ELSEVIER

Artificial Intelligence 113 (1999) 149–202

Artificial Intelligence

www.elsevier.com/locate/artint

Topological inference of teleology: Deriving function from structure via evidential reasoning [☆]

John Otis Everett [1]

Palo Alto Research Center, Xerox Corporation, 3333 Coyote Hill Road, Palo Alto, CA 94304, USA

Received 19 July 1998; received in revised form 10 June 1999

Abstract

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 article 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. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Qualitative reasoning; Functional reasoning; Plausible reasoning; Engineering thermodynamics; Artificial intelligence; Education

[☆] This research was supported by grants from the Mathematical, Computer, and Information Sciences Division of the Office of Naval Research and the Applications of Advanced Technologies Program of the Directorate for Education and Human Resources of the National Science Foundation.

[1] Email: jeverett@parc.xerox.com.

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 article 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 [22].

1.1. Context

This research builds on Forbus' work on qualitative reasoning about the behavior of physical systems [14]. 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 [9] 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., [7,51], 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 [48] to filter out incorrect functional role inferences. Unfortunately, topological constraints are insufficiently binding, so the net result of such a search is a set of equally likely construals with no principled means of preferring one to another. In response to this, we developed an 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 [50], which used multiple overlapping local viewpoints to analyze electrical circuits. Davis [8] has also noted that multiple representations of locality can provide inferential leverage in diagnostic applications.

The evidential reasoning algorithm employs Pearl's [37] method for Bayesian hierarchical updating of belief. We deliberately use the simplest possible instantiation of Bayesian theory that affords sufficient inferential power without doing violence to the integrity of the formalism. 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 [17].

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. [43]).

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., [2,4,5], we do not require 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).

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 consider 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. Many engineering students choose the profession 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 for a TNT-based system to infer this, it would have to have in its knowledge base the fact that *conductor* is a viable function for a crowbar device. Explicitly representing all potential uses of a device as roles would be impractical; a better approach would be to incorporate more general world knowledge and infer device applicability to a given situation. Fortunately, our approach provides broad coverage of our task domain—coaching thermodynamics students—without such a capability and its attendant complexity.

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 an engineering, as opposed to a cognitive, theory of the teleology of physical domains. We characterize TNT in this way 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 up on the knowledge acquisition and maintenance dimension. 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 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, ranging in size from four to forty-eight components. [1]

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 to help 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 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 a machine, 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.

[1] To place this in context, engineering students virtually never encounter cycles comprised of more than twenty components.

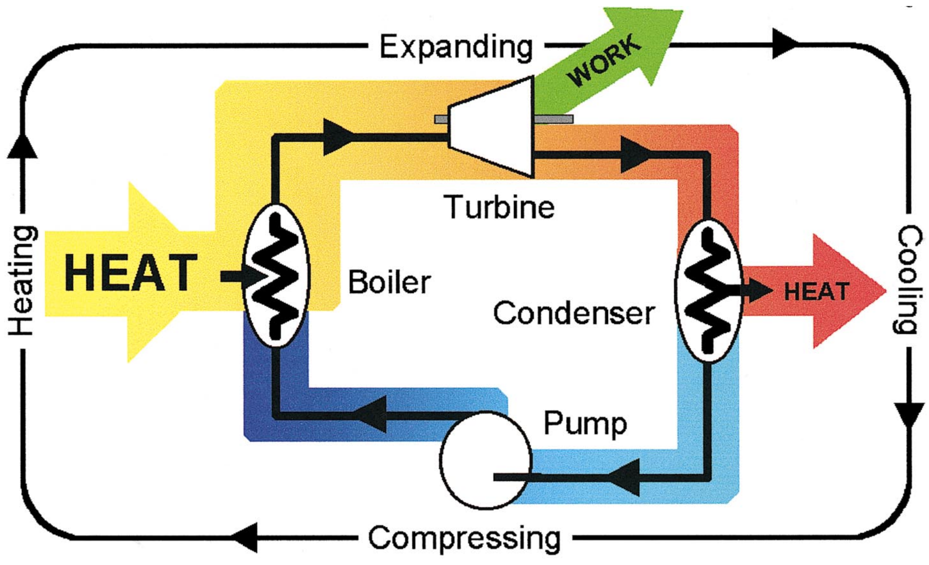


Fig. 1. Simple vapor-cycle heat engine.

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 may be analyzed as cycles by modeling 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[2] through a system of pipes and components. Fig. 1 shows a simple cycle. This basic cycle (with 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

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

the environment. The pump ensures a steady supply of working fluid to the boiler and maintains the system's direction of flow.

Note that the system must eject heat (via the cooler in this case). 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.

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.

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 includes 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 [17]. 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 since 1995, 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. [3]

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

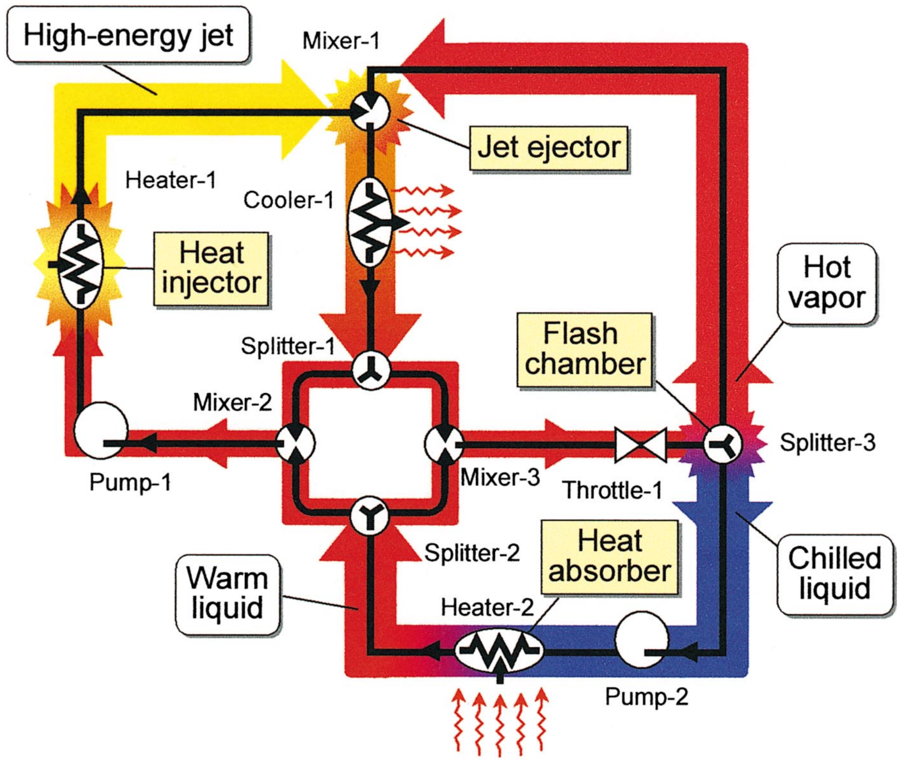


Fig. 2. A jet-ejection refrigerator.

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 [54]. This cycle, shown in Fig. 2, is a jet-ejection air-conditioning system.

In this system, chilled liquid flowing through Heater-2 (at the bottom of the diagram) absorbs heat from the area being air-conditioned (for example, the passenger compartments of a train). This working fluid flow is then split at Splitter-2, 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. This jet flows at high velocity through Mixer-1, and in the process entrains the hot vapor from Splitter-3. The geometry of Mixer-1 then decelerates this jet, which increases the pressure of the working fluid. Mixer-1 is therefore acting as a compressor which has no moving parts, commonly termed a *jet-ejector*.

Splitter-3 splits the working fluid into a hot vapor stream and a chilled liquid stream by virtue of its internal geometry and the pressure drop across Throttle-1 immediately upstream. This pressure drop causes part of the working fluid to boil, or “flash”, into vapor, hence this device is acting as a flash chamber. This flashing requires lots of energy, which comes from the working fluid itself. Therefore only part of the working fluid flashes into vapor, while the rest becomes chilled liquid, which flows to Heater-2 to create the air-conditioning effect. Note that we always use the terms "heater" and "cooler" to refer to the effect on the working fluid, which sometimes has unintuitive side-effects, as in this cycle where a "heater" is providing air-conditioning.

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.

Notice that this system has two structurally identical heaters performing completely different functions, three mixers, two acting as flow joins (Mixers 2 and 3) and one as a compressor, and three splitters, two acting as flow-splits (Splitters 1 and 2) and one acting as a flash-chamber. This ambiguity in function is precisely what TNT is designed to resolve, which it does by reasoning about patterns in the connectivity of the system.

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 Fig. 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 re-
    frigeration 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 be-
    cause 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. The answer is couched in terms of likelihood because TNT is a plausible reasoning system. In this case, the explanation is that refrigerators are more likely to require heat-absorbers. The user might well be dissatisfied with such an answer, perhaps believing that Heater-2 is preheating the working fluid, 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, because 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 com-
    press the working fluid. This takes place at MXR1.
    Jet ejectors have no moving parts and hence vapor re-
    frigeration 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) Change: Achieving a change in the environment, either by moving heat or producing work.

