QPW '89

# Building Qualitative Models of Thermodynamic Processes

**John W. Collins**
Qualitative Reasoning Group
Beckman Institute, University of Illinois
405 North Mathews St.
Urbana, IL 61801

**Abstract**

This paper describes a qualitative domain theory developed for modeling fluids and thermodynamics scenarios in QPE. This work builds on the domain models of [4], adding process definitions for pumps (for liquids), compressors (for gasses), and a phase–change process for condensation.

The model includes a complete qualitative account of the thermodynamic behavior of fluids associated with each type of process. It allows significant flexibility and composability in assembling new scenarios; the user need only describe the structural configuration to be modeled, and QPE does the rest. It has been used to model a variety of fluid system scenarios, including a two-phase refrigeration system described in detail. Lessons learned from putting together a large qualitative model are discussed, hopefully preventing future model builders from falling into the same traps.

**Area:** Commonsense Reasoning
**Subarea:** Qualitative Physics
**Type:** Science
**Length:** 5023 words

# 1 Introduction

This paper develops a qualitative domain model for thermodynamic and fluid systems, based on Qualitative Process theory. This model contains a complete set of process definitions for modeling fluid flows (liquid or gas, forced or free), heat flows, and phase transitions between the liquid and gaseous states. The model has been applied to a variety of scenarios, including a refrigeration system, which is described here in detail. We conclude with a discussion of the lessons learned as a result of putting together a large–scale qualitative model. Before describing the model in deatail, some general modeling issues are discussed.

# 2 Modeling Issues

The development of a set of domain descriptions capable of modeling a wide variety of fluid and thermodynamic scenarios requires careful consideration of several issues.

## 2.1 Modularity

In order to manage complexity, the domain is partitioned into a set of relatively independent modules. For example, *heat flow* is sufficiently different from other domains in thermodynamics to be considered a separate module. Yet no module is totally independent from the others; heat flows involve physical objects, as do all other processes. In general each module is dependent on a set of lower modules, and may be used by still higher modules. As a matter of pragmatics, each module is stored in a separate file so that only those modules required for a particular scenario need be loaded.

Hierarchical representation offers the same benefits to qualitative reasoning as to the other AI disciplines: compactness of representation together with a natural mechanism for generalization. Hierarchies are used extensively in representing physical entities; for example, a contained–liquid is a contained–stuff, which is a physob. Quantities and other properties are inherited from the general class to the specific instance.

Hierarchical representations have also been applied to processes, though to a lesser extent. When two process descriptions share a great deal in common, as with liquid flow and gas flow, a common abstract process description may be defined to contain their intersection. Their differences may then be handled using simple view or process descriptions. This reduces duplication, and thereby reduces the likelihood of introducing subtle bugs.

## 2.2 Level of Detail

Of all the modeling decisions, the most difficult has been choosing the appropriate level of detail. The first step is to partition the world up into discrete objects. The coarseness of the partitioning will determine the coarseness (and efficiency) of the reasoning. For example, reasoning at the level of contained-liquids would be too coarse if our goal were to understand sloshing. Here are a few other examples:

*Modeling Idealizations:* Should the model for liquid flow consider the acceleration of the liquid in the path, or settle for a *equilibrium* model which relates flow rate and pressures directly?

*Qualitative Approximations:* Should the model include a quantity representing the *conductance* (or *resistance*) of a fluid path, or simply define the flow rate as the qualitative difference in pressures across the path. Having conductance provides a hook for adding a continuous model for valves, and avoids the direct comparison of quantities of different units (eg. flow-rate and pressure).

# 3 A Tour of the Model

This section presents a tour through the model for the various domains, beginning with simple physical objects and concluding with a complex model of phase transitions. Space precludes including significant portions of the model in the body of the paper, so the complete listing is attached as an appendix.

## 3.1 Physical Objects

Since we are modeling a physical domain, all of our processes must necessarily involve physical objects. We define an entity for physical objects, called *physob*, which describes the basic physical properties common to all objects. For example, all physobs are given the quantities of `mass`, `volume`, `heat` and `temperature`, which are constrained to be non-negative. These properties are inherited by specific instances of physical objects, such as contained-liquids.

