis of Engineering Cycles, Figure 8.14

This cycle is a heat-engine, so it is intended to produce a change in the environment, namely work.

To increase this cycle's efficiency, the relatively hot exhaust gas is piped through a closed heat-exchanger in order to preheat the working fluid entering the combustion chamber. This takes place at HX2H. This plan takes advantage of the high temperature exhaust of a gas-turbine to preheat the working fluid. To increase this cycle's efficiency, subcycles are employed to transport heat to another location. This takes place at SUBCYCLE-4, SUBCYCLE-3, and SUBCYCLE-2. Heat-engines generally convert energy into electrical form for efficient distribution. However, in some situations it may be more efficient to transport heat energy from the system directly to its point of use, thereby avoiding conversion inefficiencies.

To preserve this cycle's integrity, a separate subcycle is used to contain the radiation of the reactor core. This takes place at SUBCYCLE-1. Nuclear plants often use a separate subcycle to contain radiation.

Cycle 21
Ideal Super-Regenerative Cycle

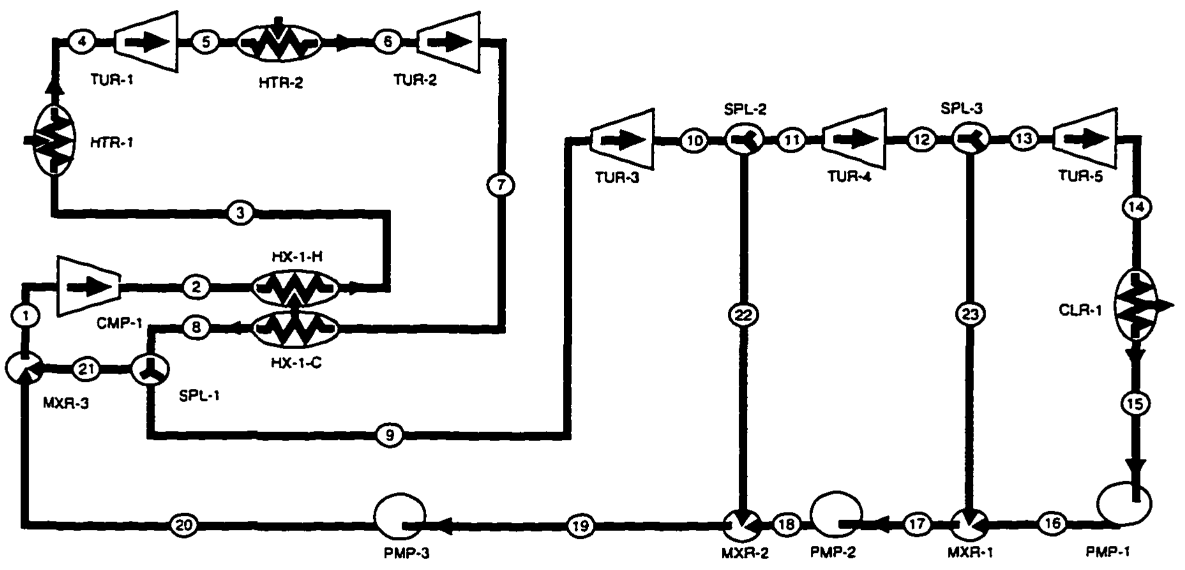


Source: Analysis of Engineering Cycles, Figure 9.1

This cycle is a heat-engine, so it is intended to produce a change in the environment, namely work. This heat engine combines the greater maximum temperatures of a gas cycle with the thermal advantage of the vapor cycle in ejecting all its heat at the lowest possible temperature.

To increase this cycle's efficiency, there are 5 points (HX2C, HX3C, HX4C, HX6C, and HX8C) where energy ejected from the compressor is used as a source of heat. Using the ejected heat from an intercooler increases the efficiency of intercooling. There are also 2 points (MXR1 and MXR2) where fluid bled from the turbine is directly mixed with the boiler feed liquid. Heat-exchange via mixing is more efficient than indirect transfer in closed heat-exchangers, and it also enables deaeration of working fluid, but it requires more pumps because inlets must be maintained at the same pressure. There are also 3 points (MXR5, MXR3, and MXR7) where superheated fluid bled from the turbine is saturated then is directly mixed with the boiler feed liquid. Direct contact is the most efficient form of heat exchange, but a large temperature difference in the streams will lead to irreversibilities. Increasing the pressure and intercooling a working-fluid bleed can convert it to a dry saturated gas at the same temperature as the wet saturated liquid to be preheated.

Cycle 22
Field Super-Regenerative Cycle

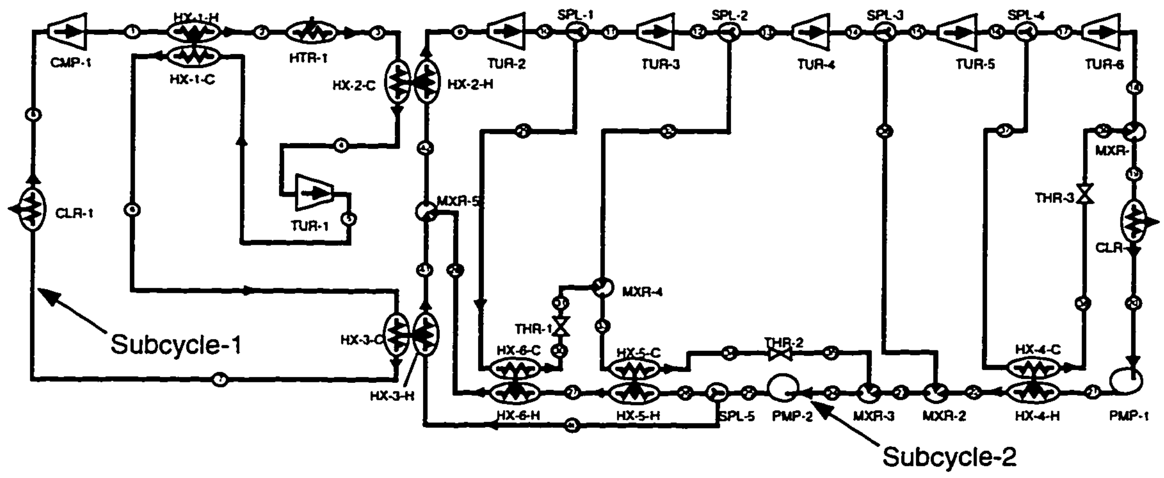


Source: Analysis of Engineering Cycles. Figure 9.3

This cycle is a heat-engine, so it is intended to produce a change in the environment, namely work. This heat engine combines the greater maximum temperatures of a gas cycle with the thermal advantage of the vapor cycle in ejecting all its heat at the lowest possible temperature.

To increase this cycle's efficiency, there are 3 points (MXR-2, MXR-1, and MXR-3) where fluid bled from the turbine is directly mixed with the boiler feed liquid. Heat-exchange via mixing is more efficient than indirect transfer in closed heat-exchangers, and it also enables deaeration of working fluid, but it requires more pumps because inlets must be maintained at the same pressure.

Cycle 23
Combined Gas and Steam Turbine Plant

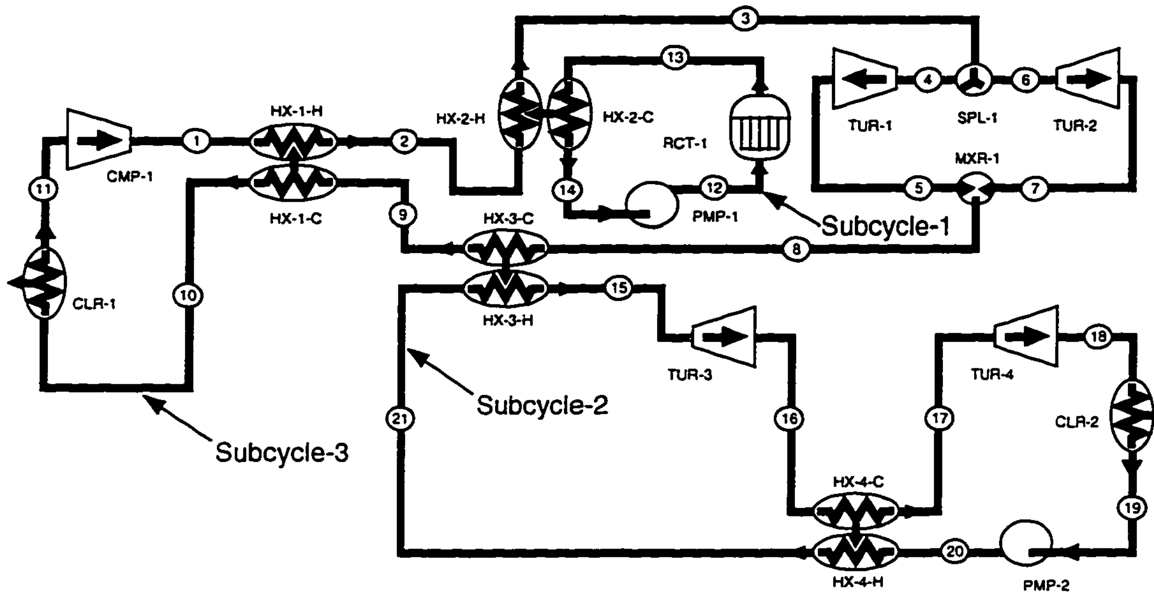


Source: Analysis of Engineering Cycles, Figure 9.5

This cycle is a heat-engine, so it is intended to produce a change in the environment, namely work.

To increase this cycle's efficiency, fluid bled from the turbine is directly mixed with the boiler feed liquid. This takes place at MXR-2. Heat-exchange via mixing is more efficient than indirect transfer in closed heat-exchangers, and it also enables deaeration of working fluid, but it requires more pumps because inlets must be maintained at the same pressure. There are also 3 points (HX-5H, HX-6H, and HX-4H) where working fluid bled from the turbine is used to supply heat in a heat-exchanger. Heat-exchange via closed heat-exchangers is less efficient than direct mixing, but the two working fluids can be at different pressures, reducing the need for pumps. To increase this cycle's efficiency, it is partitioned into two subcycles, one powering the other via its ejected heat. This takes place at SUBCYCLE-1/SUBCYCLE-2. Cascading cycles enables the use of working fluids with different thermal characteristics in the same system. In this case a high-temperature gas cycle's waste heat is used to power a vapor cycle.

Cycle 24
Gas-Turbine/Steam-Turbine Binary Cycle



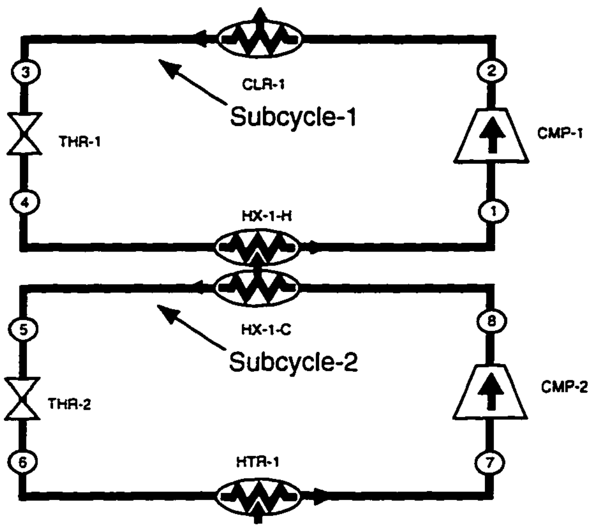
Source: Analysis of Engineering Cycles, Figure 9.9

This cycle is a heat-engine, so it is intended to produce a change in the environment, namely work.

To increase this cycle's efficiency, the working fluid flowing through the turbine is used to preheat the feed fluid. This takes place at HX-4H. Ideally we could extract heat from the turbine as the gas expands through it. In practice this causes unacceptable saturation of the fluid. To increase this cycle's efficiency, it is partitioned into two subcycles, one powering the other via its ejected heat. This takes place at SUBCYCLE-3/SUBCYCLE-2. Cascading cycles enables the use of working fluids with different thermal characteristics in the same system. In this case a high-temperature gas cycle's waste heat is used to power a vapor cycle.

To preserve this cycle's integrity, a separate subcycle is used to contain the radiation of the reactor core. This takes place at SUBCYCLE-1. Nuclear plants often use a separate subcycle to contain radiation.

Cycle 25
Binary Cascade Refrigeration Cycle

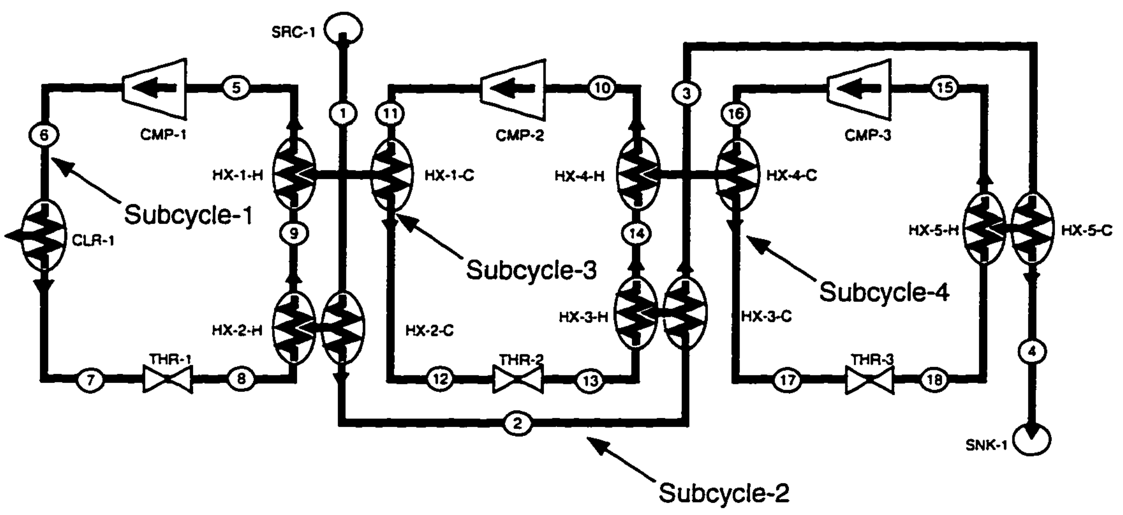


Source: Analysis of Engineering Cycles, Figure 10.2

This is a refrigerator cycle, so it is intended to move heat from one location to another.

To increase this cycle's efficiency, it is partitioned into multiple subcycles, one removing heat from the other. This takes place at SUBCYCLE-1/SUBCYCLE-2. Cascading cycles enables the use of working fluids with different thermal characteristics in the same system. In this case the energy-removing subcycle can use a refrigerant that operates at substantially different pressures in order to achieve greater heat-transfer.

Cycle 26
Ternary Cascade Refrigeration Plant for the Liquefaction of Natural Gas

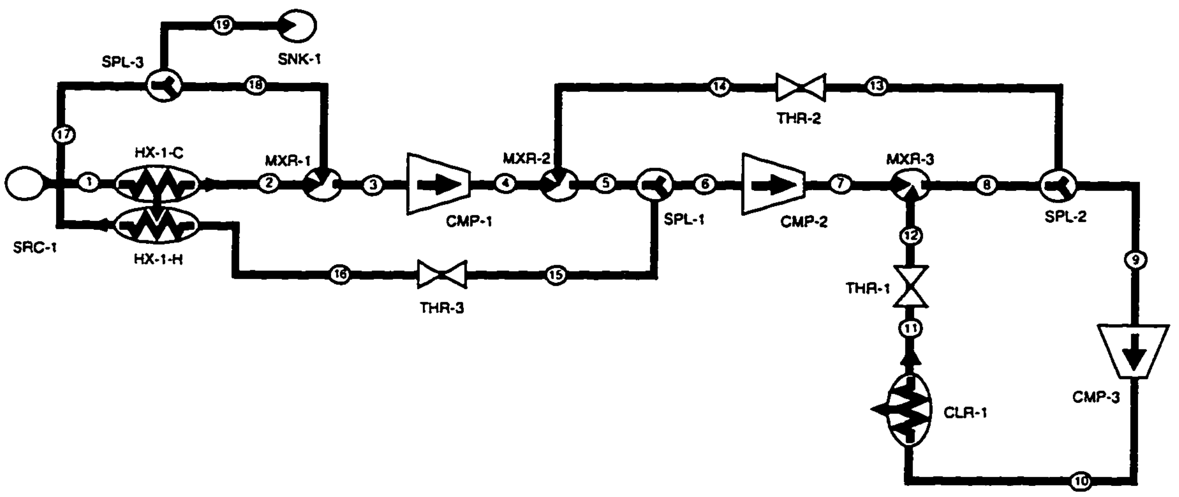


Source: Analysis of Engineering Cycles, Figure 10.3

This is a refrigerator cycle, so it is intended to move heat from one location to another. In this case the heat is being ejected in order to liquefy the fluid flowing from source to sink.