(2) Efficiency: Achieving that change with minimal input of resources.
(3) Preservation: 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 ab-
    sorbs 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 Heater-1.
    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 go-
    ing 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. Roles are central to this theory 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. 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 [45]. 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 a representation language and a reasoning mechanism. The representation language, which is the subject of this section, provides us with the ability to define localities within the topology from multiple points of view, and the reasoning mechanism, discussed in Section 4, enables the dynamic refinement of these representations as it propagates to quiescence.

The representations that TNT defines are shown in Fig. 3. 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 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 design goals at the right of the figure are those that we described in Section 2 (achieve a change in the environment, do it efficiently, preserve the system), and provide the context for the explanation generated. 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 directly achieve goals and may also associate a particular goal with the role of a particular device.

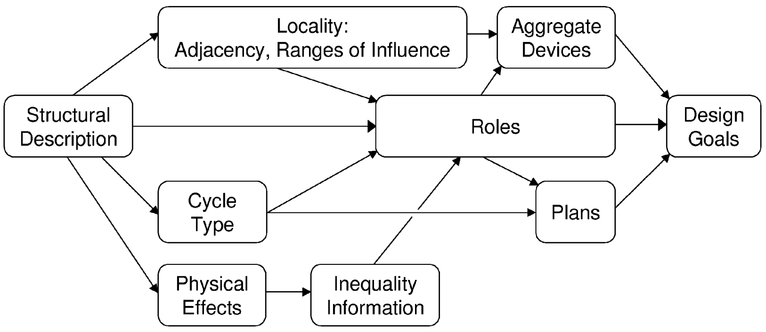


Fig. 3. TNT representations.

In the rest of this section, we will describe each of the representations shown in Fig. 3, with the exception of the identification of cycle type, which is discussed in Section 4. Our discussion will follow the left-to-right organization of Fig. 3.

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. For example, a turbine is represented as (turbine turbine-1 s20 s30), where turbine identifies the type of the component, turbine-1 is the identifier of the turbine instance, s20 is the instance of the stuff at the inlet port, and s30 is the instance of the stuff at the outlet port. This instance will share these stuffs with its immediate neighbors, so for example a heater immediately upstream would be represented as (heater heater-1 s10 s20).

This representation balances the need for expressiveness against computational tractability. An important restriction of this representation is the fixing of the number of inlet and outlet ports for each device type. 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 Fig. 4, 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* [22].

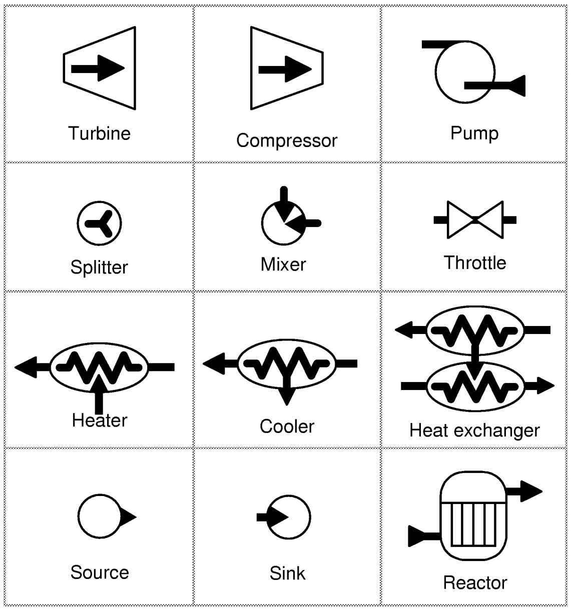


Fig. 4. Schematic device representations.

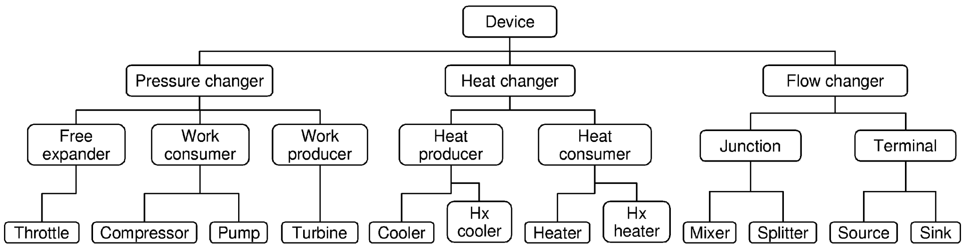


Fig. 5. Device type hierarchy.

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.

However, one of the primary pedagogical goals of CyclePad is to induce the student to reflect on the thermodynamic processes taking place in the cycle at hand. Presenting a limited palette of components requires 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, requiring the student to use a mixer to model a jet-ejector requires some thought about the underlying processes that would make such a model 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 Fig. 5.

Each component is 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 Fig. 6. Note that roles may be hierarchical. 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 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. We term these components hx-heaters and hx-coolers.

3.3. Physical effects

As we saw in the example in Section 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 [14] distinguishes between processes,

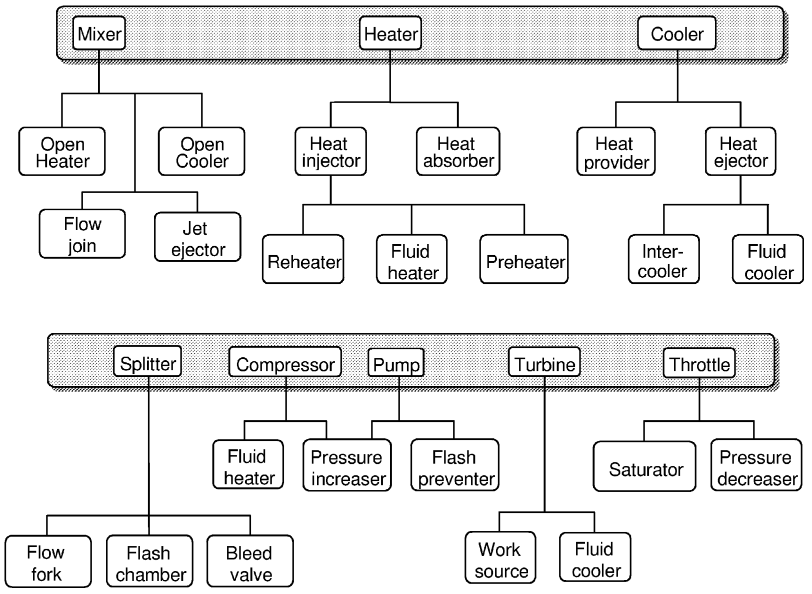


Fig. 6. Role taxonomy.

such as heat-flow, and their 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 turn, side-effects may be beneficial, neutral, or detrimental with respect to the three design goals. For example, in a heater acting as a heat-absorber, the temperature of the working fluid (so long as it is subcooled) 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.

Because TNT is a theory of reasoning directly from structure to function, 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 are used mostly to detect errors in inference.

3.4. Domain requirements

In addition to reasoning about physical effects, TNT also makes inferences at a global level, reasoning by exclusion. 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 inference, we will use the cycle shown in Fig. 7, which is a relatively simple regenerative Rankine heat-engine. In this

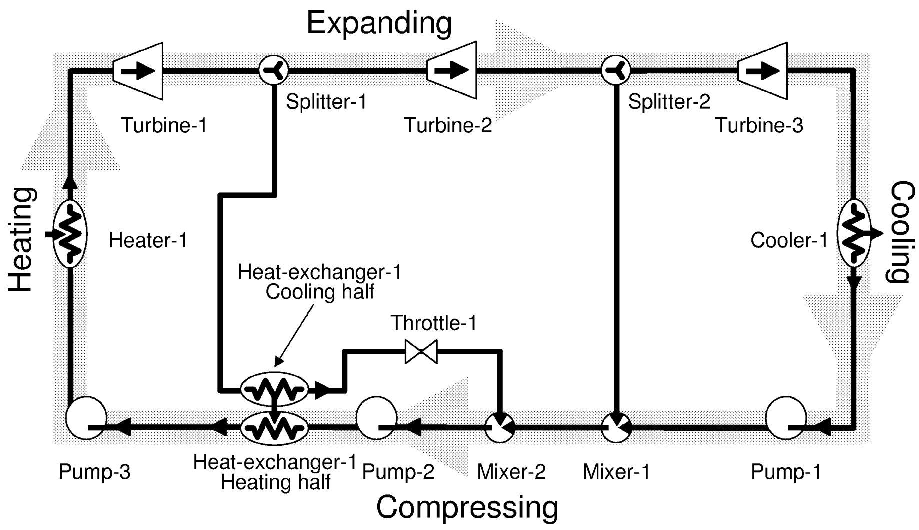


Fig. 7. Regenerative Rankine power cycle.

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 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 (efficiently achieving a change in the environment). 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 Section 2.1.

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.[4] 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 functional 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, flow-fork, or bleed-valve. This is an instance of the tight interleaving between the locality and role representations that occurs during the inference process.

In Fig. 8 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.

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; 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, because the heater is the last one upstream of the cycle's turbines.

[4] An intensive property, such as temperature, remains constant when one changes the amount of a substance.

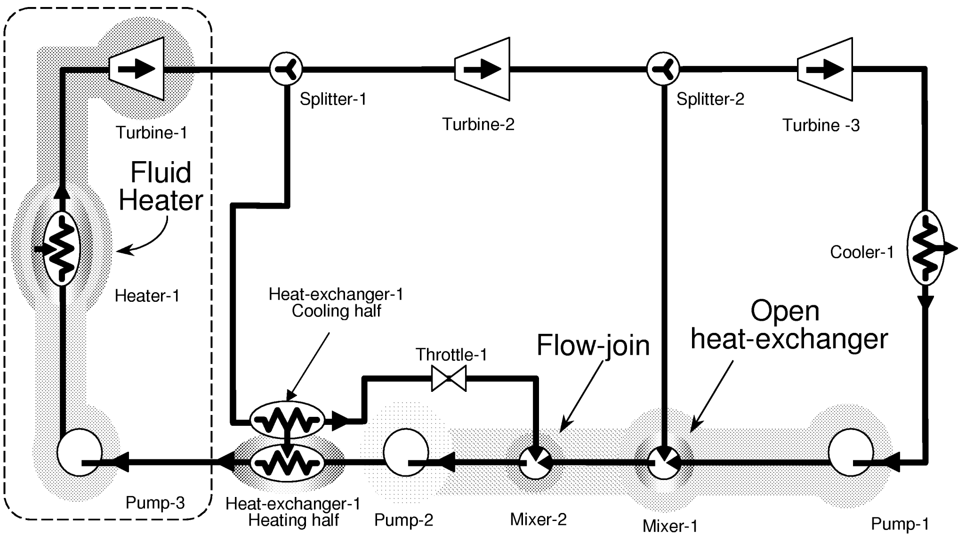


Fig. 8. Adjacency.

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. Ranges of influence are defined to extend downstream from the component; the downrange relation represents these ranges. Ranges may include splitters, as in the case of Turbine-2 in Fig. 9.

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 Fig. 9. 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 Heat-exchanger-1-heater would cause Pump-3 to stall, because compressing 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.

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 Heat-exchanger-1-heater and Pump-3 (in an arrangement known as *feed-forward*), rather than in its position upstream of Pump-2. The downrange relation

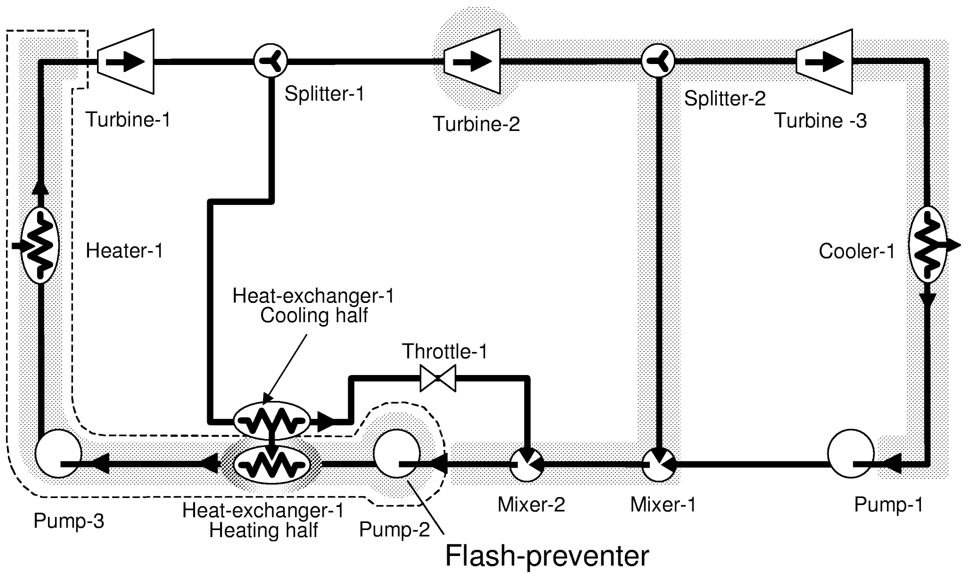


Fig. 9. Ranges of influence.

provides us with a precise yet flexibly defined locality in which to look for a heater, then a pump, then another heater, and finally a turbine.

Teleological patterns: Certain configurations of devices that are playing particular roles occur with sufficient frequency that we gain leverage by representing them explicitly. In 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, increasing efficiency. In Fig. 10 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. 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 complete and self-contained fluid systems. By this we mean that within a subcycle, a piece of working fluid can move from an arbitrary location to any other point in the subcycle. Most systems are comprised of a single subcycle, but a system can contain multiple subcycles. In such cases, the subcycles are connected solely via the heat-path of a closed heat-exchanger, so no fluid-path can exist between two subcycles. For example, nuclear plants often contain a fluid loop that passes through the reactor core and then exchanges the thermal energy acquired via closed heat exchangers. This design isolates the reactor core and reduces the changes of radiation leakage.

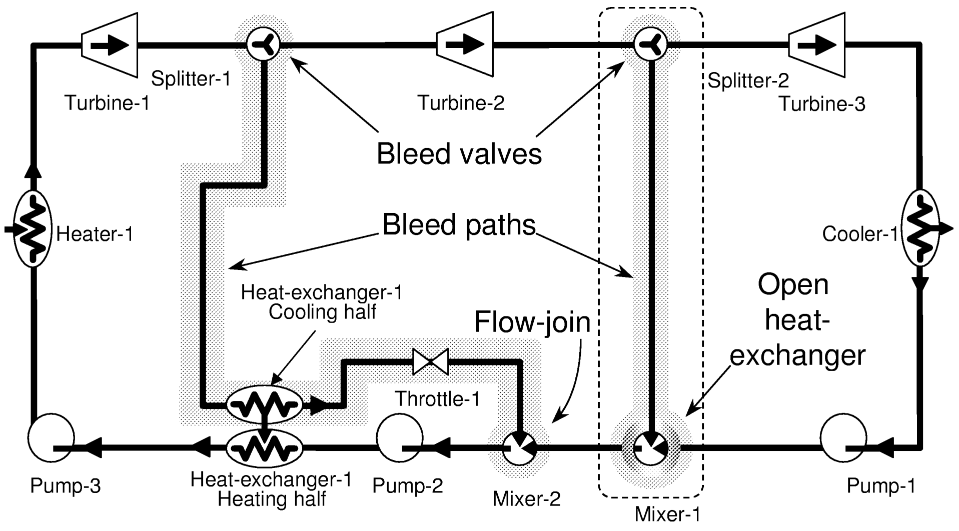


Fig. 10. Examples of teleological patterns.

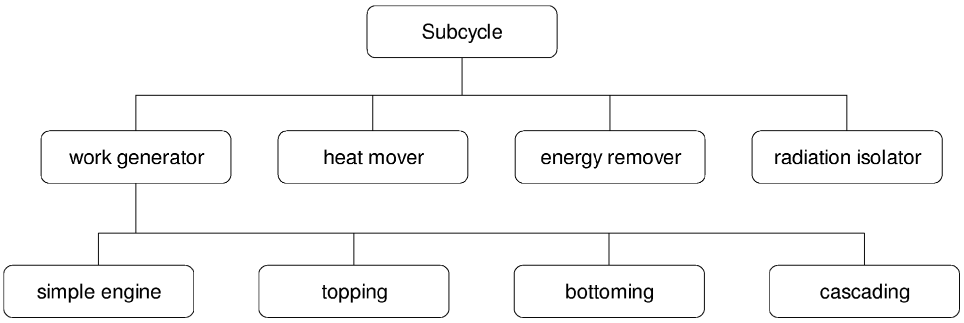


Fig. 11. Subcycle roles.

Due to their generality, subcycles, unlike bleed-paths, may play several different roles and may also be the primary participants in plans. Fig. 11 illustrates the roles a subcycle may play.

A subcycle acting as a work-generator must contain a heater, a turbine, a cooler, and a compression device, and constitutes a heat-engine. A subcycle acting as an energy-remover moves heat from area to another (and hence acts as a refrigerator or heat-pump). These are the most common roles for subcycles.

Work-generators are further specialized into simple-engines, topping, bottoming, and cascading subcycles. A topping subcycle is one whose 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.

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 Fig. 12 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 than one aggregate device of a given type.