To increase this cycle's efficiency, there are 2 points (SUBCYCLE-1/SUBCYCLE-3 and SUBCYCLE-3/SUBCYCLE-4) where it is partitioned into multiple subcycles, one removing heat from the other. Cascading cycles enables the use of working fluids with different thermal characteristics in the same system. In this case the energy-removing subcycle can use a refrigerant that operates at substantially different pressures in order to achieve greater heat-transfer.

Cycle 27
Three-Stage Cascade Plant for the Production of Solid Carbon Dioxide (Dry Ice)

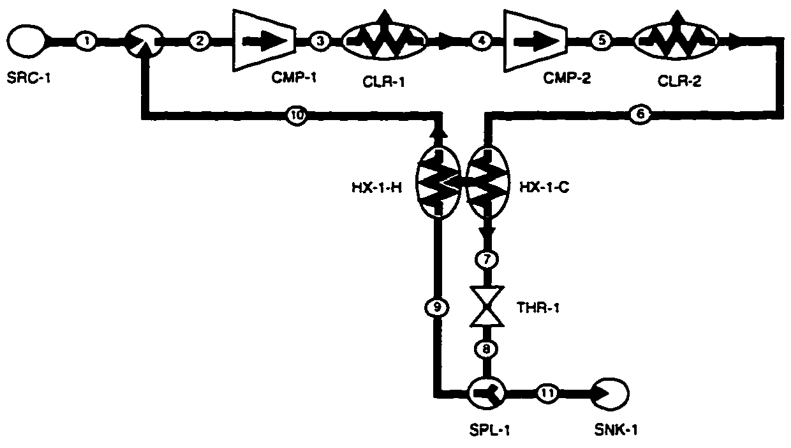


Source: Analysis of Engineering Cycles, Figure 10.5

This is a refrigerator cycle, so it is intended to move heat from one location to another. In this case the heat is being ejected in order to liquefy the fluid flowing from source to sink.

To increase this cycle's efficiency, there are 2 points (MXR3/SPL2 and MXR2/SPL1) where a flash-tank is used to cool the working fluid. A flash-tank is an efficient means for cooling the working fluid by direct contact.

Cycle 28
Simple Linde Liquefaction Process

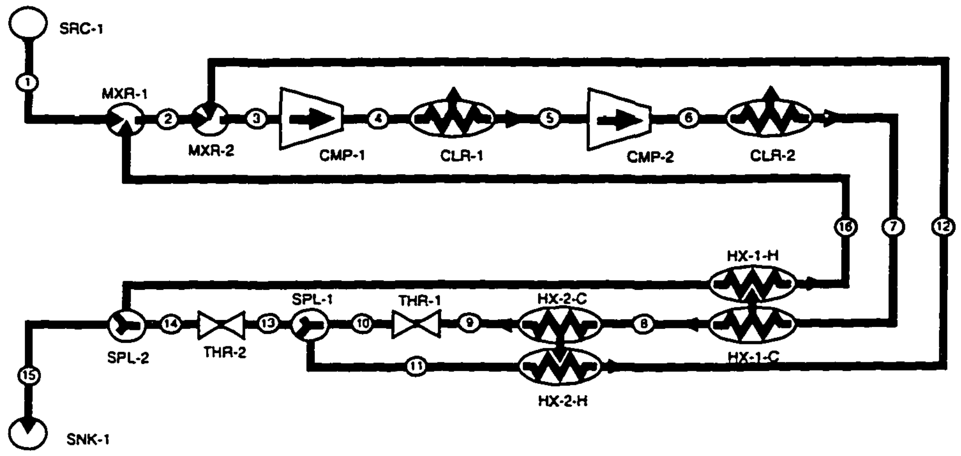


Source: Analysis of Engineering Cycles, Figure 10.8

This is a refrigerator cycle, so it is intended to move heat from one location to another. In this case the heat is being ejected in order to liquefy the fluid flowing from source to sink.

To increase this cycle's efficiency, energy is ejected from the compressor. This takes place at CLR-1. Intercooling reduces the amount of work necessary to compress a gas, thus increasing cycle efficiency.

Cycle 29
Linde Dual-Pressure (Cascade) Liquefaction Process

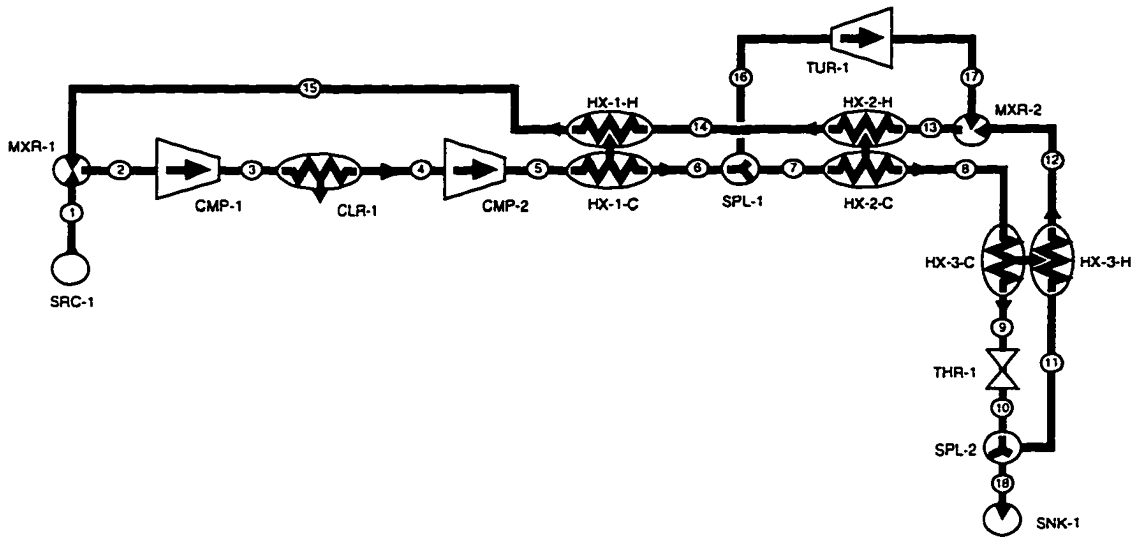


Source: Analysis of Engineering Cycles. Figure 10.8

This is a refrigerator cycle, so it is intended to move heat from one location to another. In this case the heat is being ejected in order to liquefy the fluid flowing from source to sink.

To increase this cycle's efficiency, energy is ejected from the compressor. This takes place at CLR-1. Intercooling reduces the amount of work necessary to compress a gas, thus increasing cycle efficiency.

Cycle 30
Claude Liquefaction Process

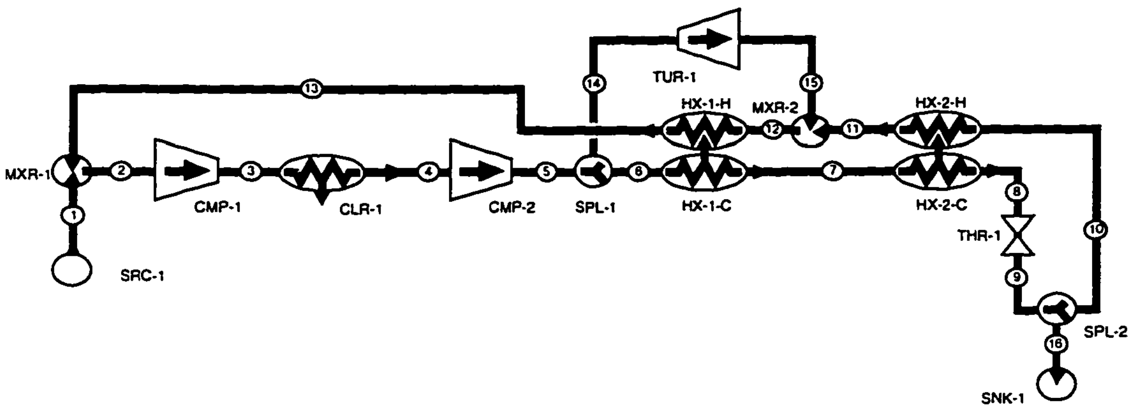


Source: Analysis of Engineering Cycles, Figure 10.11(a)

This is a refrigerator cycle, so it is intended to move heat from one location to another. In this case the heat is being ejected in order to liquefy the fluid flowing from source to sink.

To increase this cycle's efficiency, energy is ejected from the compressor. This takes place at CLR-1. Intercooling reduces the amount of work necessary to compress a gas, thus increasing cycle efficiency. To increase this cycle's efficiency, a turbine is used to cool the working fluid. This takes place at TUR-1. Using a turbine in place of a throttle to create the pressure drop in a refrigerator results in a greater decline in temperature, which is often useful in a cryogenic cycle.

Cycle 31
Heylandt Liquefaction Process

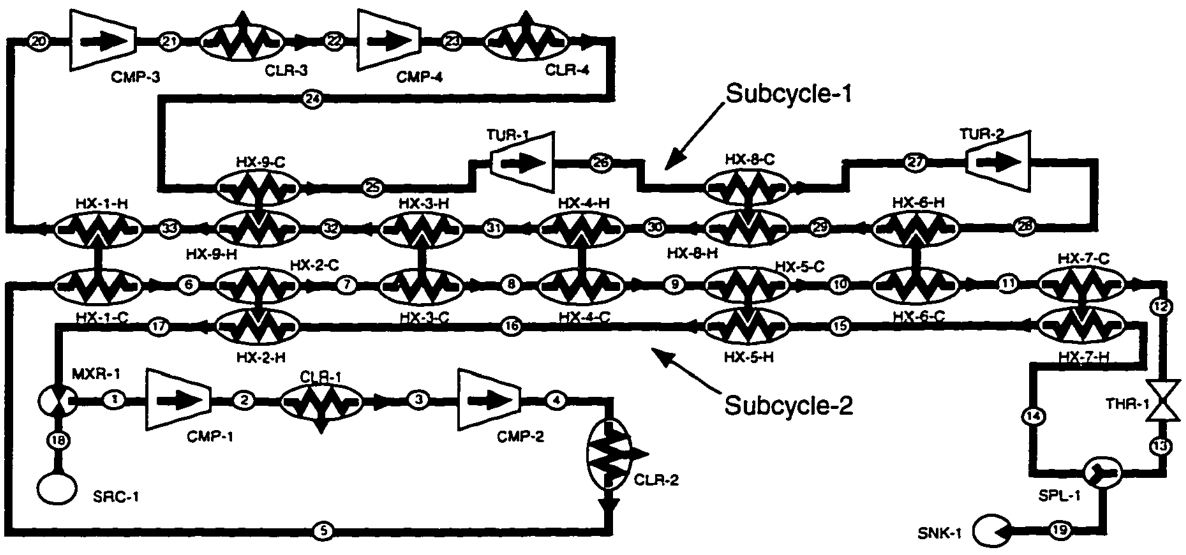


Source: Analysis of Engineering Cycles, Figure 10.11(b)

This is a refrigerator cycle, so it is intended to move heat from one location to another. In this case the heat is being ejected in order to liquefy the fluid flowing from source to sink.

To increase this cycle's efficiency, energy is ejected from the compressor. This takes place at CLR-1. Intercooling reduces the amount of work necessary to compress a gas, thus increasing cycle efficiency. To increase this cycle's efficiency, a turbine is used to cool the working fluid. This takes place at TUR-1. Using a turbine in place of a throttle to create the pressure drop in a refrigerator results in a greater decline in temperature, which is often useful in a cryogenic cycle.

Cycle 32
Helium Liquefaction and Refrigerating Plant with Series Operation of Expansion Turbines

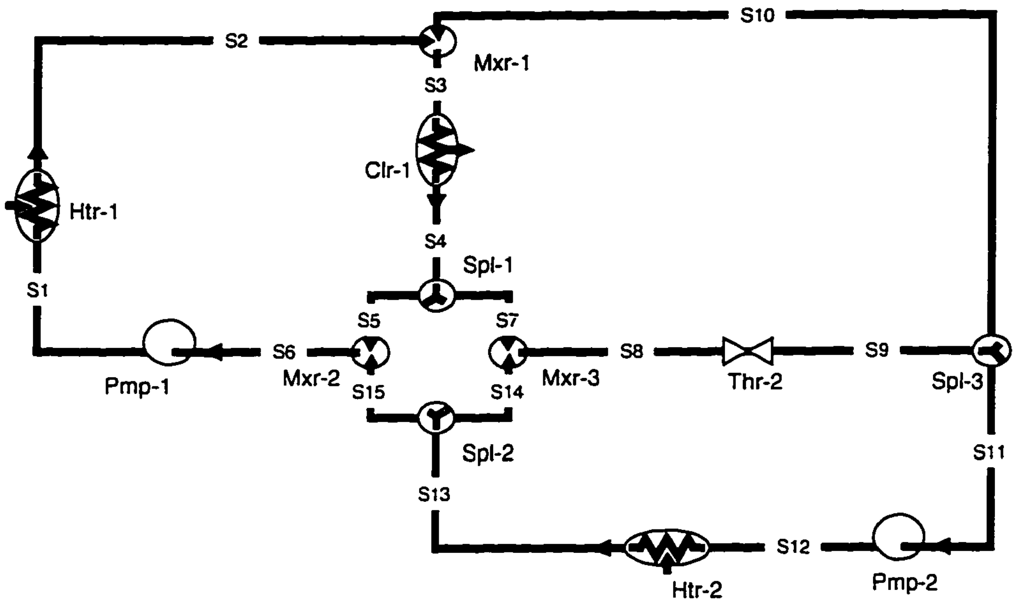


Source: Analysis of Engineering Cycles, Figure 10.13

This is a refrigerator cycle, so it is intended to move heat from one location to another. In this case the heat is being ejected in order to liquefy the fluid flowing from source to sink.

To increase this cycle's efficiency, there are 2 points (TUR-2 and TUR-1) where a turbine is used to cool the working fluid. Using a turbine in place of a throttle to create the pressure drop in a refrigerator results in a greater decline in temperature, which is often useful in a cryogenic cycle. There are also 2 points (CLR-1 and CLR-3) where energy is ejected from the compressor. Intercooling reduces the amount of work necessary to compress a gas, thus increasing cycle efficiency. To increase this cycle's efficiency, it is partitioned into multiple subcycles, one removing heat from the other. This takes place at SUBCYCLE-1/SUBCYCLE-1. Cascading cycles enables the use of working fluids with different thermal characteristics in the same system. In this case the energy-removing subcycle can use a refrigerant that operates at substantially different pressures in order to achieve greater heat-transfer.

Cycle 33
Jet Ejector Air Conditioning System

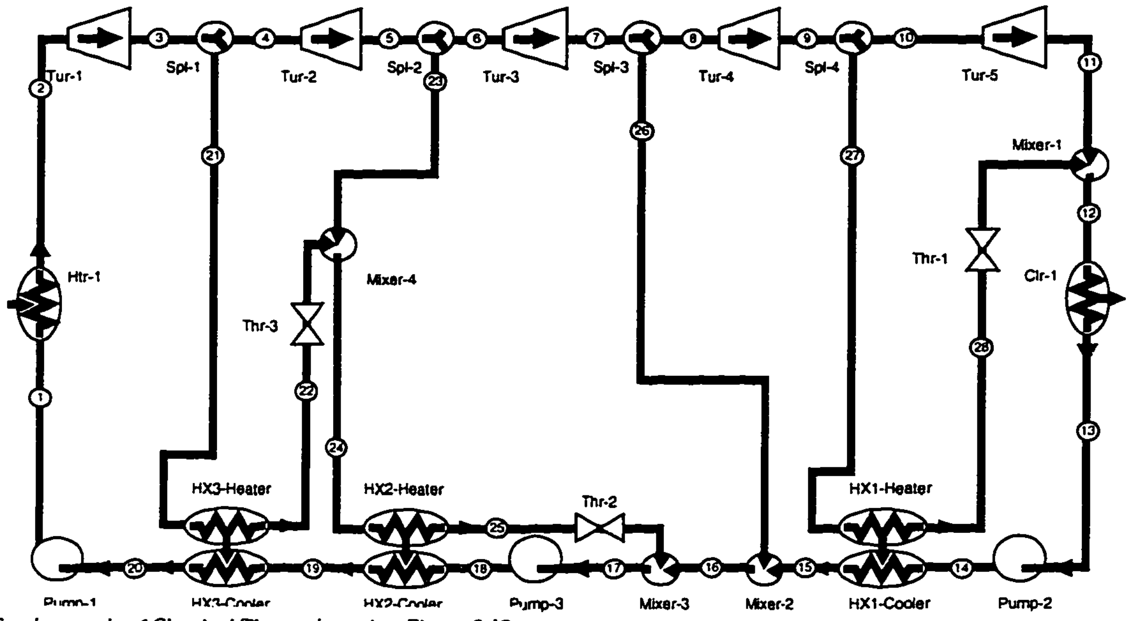


Fundamentals of Classical Thermodynamics, Figure P9.92

This is a refrigerator cycle, so it is intended to move heat from one location to another.

To preserve this cycle's integrity, a jet-ejector is used to compress the working fluid. This takes place at MXR1. Jet ejectors have no moving parts and hence vapor refrigeration systems that use them for compression are inexpensive and safe to operate, although they produce effective cooling to levels well above the freezing point of the working fluid.

Cycle 34
Regenerative Rankine Cycle with Explicit Throttles

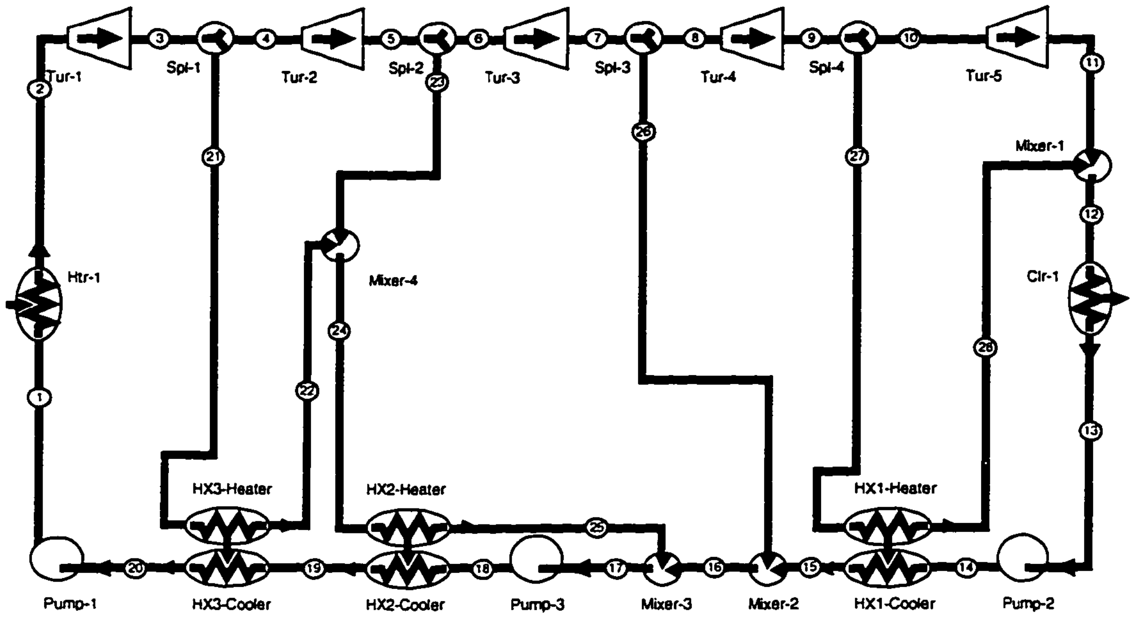


Fundamentals of Classical Thermodynamics, Figure 9.12

This cycle is a heat-engine, so it is intended to produce a change in the environment, namely work. This heat engine fully condenses its working fluid in order to realize gains in pumping efficiency over compressing gas or saturated mixtures.

To increase this cycle's efficiency, fluid bled from the turbine is directly mixed with the boiler feed liquid. This takes place at MXR2. Heat-exchange via mixing is more efficient than indirect transfer in closed heat-exchangers, and it also enables deaeration of working fluid, but it requires more pumps because inlets must be maintained at the same pressure. There are also 3 points (HX1H, HX3H, and HX2H) where working fluid bled from the turbine is used to supply heat in a heat-exchanger. Heat-exchange via closed heat-exchangers is less efficient than direct mixing, but the two working fluids can be at different pressures, reducing the need for pumps.

Cycle 35
Regenerative Rankine Cycle with Implicit Throttles

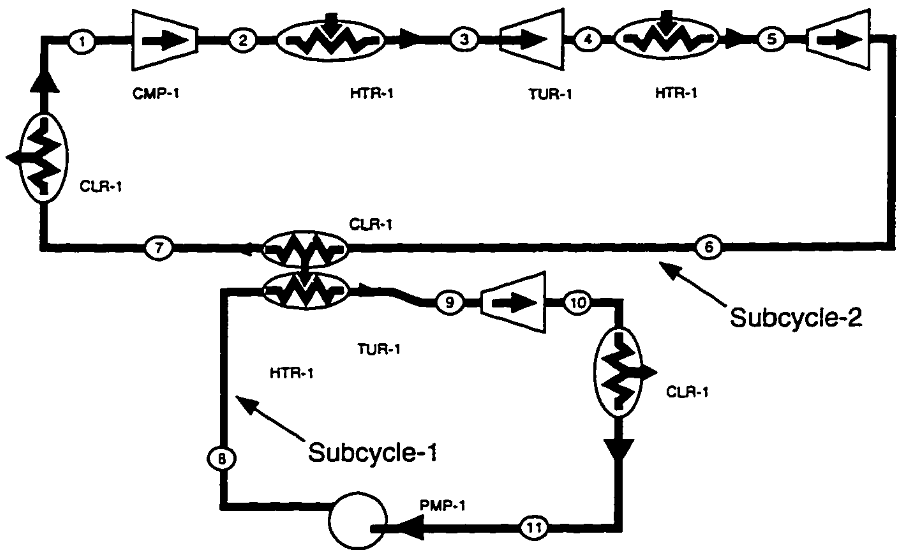


Fundamentals of Classical Thermodynamics, Figure 9.12

This cycle is a heat-engine, so it is intended to produce a change in the environment, namely work. This heat engine fully condenses its working fluid in order to realize gains in pumping efficiency over compressing gas or saturated mixtures.

To increase this cycle's efficiency, fluid bled from the turbine is directly mixed with the boiler feed liquid. This takes place at MXR2. Heat-exchange via mixing is more efficient than indirect transfer in closed heat-exchangers, and it also enables deaeration of working fluid, but it requires more pumps because inlets must be maintained at the same pressure. There are also 3 points (HX1H, HX3H, and HX2H) where working fluid bled from the turbine is used to supply heat in a heat-exchanger. Heat-exchange via closed heat-exchangers is less efficient than direct mixing, but the two working fluids can be at different pressures, reducing the need for pumps.

Cycle 36
Combined Gas-Turbine and Rankine Cycle



Basic Engineering Thermodynamics, Figure 15.11

This cycle is a heat-engine, so it is intended to produce a change in the environment, namely work.

To increase this cycle's efficiency, it is partitioned into two subcycles, one powering the other via its ejected heat. This takes place at SUBCYCLE-2/SUBCYCLE-1. Cascading cycles enables the use of working fluids with different thermal characteristics in the same system. In this case a high-temperature gas cycle's waste heat is used to power a vapor cycle.

Appendix C
CARNOT Knowledge Base

Evidential Tests

COOLER DEFINITIONS

```
(defTypeAbst (cooler ?cooler ?in ?out)
   heat-producer
   heat-changer
   device)

(defRoleAbst (cooler ?cooler ?in ?out)
  ((heat-ejector 0.6)
   (heat-provider 0.4))
  (-> heat-ejector
     (intercooler 0.25)
     (fluid-cooler 0.75)))
```

COOLER AS HEAT-EJECTOR

```
(defTest Clr-Tst1 heat-ejector 1000.0 (cooler ?clr ?c-in ?c-out)
  "A singleton cooler must be, by the second law, a heat-ejector"
  (singleton ?clr :cycle))

(defTest Clr-Tst2 heat-ejector 5.0 (cooler ?clr ?c-in ?c-out)
  "A cooler downstream of the last turbine and upstream of a pump is very
   likely to be the heat-ejector of the system"
  (turbine ?tur ?t-in ?t-out)
  ((downrange ?tur ?tur-range)
   :TEST (and (member ?clr ?tur-range)
              (notany #'rf::turbine? ?tur-range)))
  (downrange ?clr ?clr-range)
  ((genus ?pmp work-consumer)
   :TEST (and (member ?pmp ?tur-range)
              (member ?pmp ?clr-range))))

(defTest Clr-Tst3 (:not heat-ejector) 2.0 (genus ?clr heat-producer)
  "A cooler immediately upstream of a turbine is unlikely to be a heat-ejector"
  (cycle-type :heat-engine ?reason)
  (?clr has-outstf ?c-out)
  (turbine ?tur ?c-out ?tur-out))
```

COOLER AS INTERCOOLER

```
(defTest Clr-Tst4 intercooler 100.0 (genus ?clr heat-producer)
  "A cooler between two compressors is very likely to be an intercooler
   intended to make the work of the second compressor easier"
  (adjacent ?cmp1 ?clr ?stfs1)
  (compressor ?cmp1 ?c1-in ?c1-out)
  (adjacent ?clr ?cmp2 ?stfs2)
  ((compressor ?cmp2 ?c2-in ?c2-out)
   :TEST (not (eql ?cmp1 ?cmp2))))
```

COOLER AS HEAT-PROVIDER

```
(defTest Clr-Tst5 heat-provider 10.0 (cooler ?clr ?c-in ?c-out)
  "A cooler downstream of a bleed-valve is likely providing heat to another
   process outside the cycle"
  (cycle-type :heat-engine ?reason)
  (adjacent ?spl ?clr ?stfs)
  (role ?spl bleed-valve ?prob))

(defTest Clr-Tst6 heat-provider 7.0 (cooler ?clr ?c-in ?c-out)
  "A singleton cooler on a heat-mover cycle with a singleton heat-consumer
   acting as a heat-injector is very likely to be acting as a heat-provider,
   since the heat-consumer is injecting the heat for a purpose"
  (singleton ?clr ?subc)
  (subcycle ?subc)
  (role ?subc heat-mover ?bp)
  (role ?htr heat-injector ?prob)
  (singleton ?htr ?subc))

(defTest Clr-Tst7 heat-provider 20.0 (hx-cooler ?clr ?c-in ?c-out)
  "A singleton hx-cooler on a radiation-isolation subcycle is almost certainly
   providing heat to the heater half"
  (role ?subc radiation-isolator ?prob1)
  (singleton ?clr ?subc)
  (heat-path ?clr ?htr)
  (((devs ?subc) members ?subc-devs)
    :test (not (member ?htr ?subc-devs))))

(defTest Clr-Tst8 heat-provider 20.0 (hx-cooler ?hxc ?c-in ?c-out)
  "An hx-cooler with a simple cooler downstream on an energy-removing subcycle
   is very likely to be acting as a heat-provider, because the heat-ejection
   function could be entirely provided by the simple cooler"
  (role ?subc energy-remover ?bp)
  ((devs ?subc) has-member ?hxc)
  (cooler ?clr ?in ?out)
  (adjacent ?hxc ?clr ?link))
```

COOLER AS FLUID-COOLER

```
(defTest Clr-Tst9 fluid-cooler 35.0 (genus ?clr2 heat-producer)
  "A cooler immediately downstream of an intercooled set of compressors is
   probably acting as a fluid cooler"
  (compressor ?cmp1 ?c1-in ?c1-out)
  (adjacent ?cmp1 ?clr1 ?stfs1)
  (genus ?clr1 heat-producer)
  (adjacent ?clr1 ?cmp2 ?stfs2)
  (adjacent ?cmp2 ?clr2 ?stfs3)
  (compressor ?cmp2 ?c2-in ?c2-out)
  (?clr2 has-outstf ?clr2-out)
  ((?clr2-out has-sink ?dev)
   :test (not (rf::compressor? ?dev))))

(defTest Clr-Tst10 (:not fluid-cooler) 2.0 (genus ?clr heat-producer)
  "A cooler immediately upstream of a turbine is unlikely to be a
   fluid-cooler"
  (cycle-type :heat-engine ?reason)
  (?clr has-outstf ?c-out)
  (turbine ?tur ?c-out ?tur-out))
```

HEATER DEFINITIONS

```
(defTypeAbst (heater ?heater ?in ?out)
   heat-consumer
   heat-changer
   device)

(defRoleAbst (heater ?heater ?in ?out)
  ((heat-absorber 0.5) (heat-injector 0.5))
  (-> heat-injector
    (fluid-heater 0.2)
    (preheater 0.7)
    (reheater 0.1)))
```

HEATER AS HEAT-ABSORBER

```
(defTest Htr-Tst1 heat-absorber 5.0 (genus ?heater heat-consumer)
  "A singleton heater on an energy-removing subcycle is often absorbing
   thermal energy, by the second law"
  (singleton ?heater ?subcycle)
  ((role ?subcycle ?role ?prob)
   :TEST (member ?role '(heat-mover energy-remover) :test #'eq)))

(defTest Htr-Tst2 heat-absorber 12.0 (heater ?htr ?thr-out ?htr-out)
  "A heater in a refrigerating cycle with a throttle immediately upstream is
   virtually always a heat-absorber"
  (cycle-type :refrigerator ?reason)
  (throttle ?thr ?thr-in ?thr-out))

(defTest Htr-Tst3 heat-absorber 2.0 (genus ?htr heat-consumer)
  "A heater in a refrigerating cycle with an upstream throttle is
   probably a heat-absorber"
  (cycle-type :refrigerator ?reason)
  (throttle ?thr ?thr-in ?thr-out)
  (adjacent ?thr ?htr ?stfs))

(defTest Htr-Tst4 heat-absorber 1.5 (genus ?pre heat-consumer)
  "Refrigeration cycles are more likely to use heaters as heat-absorbers than
   as energy-injectors"
  (cycle-type :refrigerator ?reason))
```

HEATER AS HEAT-INJECTOR

```
(defTest Htr-Tst5 heat-injector 200.0 (genus ?heater heat-consumer)
  "A heater upstream of a mixer functioning as a jet ejector and downstream of
   a pump is probably providing that mixer with thermal energy"
  (role ?mixer jet-ejector ?prob)
  (?heater has-outstf ?out)
  ((mixer ?mixer ?in1 ?in2 ?out2)
   :TEST (or (eql ?out ?in1) (eql ?out ?in2)))
  (dev ?pump pump)
  ((downrange ?pump ?pump-range)
   :TEST (member ?heater ?pump-range)))

(defTest Htr-Tst6 heat-injector 20.0 (genus ?heater heat-consumer)
  "A singleton heater on a work-producing subcycle must be injecting
   thermal energy, by the second law"
  (singleton ?heater ?subcycle)
  (role ?subcycle work-generator ?prob))

(defTest Htr-Tst7 heat-injector 100.0 (genus ?heater heat-consumer)
  "A singleton heater must be injecting thermal energy, by the second law"
  (cycle-type :heat-engine ?reason)
  (singleton ?heater :CYCLE))
```

```
(defTest Htr-Tst8 heat-injector 5.0 (genus ?htr heat-consumer)
  "A heater in a heat-engine cycle with a pump upstream and a turbine
   downstream is probably a heat-injector"
  (cycle-type :heat-engine ?reason)
  (turbine ?tur ?t-in ?t-out)
  (adjacent ?htr ?tur ?stfs)
  ((dev ?pmp ?type)
   :TEST (member ?type '(compressor pump)))
  ((downrange ?pmp ?pmp-range)
   :TEST (and (notany 'pump? ?pmp-range)
              (member ?htr ?pmp-range)
              (member ?tur ?pmp-range))))

(defTest Htr-Tst9 heat-injector 1.5 (genus ?pre heat-consumer)
  "Heat-engines are more likely to use heaters as heat-injectors than as
   heat-absorbers"
  (cycle-type :heat-engine ?reason))
```