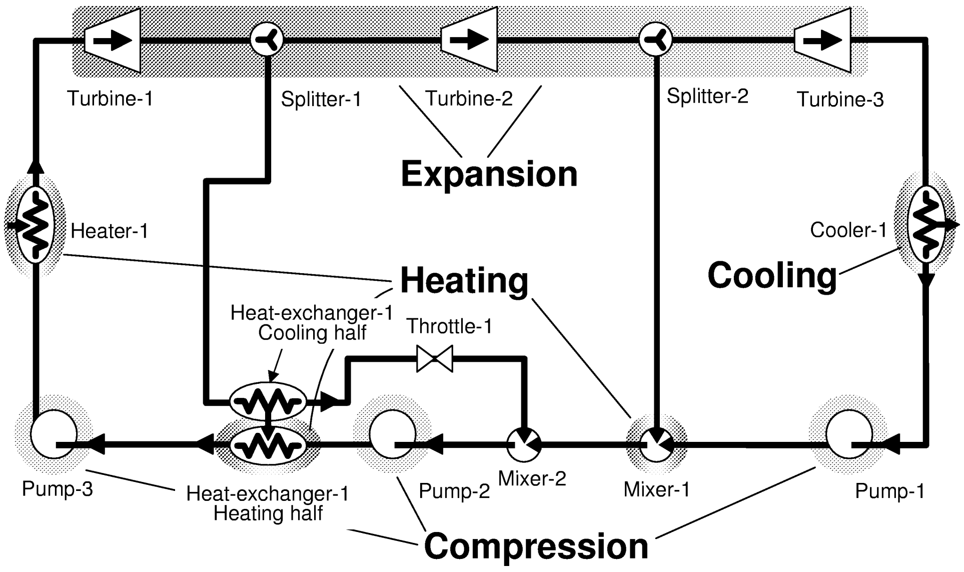


Fig. 12. Aggregate devices.

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 are represented as a mixer connected at its outlet to the inlet of a splitter, and are used to separate steam from boiling water. In effect, 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 (typically to recirculate back into the steam drum) 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, fouling and damage to the turbine blades may occur. Flash-tanks provide the reciprocal function in refrigeration systems, cooling a hot gas by bubbling it through a cool liquid.

3.6. Plans

Whereas roles represent inferences about function at the component level, *plans* represent functional inferences at the system level. Plans are abstracted patterns of function that 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 Section 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. Fig. 13 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. This plan is a specialization

```
(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)))))
```

Fig. 13. Example of a plan.

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 comment in quotes starting on the second line and the `:NL-phrase` provide the raw material for the generation of explanations. The key roles are those roles that are central to the plan, and are distinguished from conditions to provide greater focus in explanations. In this case, the open-heater role is where the actual regeneration takes place, so this is the key role for this plan. The conditions must be true prior to plan instantiation. Here they specify that this plan is only active when the bleed-path that feeds the mixer contains a closed heat-exchanger acting as an intercooler.

4. Reasoning

TNT infers function from structure in three steps, as shown in Fig. 14. 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 section provides a detailed description of these three steps.

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 Fig. 15. Of these steps, fluid-loop identification is by far the most involved.

```
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
```

Fig. 14. Reasoning algorithm.

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

Fig. 15. Topological analysis algorithm.

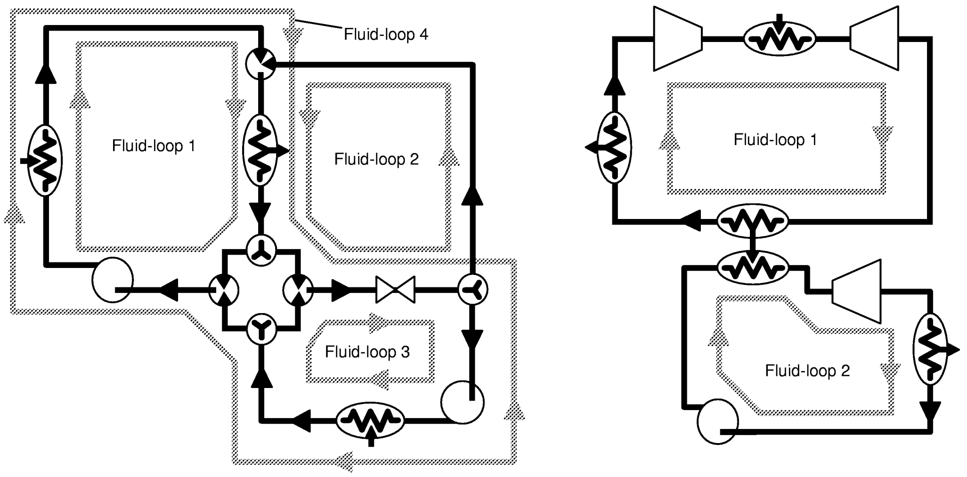


Fig. 16. Examples of fluid loops.

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. 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 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; Fig. 16 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 Fig. 17, 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 comprise the set of devices that complicate topologies; without them our representation constrains the topology to a single simple loop consisting of

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

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

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)
```

Fig. 17. Algorithm for finding fluid-loops.

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 find-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 find-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,

Table 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

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 Fig. 16 contains two subcycles. The salient feature of this and any cycle that contains multiple subcycles is 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. Therefore the system on the left of Fig. 16 has only one subcycle (despite having several fluid loops), since a given bit of working fluid can travel from one arbitrary point in the system to any other.

The algorithm for the identification of subcycles is shown in Fig. 19. We iterate over all identified fluid-loops, merging all that share components.

4.1.4. Finding downstream ranges of influence

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. Table 1 shows the criterion used for ending the range of each type of device. Ranges of influence break the cycle into groups that correspond to the four canonical cyclic processes (compression, heating, expansion, and cooling). Fig. 18 shows the algorithm.

4.1.5. Determining cycle type

Once all fluid-loops have been found, deriving the cycle-type is a global operation 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 neither need nor use for refrigeration loops.

The cycle-type-finding algorithm first 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

```
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))
```

Fig. 18. Algorithm for finding ranges of influence.

```
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
```

Fig. 19. Algorithm for finding subcycles.

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 label the fluid-loop within each subcycle that contains the greatest number of turbines as the *primary fluid-loop* for that subcycle, that is, the fluid-loop that would produce the greatest power. This fluid-loop is then used to identify bleed paths, which originate at splitters interleaved among its turbines (and hence playing the role of bleed-valves) and terminate in mixers that are also primary-fluid-loop members.

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 Section 7, and focus in this section 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 [37], which in this case is the hierarchy of roles described in Section 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.

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

$$\lambda_h = \frac{P(e|H)}{P(e|\neg H)} \tag{4.1}$$

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 $[1, \infty]$, whereas the range of suppression is $[0, 1]$. We considered using log-likelihoods to make these ranges symmetric, but decided against this approach on the grounds that domain experts would be more comfortable (and potentially more accurate) estimating a likelihood on a linear scale. Instead, we enable users to write suppressive evidence in the form (not *role-tested-for*). This form converts likelihoods provided in the range of $[1, \infty]$ into their inverses. For example, entering evidence against a work-generator role with likelihood 10 results in an evidential statement with likelihood of 0.1 internally.

4.2.1. 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 is found to be true. Success causes the instantiation of an evidence proposition, which is used to update the prior probability of the role.

An evidential test form is shown in Fig. 20. 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 role tested for is `flash-preventer`, and the likelihood that this role is present given the evidence in this form is `6.0`. The `(pump ?pmp1 ?in ?out)` statement is the initial trigger, which activates this evidential test when there are pumps present in the system. The comment in quotes both documents the knowledge base and provides the basis for explanations predicated on this evidence. The balance of the form consists of propositions whose conjunction constitutes evidence in favor of a `flash-preventer` role. The `:test` keyword enables one to specify additional relationships among these propositions. In this case, the first test form specifies that, should we find a mixer playing the role of an open heat-exchanger, that mixer must also be a member of the pump's range of influence.

An evidential test instantiates an evidence proposition of the form (evidence ⟨*device-name*⟩ ⟨*device-role*⟩ ⟨*likelihood*⟩ ⟨*test-name*⟩) 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 remaining true. Because role propositions

```
(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))))
```

Fig. 20. Example of an evidential test.

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

Fig. 21. Summary of algorithm for role inference.

are permitted as conditions in tests (as in Fig. 20), it is quite possible that an evidence proposition asserted with a 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 Fig. 3.

4.2.2. 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 Fig. 21. Local propagation will cause any evidential test whose conditions are met by facts known to be true to instantiate an evidence proposition. 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 roughly 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.

Once local propagation has reached quiescence, we either create or update a proposition of the form (evidence-for ⟨*device-name*⟩ (⟨*evidence-proposition*$_1$⟩ ⟨*evidence-proposition*$_2$⟩

```
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
```

Fig. 22. Algorithm for role inference.

...)) by iterating over all devices and collecting all evidence propositions believed to be true for each device. Any difference between this set and the list in the evidence-for proposition causes an update of the evidence-for proposition.

When this iteration is complete, we assume the most likely role for each device is 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 existing evidence sets, we must repeat the above iteration. This algorithm typically reaches quiescence in from one to twelve iterations. Although it is possible for a given knowledge base to fail to reach quiescence, it is also 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 Fig. 22.

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 [37]. 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. 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 re-propagate 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 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 and identify 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 turns out to be quite hard to deliberately construct such a loop with evidential tests that have at least the surface appearance of reasonableness.

4.2.3. Example 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 system's boiler. 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 Fig. 20 is shown in Fig. 23, 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