HEATER AS FLUID-HEATER

```
(defTest Htr-Tst10 fluid-heater 10.0 (genus ?heater heat-consumer)
  "A singleton heater on a work-producing subcycle must be a fluid-heater"
  (singleton ?heater ?subcycle)
  (role ?subcycle work-generator ?prob))

(defTest Htr-Tst11 fluid-heater 10.0 (genus ?heater heat-consumer)
  "A singleton heater on a work-producing subcycle must be a fluid-heater"
  (cycle-type :heat-engine ?reason)
  (singleton ?heater :CYCLE))

(defTest Htr-Tst12 fluid-heater 8.0 (genus ?heater heat-consumer)
  "A heater immediately upstream of a turbine is likely to be a fluid-heater"
  (?heater has-instf ?in)
  ((?in has-source ?dev)
   :test (not (rf::turbine? ?dev)))
  (?heater has-outstf ?out)
  (turbine ?tur ?out ?tur-out))

(defTest Htr-Tst13 fluid-heater 9.0 (genus ?heater heat-consumer)
  "A pump adjacent to a heater adjacent to a turbine typically means that the
   heater is a fluid-heater"
  (adjacent ?pmp ?heater ?link1)
  (genus ?pmp work-consumer)
  (adjacent ?heater ?tur ?link2)
  (genus ?tur work-producer))

(defTest Htr-Tst14 fluid-heater 12.0 (hx-heater ?htr ?h-in ?h-out)
  "An hx-heater with a heat-path to an hx-cooler on a radiation-isolating
   subcycle is very likely to be the primary fluid-heater for the cycle"
  (adjacent ?htr ?tur ?connection)
  (dev ?tur turbine)
  (role ?subc work-generator ?prob1)
  (((devs ?work-subc) members ?work-subc-devs)
   :test (member ?htr ?work-subc-devs))
  (heat-path ?clr ?htr)
  (role ?rad-subc radiation-isolator ?prob2)
  (((devs ?rad-subc) members ?rad-subc-devs)
   :test (member ?clr ?rad-subc-devs)))
```

HEATER AS PREHEATER

```
(defTest Htr-Tst15 (:not preheater) 5 (genus ?heater heat-consumer)
  "A refrigerator rarely has need to preheat its working fluid"
  (cycle-type :refrigerator ?reason))

(defTest Htr-Tst16 (:not preheater) 25 (genus ?heater heat-consumer)
  "A heater with no heaters downrange of it is unlikely to be a preheater"
  ((downrange ?heater ?range)
   :test (notany #'(lambda (d)
                     (or (rf::heat-consumer? d)
                         (rf::mxr-heater? d)))
          ?range)))

(defTest Htr-Tst17 preheater 25 (genus ?heater heat-consumer)
  "A heater on a primary fluid loop with a mxr-heater downrange of it
   is likely to be a preheater"
  (primary-fluid-loop-for ?subcycle ?floop)
  ((floop ?floop ?stfs ?devs)
   :test (member ?heater ?devs))
  (downrange ?heater ?range)
  ((role ?mxr mxr-heater ?bp)
   :test (member ?mxr ?range)))

(defTest Htr-Tst18 (:not preheater) 5 (genus ?heater heat-consumer)
  "A heater on a non-work-generating subcycle of a refrigerator is unlikely
   to be preheating the working fluid"
  (cycle-type :refrigerator ?reason)
  (((devs ?subcycle) members ?subc-devs)
   :TEST (member ?heater ?subc-devs))
  ((role ?subcycle ?subc-role ?prob)
   :TEST (not (eql ?subc-role 'work-generator))))

(defTest Htr-Tst19 (:not preheater) 25.0 (genus ?heater heat-consumer)
  "A heater adjacent to a turbine is not a preheater"
  (adjacent ?heater ?tur ?connection)
  (dev ?tur turbine))

(defTest Htr-Tst20 preheater 5.0 (genus ?pre heat-consumer)
  "A heater upstream of the heater feeding the turbines is very likely a
   preheater"
  (cycle-type :heat-engine ?reason)
  (downrange ?pre ?pre-range)
  ((role ?main fluid-heater ?prob1)
   :test (member ?main ?pre-range))
  (adjacent ?main ?tur ?connection)
  ((role ?tur work-source ?prob2)
   :test (member ?tur ?pre-range)))

(defTest Htr-Tst21 preheater 10.0 (genus ?pre heat-consumer)
  "A heater upstream of a steam-drum feeding turbines is very likely a
   preheater"
  (cycle-type :heat-engine ?reason)
  (downrange ?pre ?pre-range)
  ((role ?mxr steam-drum-half ?prob1)
   :TEST (member ?mxr ?pre-range))
  (mixer ?mxr ?in1 ?in2 ?link)
  (splitter ?spl ?link ?out1 ?out2)
  (role ?spl steam-drum-half ?prob2)
  (downrange ?spl ?spl-range)
  ((dev ?tur turbine)
   :TEST (and (member ?tur ?pre-range) (member ?tur ?spl-range))))
```

```
(defTest Htr-Tst22 preheater 7.5 (genus ?pre heat-consumer)
  "A heater immediately followed by a heat-exchanging cooler on a topping
   subcycle is probably a single combustion chamber in which the flue gasses
   are used to power the topping turbine.  An upstream heater is most likely
   an air preheater for the combustion chamber"
  (((devs ?subc) members ?devs)
   :TEST (member ?pre ?devs))
  (role ?subc topping ?prob)
  (?pre has-outstf ?pre-out)
  (heater ?htr ?pre-out ?htr-out)
  (hx-cooler ?hxc ?htr-out ?hxc-out)
  (turbine ?tur ?hxc-out ?tur-out))

(defTest Htr-Tst23 (:not preheater) 10.0 (genus ?htr heat-consumer)
  "A heater in the downrange of a turbine cannot be a preheater"
  (cycle-type :heat-engine ?reason)
  (role ?tur work-source ?bp1)
  ((downrange ?tur ?downrange)
   :test (member ?htr ?downrange)))
```

HEATER AS REHEATER

```
(defTest Htr-Tst24 reheater 40.0 (genus ?htr heat-consumer)
  "A heater with a turbine upstream and another turbine downstream is very
   likely to be reheating the working-fluid"
  (cycle-type :heat-engine ?reason)
  (turbine ?tur1 ?t1-in ?t1-out)
  ((downrange ?tur1 ?tur1-range)
   :TEST (member ?htr ?tur1-range))
  (downrange ?htr ?htr-range)
  ((turbine ?tur2 ?t2-in ?t2-out)
   :TEST (and (member ?tur2 ?tur1-range)
              (eq ?tur2 (find-if #'rf::turbine? ?htr-range)))))

(defTest Htr-Tst25 (:not reheater) 5.0 (genus ?heater heat-consumer)
  "A heater on a non-work-generating subcycle of a refrigerator is unlikely
   to be reheating the working fluid"
  (cycle-type :refrigerator ?reason)
  (((devs ?subcycle) members ?subc-devs)
   :TEST (member ?heater ?subc-devs))
  ((role ?subcycle ?subc-role ?prob)
   :TEST (not (eql ?subc-role 'work-generator))))

(defTest Htr-Tst26 (:not reheater) 25.0 (genus ?heater heat-consumer)
  "A heater with no turbines downrange of it is unlikely to be a reheater"
  ((downrange ?heater ?range)
   :test (notany #'rf::turbo-device? ?range)))

(defTest Htr-Tst27 reheater 80.0 (genus ?heater heat-consumer)
  "A heater supplying a mxr-heater that is between two turbines is acting as a
   reheater"
  (role ?mxr mxr-heater ?bp-mxr1)
  (adjacent ?heater ?mxr ?connection)
  (cycle-type :heat-engine ?reason)
  (role ?tur1 work-source ?bp-tur1)
  (turbine ?tur1 ?t1-in ?t1-out)
  (effect heating ?mxr ?t1-out ?hot-out)
  (turbine ?tur2 ?hot-out ?t2-out)
  (role ?tur2 work-source ?bp-tur2))
```

HEAT-EXCHANGER DEFINITIONS

```
(defTypeAbst (hx-heater ?hx-h ?h-in ?h-out)
   heat-consumer
   heat-changer
   device)

(defTypeAbst (hx-cooler ?hx-c ?c-in ?c-out)
   heat-producer
   heat-changer
   device)

(defRoleAbst (hx-heater ?hx-h ?h-in ?h-out)
  ((heat-absorber 0.5) (heat-injector 0.5))
  (-> heat-injector
    (fluid-heater 0.2)
    (preheater 0.7)
    (reheater 0.1)))

(defRoleAbst (hx-cooler ?hx-c ?c-in ?c-out)
  ((heat-ejector 0.6)
   (heat-provider 0.4))
  (-> heat-ejector
    (intercooler 0.25)
    (fluid-cooler 0.75)))
```

HEAT-EXCHANGER EVIDENCE

```
(defTest Htx-Tst1 fluid-cooler 12.5 (hx-cooler ?hxc ?c-in ?c-out)
  "On a subcycle with no coolers ejecting heat to the environment, the cooling
   side of a heat-exchanger whose heating side is on another subcycle is almost
   certainly cooling the working fluid"
  (role ?subc work-generator ?prob)
  (((devs ?subc) members ?devs)
   :TEST (and (member ?hxc ?devs) (notany #'rf::cooler? ?devs)))
  ((heat-path ?hxc ?hxh)
   :TEST (not (member ?hxh ?devs))))

(defTest Htx-Tst2 heat-absorber 12.5 (hx-heater ?hxh ?h-in ?h-out)
  "The heating side of a heat-exchanger whose cooling side is on
   work-generating subcycle that lacks a cooling path to the environment is
   probably absorbing heat from that cycle"
  (role ?subc work-generator ?prob)
  (((devs ?subc) members ?devs)
   :TEST (and (not (member ?hxh ?devs)) (notany #'sadi::cooler? ?devs)))
  ((heat-path ?hxc ?hxh)
   :TEST (member ?hxc ?devs)))
```

MIXER DEFINITIONS

```
(defTypeAbst (MIXER ?mixer ?in1 ?in2 ?out)
   junction
   flow-changer
   device)

(defRoleAbst (MIXER ?mixer ?in1 ?in2 ?out)
  ((flow-join 0.60)
   (mxr-heater 0.25)
   (mxr-cooler 0.10)
   (jet-ejector 0.05)))
```

MIXER AS FLOW-JOIN

```
(defTest Mxr-Tst1 (:not flow-join) 5.0 (mixer ?mxr ?in1 ?in2 ?out)
  "Mixers terminating bleed-paths are not often flow-joins"
  ((bleed-path ?spl ?devs)
   :TEST (eql (car (last ?devs)) ?mxr)))

(defTest Mxr-Tst2 flow-join 8.0 (mixer ?mxr ?m-in1 ?m-in2 ?m-out)
  "A mixer at the end of a bleed path containing a heat-provider is probably
   rejoining the flow to the cycle"
  (role ?clr heat-provider ?prob)
  ((bleed-path ?spl ?devs)
   :TEST (and (eql (car (last ?devs)) ?mxr)
              (member ?clr ?devs))))

(defTest Mxr-Tst3 flow-join 8.0 (mixer ?mxr ?m-in1 ?m-in2 ?m-out)
  "A mixer at the end of a bleed path containing a work-producing
   turbine is probably rejoining the flow to the cycle"
  ((bleed-path ?spl ?devs)
   :TEST (eql (car (last ?devs)) ?mxr))
  ((role ?tur work-source ?prob)
   :TEST (member ?tur ?devs)))

(defTest Mxr-Tst4 flow-join 8.0 (mixer ?mxr ?m-in1 ?m-in2 ?m-out)
  "A mixer at the end of a bleed path starting at a flash-chamber is likely to
   be just a flow-join"
  (role ?spl flash-chamber ?prob)
  ((bleed-path ?spl ?devs)
   :TEST (eql (car (last ?devs)) ?mxr)))
```

MIXER AS JET-EJECTOR

```
(defTest Mxr-Tst5 (:not jet-ejector) 10.0 (mixer ?mixer ?in1 ?in2 ?out)
  "Heat engines rarely use jet-ejectors"
  (cycle-type :heat-engine ?reason))

(defTest Mxr-Tst6 jet-ejector 2.0 (mixer ?mixer ?in1 ?in2 ?out)
  "Refrigerators occasionally make use of jet-ejectors"
  (cycle-type :refrigerator ?reason))

(defTest Mxr-Tst7 jet-ejector 3.0 (mixer ?mixer ?in1 ?in2 ?out)
  "Jet ejectors require an upstream heater to power them"
  ((heater ?htr ?htr-in ?htr-out)
   :TEST (or (eql ?htr-out ?in1) (eql ?htr-out ?in2))))

(defTest Mxr-Tst8 jet-ejector 35.0 (mixer ?mixer ?in1 ?in2 ?out)
  "Heating then cooling without an intermediate process would violate the
   rational designer heuristic"
  ((heater ?htr ?htr-in ?htr-out)
   :TEST (or (eql ?htr-out ?in1) (eql ?htr-out ?in2)))
  (cooler ?clr ?out ?clr-out))

(defTest Mxr-Tst9 (:not jet-ejector) 10.0 (mixer ?mixer ?in1 ?in2 ?out)
  "A mixer lacking an upstream energy source is unlikely to be a jet-ejector"
  (?mixer has-indev ?indev1)
  ((?mixer has-indev ?indev2)
   :TEST (string< ?indev1 ?indev2))
  ((dev ?indev1 ?type1)
   :TEST (not (eq ?type1 'heater)))
  ((dev ?indev2 ?type2)
   :TEST (not (eq ?type2 'heater))))
```

```
(defTest Mxr-Tst10 jet-ejector 40.0 (mixer ?mixer ?in1 ?in2 ?out)
  "A mixer participating in two fluid loops, one requiring a pressure decreaser
   and the other requiring a pressure increaser, is very likely a jet-ejector"
  (requires ?floop1 pressure-decreaser)
  ((floop ?floop1 ?stfs1 ?devs1)
   :TEST (member ?mixer ?devs1))
  (requires ?floop2 pressure-increaser)
  ((floop ?floop2 ?stfs2 ?devs2)
   :TEST (member ?mixer ?devs2)))
```