```
(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))))
```

Fig. 23. Example of a test for flash-preventer evidence.

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 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.

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.

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 correctly identifies the function of forty-nine thermodynamic cycles, using a knowledge base containing 107 evidential tests, as shown in Table 2. We estimate that this version of CARNOT has achieved at least 90% coverage of the domain of single-working fluid thermodynamic cycles. Of the forty-nine cycles CARNOT has analyzed, thirty-two are from *Analysis of Engineering Cycles* [22], and comprise all single-substance cycles presented in this text, which is considered to be a standard for this field. The other seventeen cycles are drawn from other introductory thermodynamics texts.

In this section we present empirical evidence of CARNOT's performance, and examine certain other performance characteristics, including the sensitivity of the algorithm to the specific values of the likelihoods in the knowledge base.

Table 2
Summary of CARNOT tests for and against particular roles

Device	Role	Pro	Con	Device	Role	Pro	Con
Cooler	Fluid-cooler	1	1	Pump	Flash-preventer	2	3
	Heat-ejector	2	1		Flow-producer	2	0
	Heat-provider	4	0		Total	4	3
	Intercooler	1	0	Splitter	Bleed-valve	5	1
	Total	8	2		Flash-chamber	9	3
Heater	Fluid-heater	5	0		Flow-fork	2	0
	Heat-absorber	4	0		Total	16	4
	Heat-injector	5	0	Subcycle	Bottoming	1	0
	Preheater	4	5		Energy-remover	2	0
	Reheater	2	2		Heat-mover	2	0
	Total	20	7		Radiation-isolator	1	0
Heat-exchanger	Fluid-cooler	1	0		Simple-engine	1	0
	Heat-absorber	1	0		Topping	1	0
	Total	2	0		Work-generator	1	0
Mixer	Flow-join	3	1		Total	9	0
	Jet-ejector	4	2	Throttle	Pressure-decreaser	3	0
	Mxr-heater	6	0		Saturator	5	0
	Mxr-cooler	3	0		Total	8	0
	Total	16	3	Turbine	Fluid-cooler	3	0
					Work-source	2	0
Grand Total (107 tests)		*88*	*19*		Total	5	0

5.1. 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. One of our goals in the design of TNT was to create a general account of teleological knowledge that would extend gracefully to unanticipated inputs. To test CARNOT's performance on this dimension, we constructed a validation set of ten cycles. We excluded from this set cycles that were minor variants of those in the primary test set, so as not to inflate the validation results. For example, the addition of an arbitrary number of turbine stages to a regenerative cycle (such as the one shown in Fig. 12) would not impact CARNOT's ability to identify the relevant roles. Therefore we admitted to the validation set only cycles that could not be rendered isomorphic to any cycle in the primary test set via some application of CARNOT's topological transformations. Table 3 shows the results of running CARNOT on this test set.

Using the knowledge base developed based on the primary test set, CARNOT correctly identified on average 94.3% of the roles in the ten cycles that comprise the validation test set. To address the shortcomings in this knowledge base uncovered during the validation test run, we had to add five new tests (two of them suppressive of incorrect roles), and modify two other tests to make their trigger conditions more specific. After these changes were made, CARNOT correctly identified all roles in all cycles comprising the validation test set.

Table 3
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

The changes required to successfully construe the validation cycles amounted to less than 7% of the knowledge base. Based on these results, we believe that representation of functional knowledge in terms of tests for evidence is sufficiently general to enable the construction of 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. At this point, CARNOT already construes a far wider range of cycles than most students would ever encounter in an engineering curriculum.

5.2. 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 Fig. 24. The time in seconds on the vertical axis is the result of running on a 400 MHz Pentium II processor with 256 MB of RAM. Although the largest cycle required about forty-five seconds for a solution, notice that cycles with twenty devices (which are still large from a pedagogical perspective) require around eight 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 ten seconds for the system to 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 without perceptible delay.

An analysis of the algorithmic complexity confirms these empirical findings. (See [12] for details). To summarize, the most computationally intensive parts of TNT are fluid-loop identification and role inference, and these are worst-case $O(N^3)$ where N is the number of devices comprising the input cycle.

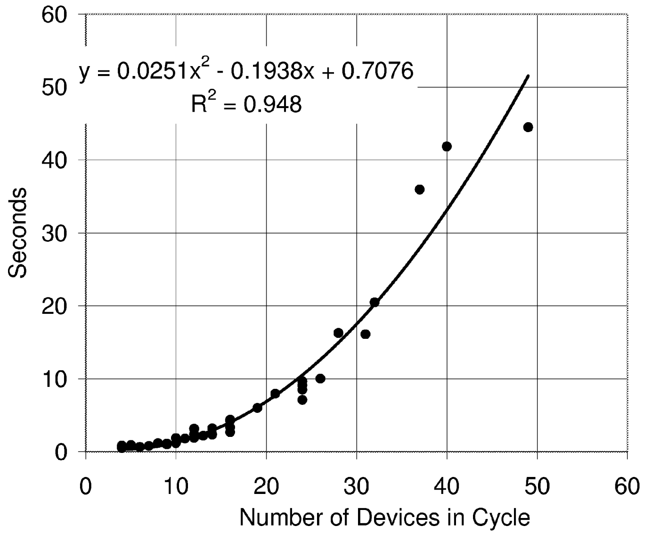


Fig. 24. Empirical performance for CARNOT on Test Set A.

5.3. Characteristics of the role inference algorithm

As we saw in Section 4.2.2, 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 Fig. 22) 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.

Fig. 25 shows the number of iterations required to infer the roles of each of the thirty-six cycles in Test Set A. The light bars indicate the number of iterations required if we sort the input to update-evidence-and-roles in increasing order of ambiguity, and the dark bars show the additional iterations required if we sort the input in decreasing order of ambiguity. The absence of a dark bar indicates no difference in the number of iterations.

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

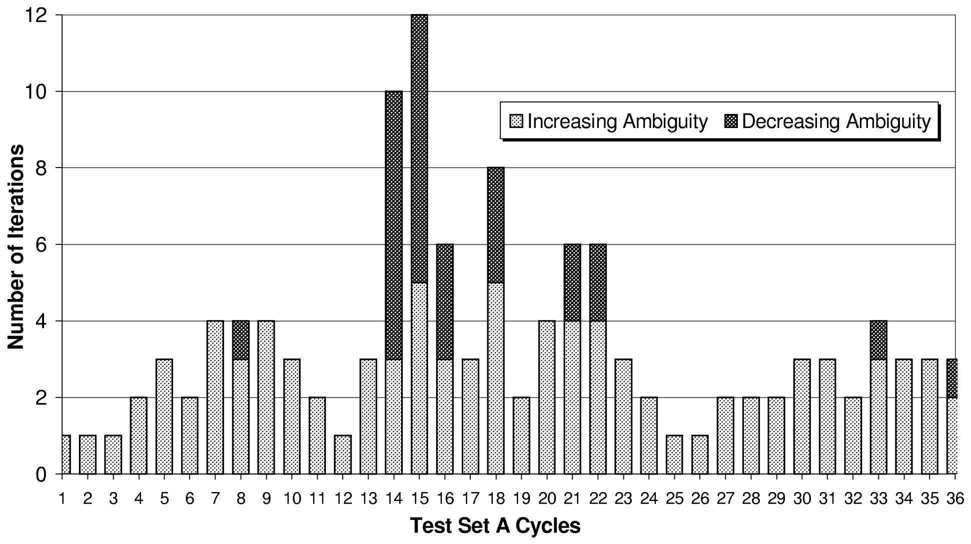


Fig. 25. Effect of device order on number of iterations to reach quiescence.

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 both utilize nuclear reactors, but so do 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.

5.4. *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. 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

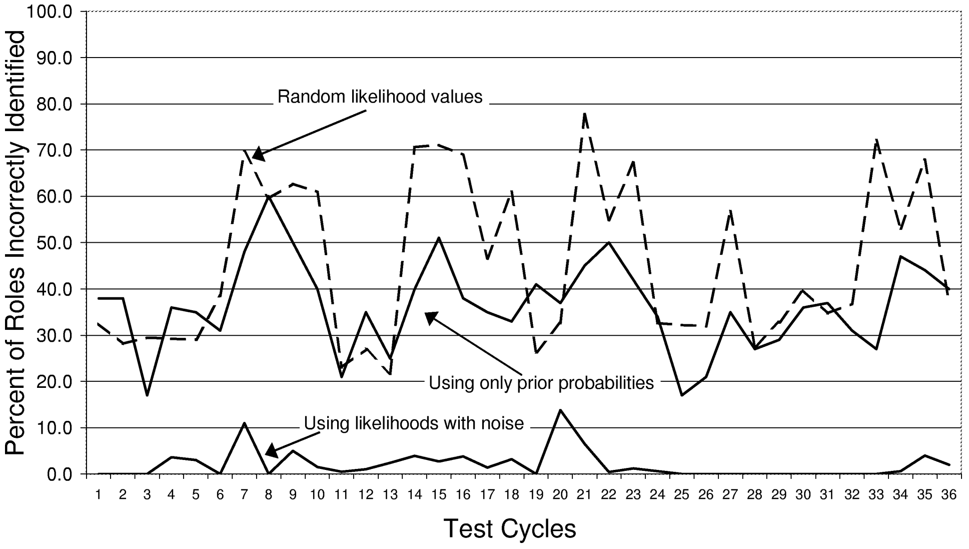


Fig. 26. Effect of introducing noise into likelihood estimates.

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 robust to noise, exhibiting only minor degradation at 100% noise, as shown in the lowest line of Fig. 26. 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 Fig. 26. 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 Fig. 26.

6. CARNOT-based coaching applications

We have characterized TNT as an engineering 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. [5]

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.

The public version of CyclePad incorporates TNT in two subsystems, a Role Annotator and an Analytical Coach. We view each of these subsystems, which we describe below, as prototypes for further research into how students can most effectively use such software. 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

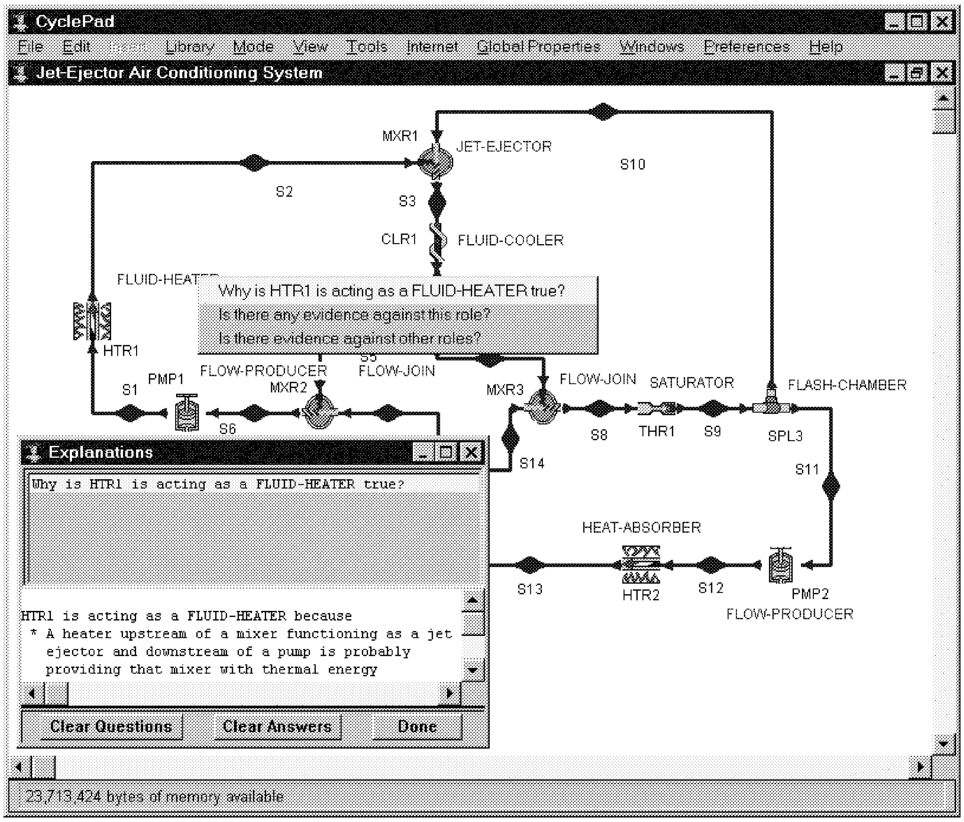
The Role Annotator enables students to see the roles that CARNOT has inferred for each device, and to explore the reasoning underlying those inferences. It displays the role inferred for each device as a mouse-sensitive annotation next to the device; clicking on the annotation opens the hypertext system, initially presenting the user with a choice of three questions, as shown in Fig. 27.

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 important for the development of critical thinking. The hypertext system provides the student with the information necessary to form an opinion of the role inference's accuracy. In Fig. 27 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

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


CyclePad
File Edit Insert Library Mode View Tools Internet Global Properties Windows Preferences Help
Jet-Ejector Air Conditioning System
MXR1
JET-EJECTOR
S10
S2
S3
CLR1
FLUID-COOLER
FLUID-HEATER
Why is HTR1 is acting as a FLUID-HEATER true?
Is there any evidence against this role?
Is there evidence against other roles?
HTR1
S1
PMP1
FLOW-PRODUCER
MXR2
S5
FLOW-JOIN
MXR3
FLOW-JOIN
SATURATOR
FLASH-CHAMBER
S6
S8
THR1
S9
SPL3
S14
S11
HEAT-ABSORBER
S13
HTR2
S12
PMP2
FLOW-PRODUCER
Explanations
Clear Questions
Clear Answers
Done