MIXER AS MXR-HEATER

```
(defTest Mxr-Tst11 mxr-heater 50.0 (mixer ?mxr ?m-in1 ?m-in2 ?m-out)
  "A mixer that is part of a multi-port chamber in a heat-engine is very likely
   acting as a heater"
  (cycle-type :heat-engine ?reason)
  (aggr ?mpc mp-chamber)
  ((?mpc has-devs ?devs)
   :test (member ?mxr ?devs)))

(defTest Mxr-Tst12 mxr-heater 20.0 (mixer ?mxr ?m-in1 ?m-in2 ?m-out)
  "A mixer with a temperature differential across its inlets in a heat-engine
   is very likely to be an open heat-exchanger"
  (cycle-type :heat-engine ?reason)
  ((structural (?ineq (t ?in1) (t ?in2)))
   :TEST (or (and (eq ?in1 ?m-in1)
                  (eq ?in2 ?m-in2))
             (and (eq ?in2 ?m-in1)
                  (eq ?in1 ?m-in2)))))

(defTest Mxr-Tst13 mxr-heater 3.5 (mixer ?mxr ?m-in1 ?m-in2 ?m-out)
  "A phase-change in the working fluid across a mixer is a strong indication
   that the mixer is acting as an open heat-exchanger"
  (cycle-type :heat-engine ?reason)
  (phase ?m-out gas)
  (phase ?m-in1 ?phase1)
  ((phase ?m-in2 ?phase2)
   :TEST (or (and (eql ?phase1 'gas)
                  (eql ?phase2 'liquid))
             (and (eql ?phase2 'gas)
                  (eql ?phase1 'liquid)))))

(defTest Mxr-Tst14 mxr-heater 8.0 (mixer ?mxr ?m-in1 ?m-in2 ?m-out)
  "A mixer receiving fluid directly from a splitter where both mixer and
   splitter are on the same fluid loop and the splitter is part of a
   turbine series is very likely to be acting as a heater"
  (cycle-type :heat-engine ?reason)
  (role ?spl bleed-valve ?prob)
  (bleed-path ?spl (?mxr)))

(defTest Mxr-Tst15 mxr-heater 35.0 (mixer ?mxr ?m-in1 ?m-in2 ?m-out)
  "A mixer at the end of a bleed path that has no heat-producers, throttles,
   turbines, or pumps on it is probably acting as a heater"
  (cycle-type :heat-engine ?reason)
  ((bleed-path ?spl ?devs)
   :TEST (and (eql (car (last ?devs)) ?mxr)
              (notany #'(lambda (dev)
                          (or (rf::heat-producer? dev)
                              (rf::throttle? dev)
                              (rf::pump? dev)
                              (rf::turbine? dev)))
                (butlast ?devs)))))
```

```
(defTest Mxr-Tst16 mxr-heater 6.0 (mixer ?mxr ?m-in1 ?m-in2 ?m-out)
  "A mixer directly connected to a splitter downstream of a turbine and
   upstream of a cooler and a pump is likely to be an open heat-exchanger,
   for deareating"
  (cycle-type :heat-engine ?reason)
  (cooler ?clr ?c-in ?c-out)
  ((pump ?pmp ?c-out ?p-out)
   :test (or (eql ?p-out ?m-in1) (eql ?p-out ?m-in1)))
  ((splitter ?spl ?s-in ?s-out1 ?s-out2)
   :test (or (eql ?s-out1 ?m-in1) (eql ?s-out2 ?m-in1)
             (eql ?s-out1 ?m-in2) (eql ?s-out2 ?m-in2)))
  (turbine ?tur ?tur-in ?tur-out)
  ((downrange ?tur ?tur-range)
   :test (and (member ?spl ?tur-range)
              (member ?clr ?tur-range)
              (member ?pmp ?tur-range))))
```

MIXER AS MXR-COOLER

```
(defTest Mxr-Tst17 mxr-cooler 35.0 (mixer ?mixer ?in1 ?in2 ?out)
  "A mixer immediately following a fluid-cooling turbine is likely acting as
   a cooler"
  (cycle-type :refrigerator ?reason)
  (role ?tur fluid-cooler ?prob)
  ((turbine ?tur ?t-in ?t-out)
   :TEST (or (eql ?t-out ?in1) (eql ?t-out ?in2))))

(defTest Mxr-Tst18 mxr-cooler 50.0 (mixer ?mxr ?m-in1 ?m-in2 ?m-out)
  "A mixer that is part of a multi-port chamber in a refrigerator is very
   likely acting as a heater"
  (cycle-type :refrigerator ?reason)
  (aggr ?mpc mp-chamber)
  ((?mpc has-devs ?devs)
   :test (member ?mxr ?devs)))

(defTest Mxr-Tst19 mxr-cooler 70.0 (mixer ?mxr ?m-in1 ?m-in2 ?m-out)
  "A mixer in between two compressors is likely acting as an intercooler"
  ((compressor ?cmp1 ?c-in1 ?c-out1)
   :TEST (or (eql ?c-out1 ?m-in1) (eql ?c-out1 ?m-in2)))
  (compressor ?cmp2 ?m-out ?c-out2))
```

PUMP

```
(defTypeAbst (PUMP ?pump ?in ?out)
   work-consumer
   pressure-changer
   device)

(defRoleAbst (PUMP ?pump ?in ?out)
  ((flow-producer 0.6)
   (flash-preventer 0.4)))
```

PUMP AS FLOW-PRODUCER

```
(defTest Pmp-Tst1 flow-producer 2.6 (pump ?pmp1 ?in ?out)
  "The pump immediately upstream of the last heater prior to a turbine is
   almost certainly intended to maintain a flow of working fluid"
  (genus ?heater heat-consumer)
  (?heater has-instf ?out)
  (?heater has-outstf ?h-out)
  (turbine ?tur ?h-out ?t-out))
```

```
(defTest Pmp-Tst2 flow-producer 4.0 (pump ?pmp1 ?in ?out)
  "A pump in a radiation-isolating subcycle is generally intended to induce
   fluid flow"
  (role ?subc radiation-isolator ?prob)
  (((devs ?subc) members ?subc-devs)
   :TEST (member ?pmp1 ?subc-devs)))
```

PUMP AS FLASH-PREVENTER

```
(defTest Pmp-Tst3 (:not flash-preventer) 2.0 (pump ?pmp1 ?in ?out)
  "Premature flashing is generally not a concern in refrigerators"
  (cycle-type :refrigerator ?reason))

(defTest Pmp-Tst4 (:not flash-preventer) 4.0 (pump ?pmp1 ?in ?out)
  "Pumps without other pumps or turbines downstream of them are probably not
   intended to prevent the working fluid from flashing"
  (cycle-type :heat-engine ?reason)
  ((downrange ?pmp1 ?pmp-range)
   :TEST (and (notany #'pump? ?pmp-range)
              (some #'rf::turbine? ?pmp-range))))

(defTest Pmp-Tst5 flash-preventer 6.0 (pump ?pmp1 ?in ?out)
  "A pump feeding a heater with another pump downstream of both is very likely
   intended to prevent the working fluid from flashing in the heater"
  (cycle-type :heat-engine ?reason)
  (primary-floop-for ?subc ?floop)
  ((floop ?floop ?stfs ?devs)
   :test (let ((pmp-posn (position ?pmp1 ?devs)))
           (and pmp-posn
                (rf::safe-less-than
                 (position-if #'rf::heat-consumer? ?devs :start (1+ pmp-posn))
                 (position-if #'rf::pump? ?devs :start (1+ pmp-posn))
                 (position-if #'rf::turbine? ?devs :start (1+ pmp-posn)))))))

(defTest Pmp-Tst6 flash-preventer 6.0 (pump ?pmp1 ?in ?out)
  "A pump feeding an open heat-exchanger with another pump downstream of both
   is probably intended to prevent the heat-exchange from vaporizing the
   working fluid"
  (cycle-type :heat-engine ?reason)
  (downrange ?pmp1 ?pmp-range)
  ((role ?mxr mxr-heater ?prob)
   :TEST (rf::safe-less-than
          (position-if #'rf::mixer? ?pmp-range)
          (position-if #'rf::pump? ?pmp-range)))
  (downrange ?mxr ?mxr-range)
  ((pump ?pmp2 ?p2-in ?p2-out)
   :TEST (and (member ?pmp2 ?pmp-range)
              (member ?pmp2 ?mxr-range))))

(defTest Pmp-Tst7 (:not flash-preventer) 10.0 (pump ?pmp1 ?in ?out)
  "A pump feeding an open heat-exchanger that is part of a steam-drum is not
   likely to be preventing flashing, because the steam drum will separate the
   saturated mixture into liquid and gas"
  (cycle-type :heat-engine ?reason)
  (downrange ?pmp1 ?pmp-range)
  ((role ?mxr mxr-heater ?prob)
   :TEST (rf::safe-less-than
          (position-if #'rf::mixer? ?pmp-range)
          (position-if #'rf::pump? ?pmp-range)))
  (recirculating-path ?spl ?mxr ?path)
  ((pump ?pmp2 ?p2-in ?p2-out)
   :TEST (and (member ?pmp2 ?pmp-range)
              (member ?pmp2 ?path))))
```

SPLITTER DEFINITIONS

```
(defTypeAbst (SPLITTER ?splitter ?in ?out1 ?out2)
   junction
   flow-changer
   device)

(defRoleAbst (SPLITTER ?splitter ?in ?out1 ?out2)
  ((flow-fork 0.45)
   (bleed-valve 0.35)
   (flash-chamber 0.20)))
```

SPLITTER AS FLOW-FORK

```
(defTest Spl-Tst1 flow-fork 8.0 (splitter ?spl ?in ?out1 ?out2)
  "A splitter connected to two devices of the same type is probably a
   flow-fork"
  (?out1 has-sink ?dev1)
  (?out2 has-sink ?dev2)
  (dev ?dev1 ?dtype)
  (dev ?dev2 ?dtype))
```

SPLITTER AS BLEED-VALVE

```
(defTest Spl-Tst2 (:not bleed-valve) 1.7 (splitter ?spl ?link1 ?link2 ?bleed)
  "Refrigerators rarely, if ever, make use of bleed-valves"
  (cycle-type :refrigerator ?reason))

(defTest Spl-Tst3 bleed-valve 30.0 (splitter ?spl ?link1 ?link2 ?bleed)
  "A splitter with a turbine upstream, another downstream, and an hx-cooler
   immediately downstream of its other outlet is probably a bleed-valve"
  (turbine ?tur1 ?in ?link1)
  (turbine ?tur2 ?link2 ?out)
  (?bleed has-sink ?dev)
  (hx-cooler ?dev ?hxc-in ?hxc-out))

(defTest Spl-Tst4 bleed-valve 30.0 (splitter ?spl ?link1 ?link2 ?bleed)
  "A splitter with a turbine upstream, another downstream, and a mixer
   immediately downstream of its other outlet is probably a bleed-valve"
  (turbine ?tur1 ?in ?link1)
  (turbine ?tur2 ?link2 ?out)
  (?bleed has-sink ?dev)
  (mixer ?dev ?mxr-in1 ?mxr-in2 ?mxr-out))

(defTest Spl-Tst5 bleed-valve 8.0 (splitter ?spl ?in ?out1 ?out2)
  "A splitter between two turbines is generally acting as a bleed-valve"
  (cycle-type :heat-engine ?reason)
  (turbine ?tur1 ?tur1-in ?in)
  ((turbine ?tur2 ?tur2-in ?out)
   :TEST (or (eql ?tur2-in ?out1) (eql ?tur2-in ?out2))))
```

```
(defTest Spl-Tst6 bleed-valve 6.0 (splitter ?spl ?in ?out1 ?out2)
  "A splitter just prior to a turbine is often a bleed-valve"
  (cycle-type :heat-engine ?reason)
  ((turbine ?tur1 ?tur1-in ?out)
   :TEST (or (eql ?tur1-in ?out1) (eql ?tur1-in ?out2)))
  ((?in has-source ?indev)
   :TEST (and (not (rf::mixer? ?indev))
              (not (rf::turbine? ?indev)))))

(defTest Spl-Tst7 bleed-valve 6.0 (splitter ?spl ?s-in ?s-out1 ?s-out2)
  "A splitter just after a turbine is often a bleed-valve"
  (cycle-type :heat-engine ?reason)
  (turbine ?tur ?t-in ?s-in)
  ((?s-out1 has-sink ?dev1)
   :test (not (rf::turbine? ?dev1)))
  ((?s-out2 has-sink ?dev2)
   :test (not (rf::turbine? ?dev2))))
```

SPLITTER AS FLASH-CHAMBER

```
(defTest Spl-Tst8 flash-chamber 50.0 (splitter ?spl ?s-in ?out1 ?out2)
  "A splitter that is part of a multi-port chamber is very likely acting as a
   flash-chamber"
  (aggr ?mpc mp-chamber)
  ((?mpc has-devs ?devs)
   :test (member ?spl ?devs)))

(defTest Spl-Tst9 flash-chamber 17.5 (splitter ?spl ?s-in ?out1 ?out2)
  "A splitter immediately downstream of a throttle in a refrigerating cycle is
   very likely to be a flash chamber for the working fluid just saturated"
  (cycle-type :refrigerator ?reason)
  (throttle ?thr ?t-in ?s-in))

(defTest Spl-Tst10 (:not flash-chamber) 20.0 (splitter ?spl ?in ?out1 ?out2)
  "A splitter coupled to a heat-absorber is unlikely to be a flash-chamber"
  (role ?htr heat-absorber ?bp)
  ((heater ?htr ?htr-in ?htr-out)
   :TEST (or (eql ?htr-in ?out1) (eql ?htr-in ?out2))))

(defTest Spl-Tst11 flash-chamber 80.0 (splitter ?spl ?s-in ?s-out1 ?s-out2)
  "A splitter in a series of turbines going directly to a throttle is likely
   to be acting as a flash-chamber to improve the quality of the steam"
  (cycle-type :heat-engine ?reason)
  (turbine ?tur1 ?t1-in ?s-in)
  ((throttle ?thr ?thr-in ?thr-out)
   :TEST (or (eql ?thr-in ?s-out1) (eql ?thr-in ?s-out2)))
  (downrange ?tur1 ?tur1-range)
  ((turbine ?tur2 ?t2-in ?t2-out)
   :TEST (member ?tur2 ?tur1-range)))

(defTest Spl-Tst12 flash-chamber 12.0 (splitter ?spl ?s-in ?s-out1 ?s-out2)
  "A splitter in a turbine series flowing into a reheater is likely to be
   acting as a flash-chamber to dry out the vapor to be reheated and increase
   the efficiency of the reheating"
  (cycle-type :heat-engine ?reason)
  (role ?tur1 work-source ?bp-tur1)
  (adjacent ?tur1 ?spl ?link1)
  (role ?rht reheater ?bp-rht)
  (adjacent ?spl ?rht ?link2)
  (role ?tur2 work-source ?bp-tur2)
  (adjacent ?rht ?tur2 ?link3))
```

```
(defTest Spl-Tstl3 flash-chamber 12.0 (splitter ?spl ?s-in ?s-outl ?s-out2)
  "A splitter on a bleed-path that contains a heater and flows back into the
   turbine is likely to be flashing the working fluid in order to provide the
   heater with vapor to superheat"
  (cycle-type :heat-engine ?reason)
  ((bleed-path ?start ?members)
   :test (and (not (eq ?start ?spl))
              (member ?spl ?members)
              (some #'rf::heat-consumer? ?members)))
  ((role ?mxr mxr-heater ?bp)
   :test (eq ?mxr (car (last ?members))))
  (adjacent ?mxr ?tur ?stfs)
  (turbine ?tur ?tur-in ?tur-out))

(defTest Spl-Tstl4 (:not flash-chamber) 40.0
                   (splitter ?splitter ?link ?outl ?out2)
  "A flash-chamber is unlikely to have a heater immediately upstream"
  (heater ?heater ?htr-in ?link))

(defTest Spl-Tstl5 (:not flash-chamber) 40.0
                   (splitter ?splitter ?link ?outl ?out2)
  "A flash-chamber is unlikely to have a cooler immediately upstream"
  (cooler ?cooler ?clr-in ?link))

(defTest Spl-Tstl6 flash-chamber 40.0 (splitter ?splitter ?in ?outl ?out2)
  "A splitter between the cooler halves of two hxs that are providing
   heat to their heater halves may be acting as a flash-chamber to supply
   steam to the downstream heat-provider"
  (hx-cooler ?hpl ?hpl-in ?in)
  (role ?hpl heat-provider ?probl)
  (role ?hp2 heat-provider ?prob2)
  ((hx-cooler ?hp2 ?hp2-in ?hp2-out)
   :test (or (eql ?hp2-in ?outl) (eql ?hp2-in ?out2))))

(defTest Spl-Tstl7 flash-chamber 30.0 (splitter ?splitter ?in ?outl ?out2)
  "A splitter with a sink at one outlet is probably a flash-chamber"
  ((sink ?snk ?s-in)
   :TEST (or (eql ?s-in ?outl) (eql ?s-in ?out2))))

(defTest Spl-Tstl8 flash-chamber 20.0 (splitter ?spl ?m-out ?s-outl ?s-out2)
  "A splitter in between two compressors is likely acting as a flash-chamber
   to remove heat"
  (mixer ?mxr ?m-inl ?m-in2 ?m-out)
  ((compressor ?cmpl ?c-inl ?c-outl)
   :TEST (or (eql ?c-outl ?m-inl) (eql ?c-outl ?m-in2)))
  ((compressor ?cmp2 ?c-in2 ?c-out2)
   :TEST (or (eql ?c-in2 ?s-outl) (eql ?c-in2 ?s-out2))))

(defTest Spl-Tstl9 flash-chamber 7.0 (splitter ?spl ?s-in ?s-outl ?s-out2)
  "A splitter between two turbines with a bleed-path containing a pump is
   probably a flash-chamber, since the pump can only handle liquid"
  (cycle-type :heat-engine ?reason)
  ((bleed-path ?spl ?path-devs)
   :test (and (some #'rf::pump? ?path-devs)
              (notany #'rf::heat-producer? ?path-devs))))
```

```
(defTest Spl-Tst20 flash-chamber 7.0 (splitter ?spl ?s-in ?s-out1 ?s-out2)
  "A splitter downstream of a reactor and upstream of a heat-providing cooler
   is probably a flash-chamber acting as a simple steam reservoir to buffer the
   reactor"
  (cycle-type :heat-engine ?reason)
  (reactor ?rct ?r-in ?s-in)
  (role ?hxc heat-provider ?bp)
  (hx-cooler ?hxc ?hx-in ?hx-out)
  (adjacent ?spl ?hxc ?connection))
```