```
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
```

```
23,713,424 bytes of memory available
```

Fig. 27. Role annotations and hypertext explanations in CyclePad.

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, engineers often make use of ideal cycles, such as the Carnot cycle, which disregard practical issues, such as the compression of saturated fluids.

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. 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 in an inefficient transfer of heat. 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.

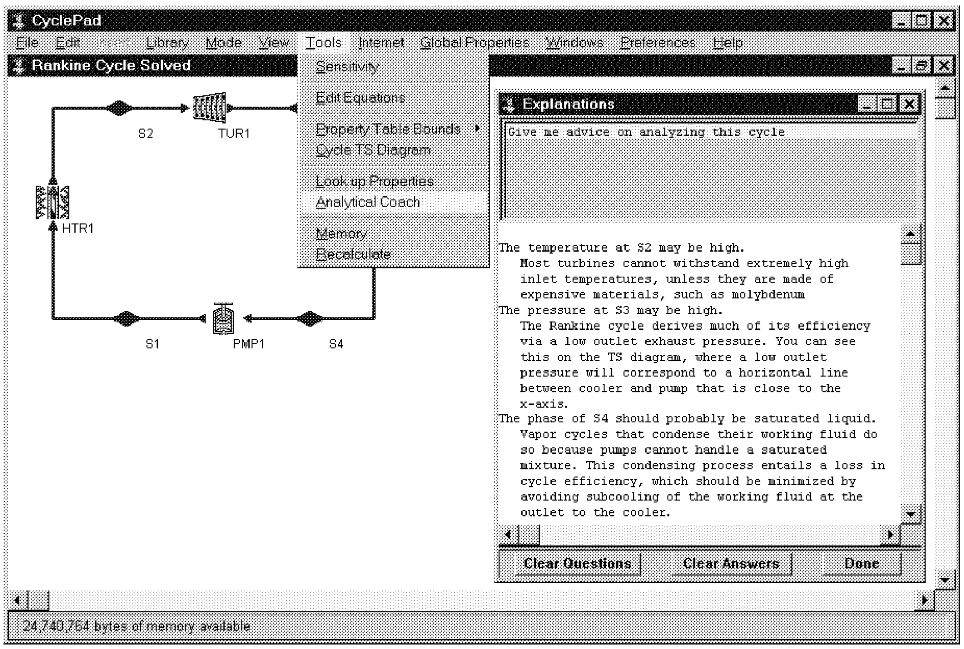
Fig. 28 illustrates the norm that one should avoid subcooling the working fluid of a vapor-power cycle. Norms are defined with reference to roles and, as shown in this case, potentially to plans, such as the vapor-power cycle plan.

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. 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.

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. Those norms whose preconditions are present in the database are instantiated and checked against the cycle's parametric data for potential violations. Should the parametric data indicate that the working fluid at this

```
(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)))))
```

Fig. 28. An example of an analytical coaching norm.



```
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.
```

Fig. 29. Analytical coaching output in CyclePad.

point is not a saturated liquid, then the Analytical Coach reports this violation to the user. Advice is displayed within the hypertext system, as shown in Fig. 29.

Running over a student-constructed simple vapor power cycle, the Analytical Coach finds three potential inefficiencies. First, the inlet temperature to the turbine is high compared to the norm for vapor-power cycles. Although this may be a deliberate choice on the part of the student, we believe that bringing this to the student's attention helps to ground the parametric values of the cycle 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 is that the pressure at the outlet of the turbine 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. Developing sound intuitions about this diagram and how it relates to system performance is an essential part of understanding thermodynamics.

The third norm violation, also related to the goal of maximizing efficiency, occurs at the inlet to the pump, where the liquid is being cooled more than necessary. This is an instantiation of the norm presented in Fig. 28.

The Analytical Coach currently contains twenty-four norms. Complete coverage of the domain could require as many as 1600 norms, due to the number of potential materials a device may be constructed of, the number of roles it may play, and the number of working fluids available to the student. For example, a turbine constructed of molybdenum will tolerate temperatures as high as 2000 K, whereas a turbine of stainless steel will fail at temperatures above 850 K. Moreover, a turbine functioning as a fluid-cooler will operate in a very different temperature regime than one functioning as a work-producer. Although this doesn't impact phase and qualitative norms, parametric norms for temperature and pressure will be necessary for each working fluid (of which there are effectively eight in CyclePad) and each material (of which there are effectively three). Given thirty-two potential roles in the CARNOT database, there are 1536 parametric norms required.

This simple calculation overstates the number, however, because there are many combinations of roles, working fluids, and materials that would be most unlikely to occur. For example, a jet-ejector requires the working fluid to go through a phase change, and although it's conceivable that some application might utilize air or helium in a jet-ejector, the likelihood of such an application is fairly small. We estimate that the norms required to support reasoning about likely systems total around five hundred.

However, norms are of most use to novices, and the cycles that they are most likely to construct disregard materials and typically utilize water as the working fluid, in the case of heat engines, and refrigerant-12 in the case of refrigerators, so we have focused our efforts in this area. From our experience with CyclePad, we expect that doubling the current set of norms, to about fifty, will provide reasonable coverage of the problems students encounter most frequently.

7. Related work

In this section we first discuss work related to the plausible inference mechanism, and then we consider how the teleological research relates to other work in qualitative reasoning. Although the reasoning algorithm employs Bayesian updating, we do not use Bayesian belief nets (BBNs), so we refer the reader interested in such to [37], and confine the following to more relevant work.

7.1. Plausible inference

TNT's inference mechanism adds an evidential reasoning algorithm to a forward-chaining rule engine that itself is coupled to a logic-based truth maintenance system, or LTMS [16,32,33]. 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 [13]. 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 [10] 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., Kirchhoff'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 [39] 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 [40,41]. Their LBMS uses a variant of Nilsson's probabilistic logic [35], 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 reduce incrementally and monotonically these intervals as more information is added to the database.

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, 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 [42], with promising results, and it does so with a small fraction of the conditionals needed to specify a conventional BBN (2262 versus 19200).

Goldman and Charniak [20] 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 [44,

p. 444]. 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 the rules for their application may be pre-specified.

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 [9]. In his theory, a causal analysis describes in qualitative terms the behavior (via simulation) of a given circuit topology, and a separate teleological analysis determines the purpose of the system by parsing via a grammar of circuit functions.

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 [50].

Whereas the TNT reasoning process is purely topological, disregarding the geometry of the input system, Davis [8] 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" [8, p. 88]. Although TNT does not reason about geometry, its multiple representations of locality are consistent with this insight.

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 connect function to fundamental domain principles. Williams, for example, has developed a theory of interaction-based invention, which takes as input a set of desired interactions and a domain theory consisting of a set of principles characterizing the domain and the available technology [56]. From this he constructs a topology of potential interactions among the quantities defined in the domain theory. Tracing a path through this topology to link all variables participating in the desired interactions produces a candidate solution containing a minimal set of interactions. This candidate set will have a direct mapping to physical structure. A final step instantiates this structure and verifies that it produces the desired interaction. This approach can potentially produce systems that exploit multiple behaviors of a given device, increasing robustness. Whereas TNT starts with a topology of connected devices and infers the intended and unintended behaviors (i.e., interactions), interaction-based invention works in the opposite direction, starting with an abstract interaction topology and working toward a potential physical instantiation.

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, based on the multilevel flow modeling formalism developed by Morten Lind [31]. This formalism, which models systems in terms of energy, matter, and information flows, is of particular interest because it 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 grew out of a dissatisfaction with the opacity of knowledge in the expert systems of 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" [30, p. 48] and "deep" models, which make explicit causal interactions. The goal of this research is to specify a language (FR) for the representation of device function, 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 compiled fragments of causal knowledge that support reasoning about behavior and assumptions enable the reasoning agent to choose among behavioral alternatives.

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" [46, p. 50]. 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 dictates 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.

Vescovi, Iwasaki, Fikes, and Chandrasekaran [55] 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 [55]; 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, where 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) [15] or the Device Modeling Environment (DME) [25].

There is some reason to believe that FR-based systems will scale to tackle problems of practical interest. In particular, Pegah, Sticklen and Bond [38] 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[6] (FM), which differs from FR in that it relies on simulation as a core reasoning strategy.

Other work in Functional Reasoning also bears on this research, the closest in spirit being Thadani's work on device understanding [51]. 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. However, TNT differs from this approach in its organization of the knowledge base around evidential tests, which we believe afford greater maintainability of the knowledge base than does the template-based approach to knowledge organization.

Bylander's method of reasoning by consolidation [6] 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.

Keuneke [27] 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 one of the first large-scale applications of the functional reasoning theory. Note that the semantics of these predicates embodies a lack of change, that is, the preservation of a given state over time. Explicitly representing such maintenance functions considerably enriches the

[6] 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 section.

expressive power of the formalism. 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 system component.

Allemang [1] 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 programming conventions (e.g., object-oriented programming). In a poorly constructed or 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 [3,19,49] 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.

In the Qualitative Reasoning community, Franke [18] 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, however, it requires an envisionment of all possible behaviors of the system, which it achieves via QSIM [28], and also a specific design change.

Narayanan has investigated the inference of behavior from the geometric information contained in diagrams [34]. It is likely that CARNOT could directly act on behavioral evidence generated such a reasoner from the visual depiction of the input, which it 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" [52]. The goal of this research is to make more knowledge about the engineering domain available to engineers during design tasks. They posit a "function behavior state" (FBS) modeler [53] 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. Functional knowledge can be used to improve designs; for example, the system might suggest a particular design strategy to achieve functional redundancy.

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. Lind [30,31] 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 are of material, energy, and information. MFM defines a function as a “useful behavior” [31, p. 19], consistent with our definition. MFM also makes explicit the goals of the system, but 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* [31, p. 16]. In contrast, MFM’s functions 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 [31, p. 44].

Larsson [29] 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, to ensure consistency. Conflict resolution, however, 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 worst-case linear complexity for these algorithms because they only traverse static links.

Such MFM models appear to scale; Larsson reports that a large-scale (544 rules) MFM model for monitoring and diagnosis of post-operative intensive care patients called Guardian [21] exhibits a worst-case execution time for a fault diagnosis of 1100 μs, and executes 500,000 rules per second. Overall, 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 is the output of the process.

8. Discussion

The representations and the inference mechanism of TNT were developed in tandem, with progress in one area informing (and sometimes confounding) the other. In this section we discuss some of the issues that arose during this process.

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 [37] for incrementally propagating evidence.