SPLITTER AS FLOW-FORK

```
(defTest Spl-Tst21 flow-fork 10.0 (splitter ?spl ?in ?out1 ?out2)
  "A splitter with two turbines immediately downstream that both flow into a
   mixer is very likely to be placing the turbines in parallel"
  (turbine ?tur1 ?out1 ?t1-out)
  (turbine ?tur2 ?out2 ?t2-out)
  ((mixer ?mxr ?in1 ?in2 ?m-out)
   :TEST (or (and (eql ?t1-out ?in1)
                  (eql ?t2-out ?in2))
             (and (eql ?t1-out ?in2) (eql ?t2-out ?in1)))))
```

SUBCYCLE DEFINITIONS

```
(defRoleAbst (subcycle ?subc)
  ((work-generator 0.4)
   (heat-mover 0.4)
   (energy-remover 0.1)
   (radiation-isolator 0.1))
  (-> work-generator
    (simple-engine 0.75)
    (topping 0.12)
    (cascading 0.01)
    (bottoming 0.12)))
```

SUBCYCLE AS WORK-GENERATOR

```
(defTest Sbc-Tst1 work-generator 2.5 (subcycle ?subc)
  "A subcycle in a heat-engine containing turbines is very likely
   to be a work generator"
  (cycle-type :heat-engine ?reason)
  (((devs ?subc) members ?devs)
   :TEST (some #'sadi::turbine? ?devs)))

(defTest Sbc-Tst2 simple-engine 2.5 (subcycle ?subc)
  "A single subcycle in a heat-engine is de facto a simple
   work-generating engine"
  (cycle-type :heat-engine ?reason)
  (((subcycles :cycle) members ?subcs)
   :TEST (and (eql ?subc (car ?subcs))
              (null (cdr ?subcs)))))
```

```
(defTest Sbc-Tst3 topping 50.0 (subcycle ?subc1)
  "In a heat-engine with two work-generating subcycles, the subcycle
   supplying heat is virtually always a topping subcycle"
  (cycle-type :heat-engine ?reason)
  (role ?subc1 work-generator ?prob1)
  ((subcycles :cycle) members ?subcs)
  ((subcycle ?subc2)
   :TEST (not (eq ?subc1 ?subc2)))
  (role ?subc2 work-generator ?prob2)
  ((devs ?subc1) members ?devs1)
  ((devs ?subc2) members ?devs2)
  ((hx-cooler ?hxc ?hxc-in ?hxc-out)
   :TEST (member ?hxc ?devs1))
  ((hx-heater ?hxh ?hxh-in ?hxh-out)
   :TEST (member ?hxh ?devs2))
  (heat-path ?hxc ?hxh))

(defTest Sbc-Tst4 bottoming 50.0 (subcycle ?subc1)
  "In a heat-engine with two work-generating subcycles, the subcycle
   absorbing heat is virtually always a bottoming subcycle"
  (cycle-type :heat-engine ?reason)
  (role ?subc work-generator ?prob1)
  ((subcycles :cycle) members ?subcs)
  ((subcycle ?subc2)
   :TEST (not (eq ?subc1 ?subc2)))
  (role ?subc2 work-generator ?prob2)
  ((devs ?subc1) members ?devs1)
  ((devs ?subc2) members ?devs2)
  ((hx-heater ?hxh ?hxh-in ?hxh-out)
   :TEST (member ?hxh ?devs1))
  ((hx-cooler ?hxc ?hxc-in ?hxc-out)
   :TEST (member ?hxc ?devs2))
  (heat-path ?hxc ?hxh))
```

SUBCYCLE AS ENERGY-REMOVER

```
(defTest Sbc-Tst5 energy-remover 100.0 (subcycle ?subc)
  "A refrigerator subcycle with sources and sinks is very likely cooling the
   working fluid"
  (cycle-type :refrigerator ?reason)
  (((devs ?subc) members ?devs)
   :TEST (some #'sadi::terminal? ?devs)))

(defTest Sbc-Tst6 energy-remover 100.0 (subcycle ?subc)
  "A heat-engine subcycle with a pump feeding an hx-heater acting as a
   heat-absorber is probably intended to cool the primary working fluid"
  (cycle-type :heat-engine ?reason)
  (((devs ?subc) members ?devs)
   :TEST (notany 'rf::turbine? ?devs))
  ((dev ?pmp pump)
   :test (member ?pmp ?devs))
  (role ?hxh heat-absorber ?prob)
  (adjacent ?pmp ?hxh ?connection))
```

```
(defTest Sbc-Tst6 energy-remover 100.0 (subcycle ?subc)
  "A heat-engine subcycle with a pump feeding an hx-heater acting as a
   heat-absorber is probably intended to cool the primary working fluid"
  (cycle-type :heat-engine ?reason)
  (((devs ?subc) members ?devs)
   :TEST (notany 'rf::turbine? ?devs))
  ((hx-heater ?hxh ?in ?out)
   :test (member ?hxh ?devs))
  (heat-path ?hxc ?hxh)
  ((primary-floop-for ?main-subc ?floop)
   :test ?floop)
  (((devs ?main-subc) members ?devs-main)
   :test (member ?hxc ?devs-main))
  ((dev ?pmp pump)
   :test (member ?pmp ?devs))
  (role ?hxh heat-absorber ?prob)
  (adjacent ?pmp ?hxh ?connection))
```

SUBCYCLE AS HEAT-MOVER

```
(defTest Sbc-Tst7 heat-mover 8.0 (subcycle ?subc)
  "A heat-engine subcycle with only pumps, heaters, and coolers is likely
   transporting thermal energy from one location to another"
  (cycle-type :heat-engine ?reason)
  (((devs ?subc) members ?devs)
   :TEST (and (some 'rf::heat-changer? ?devs)
              (some 'rf::work-consumer? ?devs)
              (notany 'rf::reactor? ?devs)
              (notany 'rf::work-producer? ?devs))))

(defTest Sbc-Tst8 heat-mover 2.5 (subcycle ?subc)
  "A refrigerator subcycle with no sources or sinks is almost certainly moving
   heat"
  (cycle-type :refrigerator ?reason)
  (((devs ?subc) members ?devs)
   :TEST (notany #'sadi::terminal? ?devs)))
```

SUBCYCLE AS RADIATION ISOLATOR

```
(defTest Sbc-Tst9 radiation-isolator 9.9 (subcycle ?subc)
  "A subcycle containing a reactor but no turbines is probably isolating the
   reactor"
  (cycle-type :heat-engine ?reason)
  (((devs ?subc) members ?devs)
   :TEST (and (notany #'sadi::turbine? ?devs)
              (some #'sadi::reactor? ?devs))))
```

THROTTLE DEFINITIONS

```
(defTypeAbst (THROTTLE ?throttle ?in ?out)
   free-expander
   pressure-changer
   device)

(defRoleAbst (THROTTLE ?throttle ?in ?out)
   ((pressure-decreaser 0.5)
    (saturator 0.5)))
```

THROTTLE AS SATURATOR

```
(defTest Thr-Tst1 saturator 2.5 (throttle ?throttle ?in ?out)
  "A throttle immediately upstream of a splitter in a refrigerating cycle is
   very likely to be saturating the working fluid to enable it to flash"
  (cycle-type :refrigerator ?reason)
  (splitter ?splitter ?out ?out2 ?out3))

(defTest Thr-Tst2 saturator 3.0 (throttle ?throttle ?c-out ?h-in)
  "A throttle with a cooler upstream and a heater downstream is most commonly
   acting as a saturator of the working fluid"
  (cycle-type :refrigerator ?reason)
  (genus ?cooler heat-producer)
  (?cooler has-outstf ?c-out)
  (genus ?heater heat-consumer)
  (?heater has-instf ?h-in))

(defTest Thr-Tst3 saturator 2.5 (throttle ?thr ?thr-in ?thr-out)
  "A throttle with a downstream heater in a refrigerating cycle is probably
   intended to saturate the working fluid"
  (cycle-type :refrigerator ?reason)
  (genus ?htr heat-consumer)
  (adjacent ?thr ?htr ?stfs))

(defTest Thr-Tst4 saturator 5.0 (throttle ?throttle ?in ?out)
  "A throttle on a refrigerator cycle is more likely to be a saturator than a
   pressure-decreaser"
  (cycle-type :refrigerator ?reason)
  (subcycle ?subcycle)
  ((role ?subcycle ?subc-role ?prob)
   :TEST (member ?subc-role '(heat-mover energy-remover)))
  ((devs ?subcycle) has-member ?throttle))

(defTest Thr-Tst5 saturator 6.0 (throttle ?throttle ?in ?out)
  "A throttle entering a mixer in a refrigerator cycle is likely to be a
   saturator"
  (cycle-type :refrigerator ?reason)
  ((mixer ?mixer ?in1 ?in2 ?out2)
   :TEST (or (eql ?out ?in1) (eql ?out ?in2))))
```

THROTTLE AS PRESSURE DECREASER

```
(defTest Thr-Tst6 pressure-decreaser 8.0 (throttle ?throttle ?in ?out)
  "A throttle on a work-producing loop in a refrigerator cycle is more likely
   to be a pressure-decreaser than a saturator"
  (cycle-type :refrigerator ?reason)
  (role ?subcycle work-generator ?prob)
  ((devs ?subcycle) has-member ?throttle))

(defTest Thr-Tst7 pressure-decreaser 3.0 (throttle ?throttle ?in ?out)
  "A throttle entering a mixer in a heat-engine cycle is probably a
   pressure-decreaser"
  (cycle-type :heat-engine ?reason)
  ((mixer ?mixer ?in1 ?in2 ?out2)
   :TEST (or (eql ?out ?in1) (eql ?out ?in2))))

(defTest Thr-Tst8 pressure-decreaser 50.0 (throttle ?throttle ?link ?out)
  "A throttle immediately downstream of a heater is probably
   a pressure-decreaser"
  added in response to tsb analysis of cycle 2
  (heater ?htr ?htr-in ?link))
```

TURBINE DEFINITIONS

```
(defTypeAbst (TURBINE ?turbine ?in ?out)
  work-producer
  pressure-changer
  device)

(defRoleAbst (TURBINE ?turbine ?in ?out)536
  ((work-source 0.95) (fluid-cooler 0.05)))
```

TURBINE AS WORK-SOURCE

```
(defTest Tur-Tst1 work-source 5.0 (turbine ?tur ?in ?out)
  "A turbine in a heat-engine cycle is virtually always a work-source"
  (cycle-type :heat-engine ?reason))

(defTest Tur-Tst2 work-source 5.0 (turbine ?turbine ?t-in ?t-out)
  "A turbine in a refrigerator cycle with a heater immediately upstream is
   most likely generating power to drive the compressors of the cycle"
  (cycle-type :refrigerator ?reason)
  (genus ?heater heat-consumer)
  (?heater has-outstf ?t-in))
```

TURBINE AS FLUID-COOLER

```
(defTest Tur-Tst3 fluid-cooler 50.0 (turbine ?turbine ?in ?out)
  "A turbine on a refrigerator cycle without any heat-injectors
   is more likely to be a fluid-cooler"
  (cycle-type :refrigerator ?reason)
  (((heat-consumers :CYCLE) members ?devs)
   :test (null ?devs)))

(defTest Tur-Tst4 fluid-cooler 160.0 (turbine ?tur ?in ?out)
  "A turbine in a refrigerating cycle with a cooler immediately upstream
   is probably being used to achieve a greater temperature drop than would
   occur through a throttle"
  (cycle-type :refrigerator ?reason)
  (genus ?cooler heat-producer)
  (?cooler has-outstf ?in))

(defTest Tur-Tst5 fluid-cooler 200.0 (turbine ?tur ?t-in ?t-out)
  "A turbine in a refrigerating cycle with an upstream cooler and a downstream
   heater is probably being used to achieve a greater temperature drop than
   would occur through a throttle"
  (cycle-type :refrigerator ?reason)
  (genus ?cooler heat-producer)
  ((downrange ?cooler ?c-range)
   :TEST (member ?tur ?c-range))
  (downrange ?tur ?t-range)
  ((genus ?heater heat-consumer)
   :TEST (member ?heater ?t-range)))
```

Plan Definitions

PLANS FOR ACHIEVING A CHANGE IN THE ENVIRONMENT

```
(defPlan generate-work ()
  "This cycle is a heat-engine, so it is intended to produce a change in the
   environment, namely work"
  :goal (:achieve-change-in-env)
  :conditions
  ((cycle-type :heat-engine ?reason)))

(defPlan move-heat ()
  "This is a refrigerator cycle, so it is intended to move heat from one
   location to another"
  :goal (:achieve-change-in-env)
  :conditions
  ((cycle-type :refrigerator ?reason)))

(defPlan liquefy-gas (move-heat)
  "In this case the heat is being ejected in order to liquefy the fluid flowing
   from source to sink"
  :goal (:achieve-change-in-env)
  :key-roles ((role ?src fluid-source ?bp1)
              (role ?snk fluid-sink ?bp2))
  :conditions
  (((floop ?fname ?fstfs ?fdevs)
    :test (and (eq ?src (car ?fdevs)) (eq ?snk (car (last ?fdevs)))))))

(defPlan vapor-power-cycle (generate-work)
  "This heat engine fully condenses its working fluid in order to realize
   gains in pumping efficiency over compressing gas or saturated mixtures"
  :goal (:achieve-change-in-env)
  :key-roles ((role ?subc work-generator ?bp))
  :conditions
  ((((devs :CYCLE) members ?devs)
    :test (notany #'rf::compressor? ?devs))
   (cooler ?clr ?in ?out)
   ((phase ?out ?phase)
    :test (not (eq ?phase 'data::gas)))))

(defPlan combined-vapor-gas-cycle (generate-work)
  "This heat engine combines the greater maximum temperatures of a gas cycle
   with the thermal advantage of the vapor cycle in ejecting all its heat at
   the lowest possible temperature"
  :goal (:achieve-change-in-env)
  :key-roles ((role ?subc work-generator ?bp))
  :conditions
  ((((devs ?subc) members ?devs)
    :test (and (some #'rf::compressor? ?devs)
               (some #'rf::pump? ?devs)))))

(defPlan gas-power-cycle (generate-work)
  "This heat engine utilizes gas working fluids throughout, which typically
   results in lower weight and a more compact design"
  :goal (:achieve-change-in-env)
  :conditions
  ((cycle-type :heat-engine ?reason)
   (((devs :CYCLE) members ?devs)
    :test (and (some 'rf::compressor? ?devs)
               (notany 'pump? ?devs)))))
```

```
(defPlan simple-gas-power-cycle (gas-power-cycle)
  "This heat engine utilizes gas working fluids throughout, which typically
   results in lower weight and a more compact design"
  :goal (:achieve-change-in-env)
  :conditions
  ((cycle-type :heat-engine ?reason)
   (((devs :CYCLE) members ?devs)
    :test (and (rf::exactly-one 'rf::compressor? ?devs)
               (rf::exactly-one 'rf::turbine? ?devs)
               (notany 'hx? ?devs) ;;no regeneration, intercooling
               (notany 'pump? ?devs)))))
```

PLANS FOR INCREASING THE EFFICIENCY OF THE CYCLE

```
(defPlan direct-regenerate-via-closed-hx ()
  "Ideally we could extract heat from the turbine as the gas expands through
   it. In practice this causes unacceptable saturation of the fluid"
  :goal (:increase-efficiency)
  :NL-phrase "the working fluid flowing through the turbine is used to preheat
              the feed fluid"
  :key-roles ((role ?pre preheater ?prob))
  :conditions
  ((hx-heater ?pre ?pre-in ?pre-out)
   (heat-path ?clr ?pre)
   ((floop ?fname ?stfs ?devs)
    :test (and (member ?pre ?devs)
               (member ?clr ?devs)))
   (hx-cooler ?clr ?clr-in ?clr-out)
   (turbine ?tur ?clr-out ?tur-out)))

(defPlan reheat-vapor-cycle-working-fluid ()
  "Reheating in a vapor power cycle enables the use of higher turbine inlet
   pressures, which increases efficiency.  Reheat ensures that the exit vapor
   is of high quality.  If turbine blades could withstand higher inlet
   temperatures then reheat would be unnecessary"
  :goal (:increase-efficiency)
  :NL-phrase "energy is injected into the working fluid as it flows through
              the turbine"
  :key-roles ((role ?rhtr reheater ?prob))
  :conditions ((plan ?instance vapor-power-cycle)))

(defPlan reheat-gas-cycle-working-fluid ()
  "Reheating in a gas power cycle takes advantage of the fact that oxygen
   remains in the working fluid even after the initial combustion, so injection
   of more fuel enables further extraction of energy from the exhaust of the
   first  turbine stages.  In an airplane engine this would be called an
   afterburner"
  :goal (:increase-efficiency)
  :NL-phrase "energy is injected into the working fluid as it flows through
              the turbine"
  :key-roles ((role ?rhtr reheater ?prob))
  :conditions ((plan ?instance gas-power-cycle)))

(defPlan intercool-compressors ()
  "Intercooling reduces the amount of work necessary to compress a gas, thus
   increasing cycle efficiency"
  :goal (:increase-efficiency)
  :NL-phrase "energy is ejected from the compressor"
  :key-roles ((role ?intrclr intercooler ?prob))
  :conditions
  ((cooler ?intrclr ?clr-in ?clr-out)))
```

```
(defPlan intercool-and-use-ejected-heat (intercool-compressors)
  "Using the ejected heat from an intercooler increases the efficiency of
   intercooling"
  :goal (:increase-efficiency)
  :NL-phrase "energy ejected from the compressor is used as a source of heat"
  :key-roles ((role ?intrclr intercooler ?prob))
  :conditions ((hx-cooler ?intrclr ?clr-in ?clr-out)))

(defPlan indirect-contact-regenerate ()
  "Heat-exchange via closed heat-exchangers is less efficient than direct
   mixing, but the two working fluids can be at different pressures, reducing
   the need for pumps"
  :goal (:increase-efficiency)
  :NL-phrase "working fluid bled from the turbine is used to supply heat in a
              heat-exchanger"
  :key-roles ((role ?pre preheater ?prob))
  :conditions
  ((cycle-type :heat-engine ?reason)
   (bleed-path ?bleed-valve ?devs)
   ((heat-path ?clr ?pre)
    :test (member ?clr ?devs))))

(defPlan preheat-with-exhaust-gas ()
  "This plan takes advantage of the high temperature exhaust of a gas-turbine
   to preheat the working fluid"
  :goal (:increase-efficiency)
  :key-roles ((role ?pre preheater ?prob))
  :NL-phrase "the relatively hot exhaust gas is piped through a closed heat-
              exchanger in order to preheat the working fluid entering the
              combustion chamber"
  :conditions
  ((((devs :CYCLE) members ?devs)
    :test (notany #'rf::bleed-valve? ?devs))
   (hx-heater ?pre ?pre-in ?pre-out)
   (heat-path ?clr ?pre)
   (turbine ?last-tur ?lt-in ?lt-out)
   ((downrange ?last-tur ?last-tur-dr)
    :test (notany #'rf::turbine? ?last-tur-dr))
   (adjacent ?last-tur ?clr ?connection)
   ((downrange ?clr ?clr-downrange)
    :test (notany #'rf::turbine? ?clr-downrange))
   ((floop ?fname ?stfs ?fdevs)
    :test (and (member ?pre ?fdevs)
               (member ?clr ?fdevs)))))

(defPlan direct-regenerate-with-minimal-delta-T (direct-contact-regenerate)
  "Direct contact is the most efficient form of heat exchange, but a large
   temperature difference in the streams will lead to irreversibilities.
   Increasing the pressure and intercooling a working-fluid bleed can convert
   it to a dry saturated gas at the same temperature as the wet saturated
   liquid to be preheated"
  :goal (:increase-efficiency)
  :NL-phrase "superheated fluid bled from the turbine is saturated then is
              directly mixed with the boiler feed liquid"
  :key-roles ((role ?mxr mxr-heater ?bp))
  :conditions
  ((role ?intrclr intercooler ?prob)
   (hx-cooler ?intrclr ?clr-in ?clr-out)
   ((bleed-path ?bleed-valve ?devs)
    :test (and (member ?mxr ?devs)
               (member ?intrclr ?devs)))))
```

```
(defPlan bleed-working-fluid-for-industrial-process ()
  "Conversion of energy from one form to another inevitably entails
   irreversibility, so using vapor generated for a power cycle is more
   efficient than using the power generated to produce the vapor"
  :goal (:increase-efficiency)
  :NL-phrase "vapor is bled for use in an industrial process"
  :key-roles ((role ?clr heat-provider ?prob))
  :conditions
  ((heat-path ?clr :ENV)
   ((bleed-path ?bleed-valve ?devs)
    :test (member ?clr ?devs))))

(defPlan gas-liquid-cascaded-cycle (generate-work)
  "Cascading cycles enables the use of working fluids with different thermal
   characteristics in the same system.  In this case a high-temperature gas
   cycle's waste heat is used to power a vapor cycle"
  :goal (:increase-efficiency)
  :NL-phrase "it is partitioned into two subcycles, one powering the other via
              its ejected heat"
  :key-roles ((role ?subc1 topping ?bp1)
              (role ?subc2 bottoming ?bp2))
  :conditions
  ((((devs ?subc1) members ?subc1-devs)
    :test (notany #'rf::pump? ?subc1-devs))
   (((devs ?subc2) members ?subc2-devs)
    :test (notany #'rf::compressor? ?subc2-devs))))

(defPlan refrigerating-cascaded-cycle (move-heat)
  "Cascading cycles enables the use of working fluids with different thermal
   characteristics in the same system.  In this case the energy-removing
   subcycle can use a refrigerant that operates at substantially different
   pressures in order to achieve greater heat-transfer"
  :goal (:increase-efficiency)
  :NL-phrase "it is partitioned into multiple subcycles, one removing heat from
              the other"
  :key-roles ((role ?subc1 heat-mover ?bp1)
              (role ?subc2 heat-mover ?bp2))
  :conditions
  (((devs ?subc1) members ?subc1-devs)
   ((devs ?subc2) members ?subc2-devs)
   ((heat-path ?cooler ?heater)
    :test (and (member ?cooler ?subc1-devs)
               (member ?heater ?subc2-devs)))))
```

```
(defPlan transport-heat (move-heat)
  "Heat-engines generally convert energy into electrical form for efficient
   distribution.  However, in some situations it may be more efficient to
   transport heat energy from the system directly to its point of use, thereby
   avoiding conversion inefficiencies"
  :goal (:increase-efficiency)
  :NL-phrase "a subcycle is employed to transport heat to another location"
  :key-roles ((role ?subc1 work-generator ?bp1)
              (role ?subc2 heat-mover ?bp3))
  :conditions
  ((cycle-type :heat-engine ?reason)
   ((devs ?subc1) members ?subc1-devs)
   (((devs ?subc2) members ?subc2-devs)
    :test (not (eq ?subc1 ?subc2)))
   ((role ?hxc1 heat-ejector ?bp4)
    :test (member ?hxc1 ?subc1-devs))
   ((role ?hxh1 heat-absorber ?bp5)
    :test (member ?hxh1 ?subc2-devs))
   (heat-path ?hxc1 ?hxh1)
   ((role ?clr heat-provider ?bp8)
    :test (member ?clr ?subc2-devs))))

(defPlan multi-transport-heat (move-heat)
  "Heat-engines generally convert energy into electrical form for efficient
   distribution.  However, in some situations it may be more efficient to
   transport heat energy from the system directly to its point of use, thereby
   avoiding conversion inefficiencies"
  :goal (:increase-efficiency)
  :NL-phrase "subcycles are employed to transport heat to another location"
  :key-roles ((role ?subc1 work-generator ?bp1)
              (role ?subc2 energy-remover ?bp2)
              (role ?subc3 heat-mover ?bp3))
  :conditions
  ((cycle-type :heat-engine ?reason)
   ((devs ?subc1) members ?subc1-devs)
   (((devs ?subc2) members ?subc2-devs)
    :test (not (eq ?subc1 ?subc2)))
   (((devs ?subc3) members ?subc3-devs)
    :test (and (not (eq ?subc2 ?subc3))
               (not (eq ?subc1 ?subc3))))
   ((role ?hxc1 heat-ejector ?bp4)
    :test (member ?hxc1 ?subc1-devs))
   ((role ?hxh1 heat-absorber ?bp5)
    :test (member ?hxh1 ?subc2-devs))
   (heat-path ?hxc1 ?hxh1)
   ((role ?hxc2 heat-provider ?bp6)
    :test (member ?hxc2 ?subc2-devs))
   ((role ?hxh2 heat-injector ?bp7)
    :test (member ?hxh2 ?subc3-devs))
   (heat-path ?hxc2 ?hxh2)
   ((role ?clr heat-provider ?bp8)
    :test (member ?clr ?subc3-devs))))
```

```
(defPlan exchange-heat-at-multiple-pressure ()
  "Heat-exchange at multiple-pressures minimizes the temperature difference
   between the two working fluids, thereby creating less irreversibility.  A
   relatively cool, high-pressure saturated liquid can absorb heat from a
   high-temperature working fluid, cooling that fluid to the point where it can
   supply heat to the same saturated liquid at a lower temperature"
  :goal (:increase-efficiency)
  :NL-phrase "heat is injected into the work-generating subcycle at multiple
              pressures"
  :key-roles ((role ?subc1 heat-mover ?bp1)
              (role ?subc2 work-generator ?bp2))
  :conditions
  ((((devs ?subc1) members ?subc-devs1)
    :test (> (count-if #'rf::hx-cooler? ?subc-devs1) 1))
   (((devs ?subc2) members ?subc-devs2)
    :test (or (> (count-if #'rf::pump? ?subc-devs2) 1)
              (> (count-if #'rf::compressor? ?subc-devs2) 1)))))

(defPlan nuclear-exchange-heat-at-multiple-pressure ()
  "Heat-exchange at multiple-pressures minimizes the temperature difference
   between the two working fluids, thereby creating less irreversibility.  A
   relatively cool, high-pressure saturated liquid can absorb heat from a high-
   temperature working fluid, cooling that fluid to the point where it can
   supply heat to the same saturated liquid at a lower temperature"
  :goal (:increase-efficiency)
  :NL-phrase "heat is injected into the work-generating subcycle at multiple
              pressures"
  :key-roles ((role ?subc1 radiation-isolator ?bp1)
              (role ?subc2 work-generator ?bp2))
  :conditions
  ((((devs ?subc1) members ?subc-devs1)
    :test (> (count-if #'rf::hx-cooler? ?subc-devs1) 1))
   (((devs ?subc2) members ?subc-devs2)
    :test (or (> (count-if #'rf::pump? ?subc-devs2) 1)
              (> (count-if #'rf::compressor? ?subc-devs2) 1)))))

(defPlan cool-reactor-feed-to-adjust-load ()
  "Liquid-moderated reactors may be controlled by cooling their inlet
   working-fluid, which will result in the raising of more vapor"
  :goal (:increase-efficiency)
  :NL-phrase "working fluid is used to cool the fluid flowing through the
              reactor core"
  :key-roles ((role ?hxc heat-provider ?bp1))
  :conditions
  ((hx-cooler ?hxc ?hxc-in ?hxc-out)
   (reactor ?rct ?hxc-out ?rct-out)))

(defPlan use-turbine-to-obtain-greater-T-drop (move-heat)
  "Using a turbine in place of a throttle to create the pressure drop in a
   refrigerator results in a greater decline in temperature, which is often
   useful in a cryogenic cycle"
  :goal (:increase-efficiency)
  :NL-phrase "a turbine is used to cool the working fluid"
  :key-roles ((role ?tur fluid-cooler ?bp))
  :conditions
  ((turbine ?tur ?in ?out)
   (cycle-type :refrigerator ?reason)))
```

```
(defPlan use-flash-tank ()
  "A flash-tank is an efficient means for cooling the working fluid by direct
   contact"
  :goal (:increase-efficiency)
  :NL-phrase "a flash-tank is used to cool the working fluid"
  :key-roles ((role ?mxr mxr-cooler ?bp1)
              (role ?spl flash-chamber ?bp2))
  :conditions
  ((cycle-type :refrigerator ?reason)
   (aggr ?mpc mp-chamber)
   (?mpc has-devs (?mxr ?spl))))

(defPlan direct-contact-regenerate ()
  "Heat-exchange via mixing is more efficient than indirect transfer in closed
   heat-exchangers, and it also enables deaeration of working fluid, but it
   requires more pumps because inlets must be maintained at the same pressure"
  :goal (:increase-efficiency :preserve-system)
  :NL-phrase "fluid bled from the turbine is directly mixed with the boiler
              feed liquid"
  :key-roles ((role ?mxr mxr-heater ?prob))
  :conditions
  ((cycle-type :heat-engine ?reason)
   ((bleed-path ?bleed-valve ?devs)
    :test (member ?mxr ?devs))))
```

PLANS FOR PRESERVING THE SYSTEM

```
(defPlan flash-to-dry-vapor ()
  "Reactors often generate vapor of lower quality, and hence may need
   flash-chambers in the turbine series to dry the vapor entering the latter
   stages"
  :goal (:preserve-system)
  :NL-phrase "the working fluid flowing through the turbine is flashed to
              improve its quality"
  :key-roles ((role ?spl flash-chamber ?prob))
  :conditions
  ((turbine ?tur1 ?tur1-in ?tur1-out)
   ((downrange ?tur ?tur-downrange)
    :test (member ?spl ?tur-downrange))
   ((downrange ?spl ?spl-downrange)
    :test (some #'rf::turbine? ?spl-downrange))
   (bleed-path ?spl ?devs)))

(defPlan bleed-vapor-for-reheat ()
  "Since a reactor generally does not achieve much in the way of superheat,
   reheating the vapor flowing to later turbine stages is often necessary for
   preserving the integrity of the turbine blades"
  :goal (:preserve-system)
  :NL-phrase "vapor bled from the turbine is used to reheat the fluid in later
              turbine stages"
  :key-roles ((role ?rhtr reheater ?prob))
  :conditions
  (((heat-path ?heat-provider ?rhtr)
    :test (not (eq ?heat-provider :ENV)))
   ((bleed-path ?bleed-valve ?devs)
    :test (member ?heat-provider ?devs))))

(defPlan isolate-reactor ()
  "Nuclear plants often use a separate subcycle to contain radiation"
  :goal (:preserve-system)
  :NL-phrase "a separate subcycle is used to contain the radiation of the
              reactor core"
  :key-roles ((role ?subc radiation-isolator ?prob)))
```

```
(defPlan use-jet-ejector-to-power-refrigerator (move-heat)
  "Jet ejectors have no moving parts and hence vapor refrigeration
   systems that use them for compression are inexpensive and safe to
   operate, although they produce effective cooling to levels well above the
   freezing point of the working fluid"
  :goal (:preserve-system)
  :NL-phrase "a jet-ejector is used to compress the working fluid"
  :key-roles ((role ?mxr jet-ejector ?bp-mxr))
  :conditions
  ((cycle-type :refrigerator ?reason)))

(defPlan use-vapor-drum-to-preserve-system ()
  "A vapor-drum buffers the system against sudden changes in load"
  :goal (:preserve-system)
  :NL-phrase "a vapor-drum is used to maintain a ready supply of gaseous
              working fluid"
  :key-roles ((role ?mxr mxr-heater ?bp1)
              (role ?spl flash-chamber ?bp2))
  :conditions
  ((cycle-type :heat-engine ?reason)
   (aggr ?mpc mp-chamber)
   (?mpc has-devs (?mxr ?spl))))
```