Representing knowledge in this manner imposes a constraint of conditional independence on each piece of evidence. In other words, each piece of evidence must be independent of all other instantiated evidence, or else its associated likelihood will overstate the probability of the role for which it provides support. 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 and all ten in the validation set, consists of 107 tests, of which no single device and role combination accounts for more than nine tests.

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 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 check 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, this 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.

The plausible inference mechanism in CARNOT replaces a more conventional qualitative reasoning approach, dependency-directed search [48] for a set of intermediate behavioral representations we called *views* [11]. We found dependency-directed search to be extremely inefficient, because it operates by ruling out interpretations. The final result is a set of equally likely construals. The set of construals for a heat-engine with five mixers would contain all the permutations of those devices construed as open heat-exchangers or flow-joins, with no principled means for preferring one construal over another. A domain expert, 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 that produces a single, most probable construal.

Plausible reasoning, however, is also subject to exponential behavior. The obvious example is the joint probability distribution, a simple (but exponential) enumeration of the probability of every possible state of the world in a table. Bayesian belief networks (BBNs) [33] 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.

Secondly, the nodes of BBNs typically represent particular random variables that describe possible partial states of the world. For example, in medical diagnosis, each symptom would be associated with a node taking on the values present/absent, or perhaps absent/mild/severe. Topological patterns, the analog of symptoms in our theory, are not so succinctly described.

Finally, TNT allows for roles to act as evidence for or against other roles. This would result in a multiply-connected network topology, which requires more complex methods for inference, such as clustering [47], cutset conditioning [24,26,36], or stochastic simulation [23].

9. Conclusion

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 that 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 tests 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.

We believe that the pedagogical tools TNT makes possible will enable the construction of new curricula that have design as a central focus and demonstrably improve understanding of domain concepts. And, finally, there are other applications for TNT that we have yet to explore. For example, a TNT-based subsystem might be embedded in a 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.

References

[1] D.T. Allemang, Understanding programs as devices, Department of Computer Science, The University of Ohio, Columbus, OH, 1990.
[2] P. Baffes, R. Mooney, Refinement-based student modeling and automated bug library construction, Journal of Artificial Intelligence in Education 7 (1) (1996) 75–116.
[3] S.R. Bhatta, A. Goel, 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, France, 1993.
[4] J.S. Brown, R.R. Burton, Diagnostic models for procedural bugs in basic mathematical skills, Cognitive Science 2 (1978) 155–192.
[5] J.S. Brown, K. Van Lehn, Repair theory: A generative theory of bugs in procedural skills, Cognitive Science 4 (1980) 379–426.
[6] T. Bylander, B. Chandrasekaran, Understanding behavior using consolidation, in: Proceedings Ninth International Joint Conference on Artificial Intelligence (IJCAI-85), Los Angeles, CA, 1985.
[7] B. Chandrasekaran, Functional representation and causal processes, in: M. Yovits (Ed.), Advances in Computers, Academic Press, New York, 1994.
[8] R. Davis, Diagnosis via causal reasoning: Paths of interaction and the locality principle, in: Proceedings of the National Conference on Artificial Intelligence (AAAI-83), Washington, DC, Morgan Kaufmann, San Mateo, CA, 1983.
[9] J. de Kleer, Causal and teleological reasoning in circuit recognition, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Boston, MA, 1979.
[10] J. de Kleer, B. Williams, Diagnosing multiple faults, Artificial Intelligence 32 (1987) 97–130.
[11] J.O. Everett, A theory of mapping from structure to function applied to engineering domains, in: Proceedings International Joint Conference on Artificial Intelligence (IJCAI-95), Montreal, Quebec, Morgan Kaufmann, San Mateo, CA, 1995.
[12] J.O. Everett, Topological inference of teleology: Deriving function from structure via evidential reasoning, Department of Computer Science, Northwestern University, Evanston, IL, 1997.

[13] B. Falkenhainer, Toward a general purpose belief maintenance system, in: Proceedings Second Workshop on Uncertainty in AI, Philadelphia, PA, 1986.
[14] K.D. Forbus, Qualitative process theory, Artificial Intelligence 24 (1984) 85–168.
[15] K.D. Forbus, The qualitative process engine, in: D.S. Weld, J. de Kleer (Eds.), Readings in Qualitative Reasoning about Physical Systems, Morgan Kaufmann, San Mateo, CA, 1990, pp. 220–235.
[16] K.D. Forbus, J. de Kleer, Building problem solvers, in: M. Brady, D. Bobrow, R. Davis (Eds.), Artificial Intelligence, The MIT Press, Cambridge, MA, 1993.
[17] K.D. Forbus, P.B. Whalley, Using qualitative physics to build articulate software for thermodynamics education, in: Proceedings Twelfth National Conference on Artificial Intelligence (AAAI-94), Seattle, WA, AAAI Press/MIT Press, Cambridge, MA, 1994.
[18] D. Franke, A theory of teleology, Department of Computer Science, The University of Texas, Austin, TX, 1993.
[19] A. Goel, A model-based approach to case adaptation, in: Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society, Chicago, IL, 1991.
[20] R.P. Goldman, E. Charniak, A language for construction of belief networks, IEEE Transactions on Pattern Analysis and Machine Intelligence 15 (3) (1993).
[21] B. Hayes-Roth et al., Guardian: A prototype intelligent agent for intensive care monitoring, Artificial Intelligence in Medicine 4 (1992) 165–185.
[22] R.W. Haywood, Analysis of engineering cycles: Power, refrigerating, and gas liquefaction plant, in: Thermodynamics and Fluid Mechanics, Pergamon Press, Oxford, 1991.
[23] M. Henrion (Ed.), Propagation of Uncertainty in Bayesian Networks by Probabilistic Logic Sampling, Uncertainty in Artificial Intelligence, Elsevier, Amsterdam, 1988.
[24] E.J. Horvitz, H.J. Suermondt, G.F. Cooper, Bounded conditioning: Flexible inference for decisions under scarce resources, in: Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence (UAI-89), Windsor, Ontario, Morgan Kaufmann, San Mateo, CA, 1989.
[25] Y. Iwasaki, C.M. Low, Model generation and simulation of device behavior with continuous and discrete changes, Journal of Intelligent Systems Engineering 1 (2) (1993).
[26] F.V. Jensen, S.L. Lauritzen, K.G. Olesen, Bayesian updating in causal probabilistic networks by local computations, Computational Statistics Quarterly 5 (4) (1990) 269–282.
[27] A.M. Keuneke, Machine understanding of devices: Causal explanation of diagnostic conclusions, Department of Computer Science, The University of Ohio, Columbus, OH, 1989.
[28] B. Kuipers, Qualitative simulation, Artificial Intelligence 29 (1986) 289–388.
[29] J.E. Larsson, Diagnosis based on explicit means–ends models, Artificial Intelligence 80 (1) (1996) 29–93.
[30] M. Lind, Multilevel flow modelling of process plant for diagnosis and control, Risø National Laboratory, Roskilde, Denmark, 1982.
[31] M. Lind, Representing goals and functions of complex systems: An introduction to multilevel flow modeling, Technical University of Denmark, Copenhagen, 1990.
[32] D. McAllester, A three-valued truth maintenance system, Department of Electrical Engineering, MIT, Cambridge, MA, 1978.
[33] D.A. McAllester, Truth maintenance, in: Proceedings Eighth National Conference on Artificial Intelligence (AAAI-90), Boston, MA, Morgan Kaufmann, San Mateo, CA, 1990.
[34] N.H. Narayanan, M. Suwa, H. Motoda, Hypothesizing behaviors from device diagrams, in: J. Glasgow, N.H. Narayanan, B. Chandrasekaran (Eds.), Diagrammatic Reasoning: Computational and Cognitive Perspectives, AAAI Press, Menlo Park, CA, 1995.
[35] N.J. Nilsson, Probabilistic logic, Artificial Intelligence 28 (1) (1986) 71–87.
[36] J. Pearl, Fusion, propagation, and structuring in belief networks, Artificial Intelligence 29 (1986) 241–288.
[37] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann, Los Altos, CA, 1988.
[38] M. Pegah, J. Sticklen, W.E. Bond, Representing and reasoning about the fuel system of the McDonnell Douglas F/A-18 from a functional perspective, IEEE Expert 8 (2) (1993) 65–71.
[39] M. Ramoni, A. Riva, Belief maintenance with probabilistic logic, in: Proceedings AAAI Fall Symposium on Automated Deduction in Nonstandard Logics, Raleigh, NC, 1993.
[40] M. Ramoni, A. Riva, Belief maintenance in Bayesian networks, in: Proceedings Tenth Annual Conference on Uncertainty in Artificial Intelligence, Seattle, WA, 1994.

[41] M. Ramoni, A. Riva, Probabilistic reasoning under ignorance, in: Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society, Atlanta, GA, 1994.
[42] M. Ramoni et al., Forecasting glucose concentrations in diabetic patients using ignorant belief networks, in: Proceedings AAAI Spring Symposium on Artificial Intelligence in Medicine, Stanford, CA, 1994.
[43] B.J. Reiser et al., Cognitive and motivational consequences of tutoring and discovery learning, Institute for the Learning Sciences, Northwestern University, Evanston, IL, 1994.
[44] S.J. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall Series in Artificial Intelligence, Prentice Hall, Upper Saddle River, NJ, 1995.
[45] R.C. Schank, Explanation Patterns: Understanding Mechanically and Creatively, Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.
[46] V. Sembugamoorthy, B. Chandrasekaran, Functional representation of devices and compilation of diagnostic problem-solving systems, in: J. Kolodner, C. Riesbeck (Eds.), Experience, Memory, and Reasoning, Lawrence Erlbaum Associates, Hillsdale, NJ, 1986, pp. 47–73.
[47] D. Spiegelhalter (Ed.), Probabilistic Reasoning in Predictive Expert Systems, Uncertainty in Artificial Intelligence, Elsevier, Amsterdam, 1986.
[48] R.M. Stallman, G.J. Sussman, Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis, Artificial Intelligence 9 (2) (1977) 135–196.
[49] E. Stroulia, A. Goel, Generic teleological mechanisms and their use in case adaptation, in: Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society, Bloomington, IN, 1992.
[50] G.J. Sussman, G.L. Steele, CONSTRAINTS—A language for expressing almost-hierarchical descriptions, Artificial Intelligence 14 (1) (1980) 1–39.
[51] S. Thadani, Constructing functional models of a device from its structural description, Department of Computer Science, The Ohio State University, Columbus, OH, 1994.
[52] T. Tomiyama, Y. Umeda, T. Kiriyama, A framework for knowledge intensive engineering, in: Proceedings Fourth International Workshop in Computer Aided Systems Theory (CAST-94), Ottawa, Ontario, 1994.
[53] Y. Umeda et al., Function, behavior, and structure, in: J. Gero (Ed.), Applications of Artificial Intelligence in Engineering, Springer, Berlin, 1990, pp. 177–193.
[54] G.J. Van Wylen, R.E. Sonntag, C. Borgnakke, Fundamentals of Classical Thermodynamics, Wiley, New York, 1994.
[55] M. Vescovi, et al., CFRL: A language for specifying the causal functionality of engineered devices, in: Proceedings Eleventh National Conference on Artificial Intelligence (AAAI-93), Washington, DC, 1993.
[56] B.C. Williams, Interaction-based invention: Designing novel devices from first principles, in: Proceedings Eighth National Conference on Artificial Intelligence (AAAI-90), Boston, MA, 1990.