Appendix D
Formal Definitions of Locality Predicates

Adjacent Predicate

$$\forall x, y \ \text{Adjacent}(x, y) \Leftrightarrow \left\{\begin{array}{l} \text{device}(x) \wedge \text{device}(y) \wedge \text{outlet}(x) = \text{inlet}(y) \\ \exists z \ \text{mixer}(z) \wedge \text{flowjoin}(z) \wedge \text{outlet}(x) = \text{inlet}(z) \wedge \\ \qquad\qquad \text{outlet}(z) = \text{inlet}(y) \\ \exists z \ \text{splitter}(z) \wedge \text{flowfork}(z) \wedge \text{outlet}(x) = \text{inlet}(z) \wedge \\ \qquad\qquad \text{outlet}(z) = \text{inlet}(y) \\ \exists z \ \text{mixer}(z) \wedge \text{flowjoin}(z) \wedge \text{Adjacent}(x, z) \wedge \text{Adjacent}(z, y) \\ \exists z \ \text{splitter}(z) \wedge \text{flowsplit}(z) \wedge \text{Adjacent}(x, z) \wedge \text{Adjacent}(z, y) \end{array}\right\}$$

Downrange Predicate

$$\forall x, y \ \text{Upstream}(x, y) \Leftrightarrow \left\{\begin{array}{l} \text{Device}(x) \wedge \text{Device}(y) \\ \wedge \ \text{Outlet}(x) = \text{Inlet}(y) \end{array}\right\}$$

$$\forall x, y, P \ \ \text{Path}(x, y, P) \Leftrightarrow \left\{\begin{array}{l} \text{Device}(x) \wedge \text{Device}(y) \wedge \text{Set}(P) \\ \wedge \ \forall z \ \ z \in P \ \text{Device}(z) \\ \wedge \ \forall z \ \ z \in P \wedge z \neq y \ \exists w \ \ w \in P \wedge \text{Upstream}(z, w) \end{array}\right\}$$

$$\forall x, e \ \ \text{OpposingEffect}(x) = e \Leftrightarrow \left\{\begin{array}{l} \text{Device}(x) \wedge \text{Effect}(e) \wedge \\ \text{Type}(x) = \text{heater} \wedge e = \text{cooling} \\ \vee \ \text{Type}(x) = \text{cooler} \wedge e = \text{heating} \\ \vee \ \text{Type}(x) = \text{turbine} \wedge e = \text{compressing} \\ \vee \ \text{Type}(x) = \text{compressor} \wedge e = \text{expanding} \\ \vee \ \text{Type}(x) = \text{pump} \wedge e = \text{expanding} \\ \vee \ \text{Type}(x) = \text{throttle} \wedge e = \text{compressing} \\ \vee \ \text{Type}(x) = \text{mixer} \wedge e = \text{splitting} \\ \vee \ \text{Type}(x) = \text{splitter} \wedge e = \text{mixing} \end{array}\right\}$$

$$\forall x, y \ \text{Downrange}(x, y) \Leftrightarrow \left\{\begin{array}{l} \text{Device}(x) \wedge \text{Device}(y) \wedge \text{Path}(x, y, P) \\ \wedge \ \forall z \ z \in P \ \ \text{OpposingEffect}(x) \neq \text{Effect}(z) \end{array}\right\}$$

References

Allemang, Dean T. 1990. Understanding Programs as Devices. Ph.D. dissertation, The University of Ohio.

Baffes, Paul and Raymond Mooney. 1996. Refinement-Based Student Modeling and Automated Bug Library Construction. *Journal of Artificial Intelligence in Education* 7, no. 1: 75-116.

Bhatta, Sambasiva R. and Ashok Goel. 1993. Model-Based Learning of Structural Indices to Design Cases. In *Proceedings of the Workshop on Reuse of Designs: An Interdisciplinary Cognitive Approach at the 1993 International Joint Conference on Artificial Intelligence*. Chambery, Savoie, France.

Brown, J.S. and R.R. Burton. 1978. Diagnostic Models for Procedural Bugs in Basic Mathematical Skills. *Cognitive Science* 2: 155-192.

Brown, J.S. and K. Van Lehn. 1980. Repair Theory: A Generative Theory of Bugs in Procedural Skills. *Cognitive Science* 4: 379-426.

Buchanan, B.G. and E.H. Shortliffe. 1984. *Rule-based expert systems*. Reading, MA: Addison-Wesley.

Bylander, Thomas and B. Chandrasekaran. 1985. Understanding Behavior Using Consolidation. In *Ninth International Joint Conference on Artificial Intelligence*:450-454.

Chandrasekaran, B. 1994. Functional Representation and Causal Processes. In *Advances in Computers*, ed. Marshall Yovits, 38: Academic Press.

Charniak, Eugene. 1991. Bayesian Networks without Tears. *AI Magazine*, Winter 1991, 50-63.

Davis, Randall. 1983. Diagnosis via Causal Reasoning: Paths of Interaction and the Locality Principle. In *Proceedings of the National Conference on Artificial Intelligence*, 1:88-94. Washington, D.C.: Morgan Kaufmann.

de Groot, Adriaan D. 1978. *Thought and Choice in Chess*. New York NY: Mouton.

de Kleer, Johan. 1975. Qualitative and Quantitative Knowledge in Classical Mechanics. Technical Report 352. Massachusetts Institute of Technology.

de Kleer, Johan. 1979. Causal and Teleological Reasoning in Circuit Recognition. Technical Report AI-TR-529. Massachusetts Institute of Technology Artificial Intelligence Laboratory.

de Kleer, Johan. 1984. How Circuits Work. *Artificial Intelligence* 24, no. 1-3: 205-280.

de Kleer, Johan and John S. Brown. 1986. Theories of Causal Ordering. *Artificial Intelligence* 29, no. 1: 33-61.

de Kleer, Johan and Brian Williams. 1987. Diagnosing Multiple Faults. *Artificial Intelligence* 32: 97-130.

Everett, John O. 1995. A Theory of Mapping from Structure to Function Applied to Engineering Domains. In *International Joint Conference on Artificial Intelligence*, 2:1837-1843. Montreal: Morgan Kaufmann.

Falkenhainer, Brian. 1986. Toward a General Purpose Belief Maintenance System. In *The Second Workshop on Uncertainty in AI*:71-76.

Falkenhainer, Brian and Kenneth D. Forbus. 1991. Compositional Modeling: Finding the Right Model for the Job. *Artificial Intelligence* 51, no. 1: 95-143.

Falkenhainer, Brian, Kenneth D. Forbus, and Dedre Gentner. 1989. The Structure Mapping Engine: Algorithm and Examples. *Artificial Intelligence* 41, no. 1: 1-63.

Forbus, Kenneth D. 1984. Qualitative Process Theory. *Artificial Intelligence* 24: 85-168.

Forbus, Kenneth D. 1990. The Qualitative Process Engine. In *Readings in Qualitative Reasoning about Physical Systems*, ed. Daniel S. Weld and Johan de Kleer:220-235. San Mateo, CA: Morgan Kaufmann.

Forbus, Kenneth D. and Johan de Kleer. 1993. *Building Problem Solvers.* Edited by Michael Brady, Daniel Bobrow, and Randall Davis. Artificial Intelligence. Cambridge, MA: The MIT Press.

Forbus, Kenneth D., Dedre Gentner, John O. Everett, and Melissa Wu. 1997. Towards a Computational Model of Evaluating and Using Analogical Inferences. In *Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society.* Palo Alto, CA: Lawrence Erlbaum Associates.

Forbus, Kenneth D., Dedre Gentner, and B.K. Law. 1995. MAC/FAC: A Model of Similarity-Based Retrieval. *Cognitive Science* 19, no. 2: 141-205.

Forbus, Kenneth D. and Peter B. Whalley. 1994. Using qualitative physics to build articulate software for thermodynamics education. In *Twelfth National Conference on Artificial Intelligence*. Seattle, WA: AAAI Press/MIT Press.

Franke, David. 1993. A Theory of Teleology. Ph.D. dissertation, The University of Texas.

Gentner, Dedre. 1983. Structure-Mapping: A Theoretical Framework for Analogy. *Cognitive Science* 23: 155-170.

Goel, Ashok. 1991. A Model-Based Approach to Case Adaptation. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*: 143-148. Chicago, IL.

Goldman, Robert P. and Eugene Charniak. 1993. A Language for Construction of Belief Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15, no. 3.

Hayes-Roth, B., R. Washington, D. Ash, A. Hewett, A. Collinot, and A. Seiver. 1992. Guardian: A Prototype Intelligent Agent for Intensive Care Monitoring. *Artificial Intelligence in Medicine* 4: 165-185.

Haywood, R.W. 1991. *Analysis of Engineering Cycles: Power, Refrigerating, and Gas Liquefaction Plant*. Edited by W.A. Woods. Thermodynamics and Fluid Mechanics. Oxford: Pergamon Press.

Henrion, Max, ed. 1988. *Propagation of Uncertainty in Bayesian Networks by Probabilistic Logic Sampling*. Edited by L. N. Kanal and J. F. Lemmer. Uncertainty in Artificial Intelligence. Amsterdam, London, New York: Elsevier/North-Holland.

Horvitz, Eric J., H. J. Suermondt, and G. F. Cooper. 1989. Bounded Conditioning: Flexible Inference for Decisions under Scarce Resources. In *Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence (UAI-89)*:182-193. Windsor, Ontario: Morgan Kaufmann.

Iwasaki, Yumi and Chee Meng Low. 1993. Model Generation and Simulation of Device Behavior with Continuous and Discrete Changes. *Journal of Intelligent Systems Engineering* 1, no. 2.

Iwasaki, Yumi and Herbert A. Simon. 1986. Causality in Device Behavior. *Artificial Intelligence* 29, no. 1: 3-32.

Jensen, Finn V., S. L. Lauritzen, and K. G. Olesen. 1990. Bayesian Updating in Causal Probabilistic Networks by Local Computations. *Computational Statistics Quarterly* 5, no. 4: 269-282.

Keuneke, Anne Marie. 1989. Machine Understanding of Devices: Causal Explanation of Diagnostic Conclusions, The University of Ohio.

Keuneke, Anne M. and Dean Allemang. 1989. Exploring the 'No-Function-In-Structure' Principle. *Journal of Experimental and Theoretical Artificial Intelligence* 1: 79-89.

Kuipers, Benjamin. 1986. Qualitative Simulation. *Artificial Intelligence* 29: 289-388.

Larsson, Jan Eric. 1996. Diagnosis Based on Explicit Means-Ends Models. *Artificial Intelligence* 80, no. 1: 29-93.

Lind, Morten. 1982. Multilevel Flow Modelling of Process Plant for Diagnosis and Control. Technical Risø-M-2357. Risø National Laboratory.

Lind, Morten. 1990. Representing Goals and Functions of Complex Systems: An Introduction to Multilevel Flow Modeling. Technical 90-D-381

ISBN 87-87950-52-9. Technical University of Denmark.

McAllester, David. 1978. A Three-Valued Truth Maintenance System. S.B., MIT.

McAllester, David A. 1990. Truth Maintenance. In *Eighth National Conference on Artificial Intelligence*:1109-1116: Morgan Kaufmann.

Narayanan, N. Hari, Masaki Suwa, and Hiroshi Motoda. 1995. Hypothesizing Behaviors from Device Diagrams. In *Diagrammatic Reasoning: Computational and Cognitive Perspectives*, ed. Janice Glasgow, N. Hari Narayanan, and B. Chandrasekaran. Menlo Park, CA: AAAI Press.

Nayak, P. Pandurang and Leo Joskowicz. 1996. Efficient Compositional Modeling for Generating Causal Explanations. *Artificial Intelligence* 83, no. 2: 193-227.

Nilsson, Nils J. 1986. Probabilistic Logic. *Artificial Intelligence* 28, no. 1: 71-87.

Pearl, Judea. 1986. Fusion, Propagation, and Structuring in Belief Networks. *Artificial Intelligence* 29: 241-288.

Pearl, Judea. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Los Altos, CA: Morgan Kaufmann.

Pegah, M., J. Sticklen, and W.E. Bond. 1993. Representing and Reasoning about the Fuel System of the McDonnell Douglas F/A-18 from a Functional Perspective. *IEEE Expert* 8, no. 2: 65-71.

Ramoni, Marco and Alberto Riva. 1993. Belief Maintenance with Probabilistic Logic. In *AAAI Fall Symposium on Automated Deduction in Nonstandard Logics*. Raleigh, NC.

Ramoni, Marco and Alberto Riva. 1994a. Belief Maintenance in Bayesian Networks. In *Tenth Annual Conference on Uncertainty in Artificial Intelligence*. Seattle, WA.

Ramoni, Marco and Alberto Riva. 1994b. Probabilistic Reasoning Under Ignorance. In *Proceedings of the Sixteenth Annual Cognitive Science Society*. Atlanta, GA.

Ramoni, Marco, Alberto Riva, Mario Stefanelli, and Vimla L. Patel. 1994. Forecasting Glucose Concentrations in Diabetic Patients Using Ignorant Belief Networks. In *AAAI Spring Symposium on Artificial Intelligence in Medicine*. Stanford, CA.

Reiser, Brian J., William A. Copen, Michael Ranney, Adnan Hamid, and Daniel Y. Kimberg. 1994. Cognitive and Motivational Consequences of Tutoring and Discovery Learning. Technical 54. Institute for the Learning Sciences, Northwestern University.

Russell, Stuart J. and Peter Norvig. 1995. *Artificial Intelligence: A Modern Approach*. Edited by Stuart J. Russell and Peter Norvig. Prentice Hall Series in Artificial Intelligence. Upper Saddle River, New Jersey: Prentice Hall.

Schank, Roger C. 1986. *Explanation Patterns: Understanding Mechanically and Creatively*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Sembugamoorthy, V. and B. Chandrasekaran. 1986. Functional Representation of Devices and Compilation of Diagnostic Problem-Solving Systems. In *Experience, Memory, and Reasoning*, ed. Janet Kolodner and Christopher Riesbeck:47-73: Lawrence Erlbaum Associates.

Spiegelhalter, David, ed. 1986. *Probabilistic Reasoning in Predictive Expert Systems*. Edited by L. N. Kanal and J. F. Lemmer. Uncertainty in Artificial Intelligence. Amsterdam, London, New York: Elsevier/North-Holland.

Stallman, Richard M. and Gerald J. Sussman. 1977. Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis. *Artificial Intelligence* 9, no. 2: 135-196.

Stroulia, Eleni and Ashok Goel. 1992. Generic Teleological Mechanisms and Their Use in Case Adaptation. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*:319-324. Bloomington, IN.

Sussman, Gerald J. and Guy L. Steele. 1980. "CONSTRAINTS--A Language for Expressing Almost-Hierarchical Descriptions". *Artificial Intelligence* 14, no. 1: 1-39.

Thadani, Sunil. 1994. Constructing Functional Models of a Device from its Structural Description. Ph.D. dissertation, The Ohio State University.

Tomiyama, Tetsuo, Y. Umeda, and T. Kiriyama. 1994. A Framework for Knowledge Intensive Engineering. In *Fourth International Workshop in Computer Aided Systems Theory (CAST-94)*, ed. T.I. Oren and G.J. Klir. Ottawa, Canada.

Tversky, Amos and D. Kahneman. 1982. Causal schemata in judgements involving uncertainty. In *Judgement Under Uncertainty: Heuristics and Biases*, ed. D. Kahneman, P. Slovic, and A. Tversky. Cambridge: Cambridge University Press.

Umeda, Yasushi, H. Takeda, Tetsuo Tomiyama, and H. Yoshikawa. 1990. Function, Behavior, and Structure. In *Applications of Artificial Intelligence in Engineering*, ed. J. Gero, 1:177-193. Berlin: Springer-Verlag.

Van Wylen, Gordon J. , Richard E. Sonntag, and Claus Borgnakke. 1994. *Fundamentals of Classical Thermodynamics*. New York: John Wiley & Sons, Inc.

Vescovi, Marco, Yumi Iwasaki, Richard Fikes, and B. Chandrasekaran. 1993. CFRL: A Language for Specifying the Causal Functionality of Engineered Devices. In *Eleventh National Conference on Artificial Intelligence*. Washington, DC: AAAI Press.