## 3.2 Contained Liquids

Hayes [6] has formalized two views for liquid objects: the *contained-liquid* view and the *piece-of-stuff* view. A contained-liquid is defined as the liquid which exists within the confines of some container. The amount of a contained-liquid can change, as when liquid flows in or out; and can go to `ZERO`—in which case the contained-liquid vanishes. The contained-liquid re-appears when liquid (of the same substance) is added to the empty container. This view of liquids is the basis for nearly all qualitative models for fluids developed so far. An alternative view of fluids is provided by the piece-of-stuff ontology, which defines a fluid object as a particular collection of molecules whose mass is fixed but whose location may vary.

Contained-liquids are modeled as a specialization of a contained-stuff, which may be `liquid` or `gas`. A contained-stuff exists whenever there is a non-`ZERO` amount of fluid in some container. When a contained-liquid exists, it is a physob, and inherits `mass`, `volume`, `temperature` and `heat`.

The `mass` of the contained-liquid is equal to and determined by the amount of liquid substance in the container.[1] The `volume` is qualitatively proportional ($\propto_{Q+}$) to the `mass` of the contained-liquid, and has the same sign; that is, the two quantities have a *Correspondence* at `ZERO`.

Contained-liquids also have the quantity: `level`, which depends on `volume`. `level` has two interesting limit points: the `bottom-height` and `top-height` of the container, which correspond to empty and full containers, respectively. In simpler models the `level` is not allowed to exceed the `top-height` of the container; however this restriction is relaxed for the overflow model discussed below.

## 3.3 Liquid Flow

Liquid flow occurs whenever an unobstructed fluid path connects two containers of liquid at different levels. We ignore the dynamics involved in accelerating the mass of fluid in the path, and simply consider the flow rate as a monotonic function of the pressure drop across the path. Whenever a liquid flow process is active, its `flow-rate` positively influences the amount of liquid at the destination and negatively influences the amount of liquid at the source of the flow.

### 3.3.1 Portals

The boundaries of a contained stuff are for the most part rigidly defined by the physical presence of its container. However, not all containers are completely closed. For instance, a container may have an open

[1] These two quantities differ in that `mass` disappears when the contained-liquid ceases to exist, and so can never equal `ZERO`.

top, or may have fluid paths (eg.,pipes) connected to it. These areas are characterized by the absence of a physical wall defining the boundaries between the inside and outside of the container. Following the terminology used by Hayes [6], we call these interfaces *portals*.

Portals are used in our model to define the connectivity of containers and fluid paths. Portals posess the quantities: **pressure** and **temperature**; these may be viewed as belonging to the fluid (liquid or gas) at the location of the portal. Portals also have a quantity: **height**, which is compared to the **level** of the contained-liquid to determine the portal's **submerged-depth**. The liquid-flow process is augmented to require that the portal at the source of the flow be submerged; this is stronger than the contained-liquid requirement, since the portal need not be at the bottom of the container.

### 3.3.2 Flow through Non-Level Paths

The **pressure** of a contained-liquid is a function of the **level** of the liquid. Since there is only a single **pressure** quantity for each contained-liquid, it is clear that this quantity cannot represent the liquid's **pressure** as a function of depth; rather it must represent the **pressure** at some arbitrary depth—such as the bottom of the container, or the arbitrary reference from which **heights** and **levels** are measured.

The **pressure** at a submerged portal is not necessarily equal to that of its submerging contained-liquid, since the portal's **height** is not generally equal to **ZERO**. The portal's **pressure** is a function of its **submerged-depth**. The portal's **pressure** is also dependent on the **pressure** of any contained-gas which occupies the same container.

It should be possible to determine the direction and rate of liquid flow by looking only at properties of the portals at either end of the flow path. But the **pressures** at the portals of a non-level path do not by themselves determine the direction of flow; the additional force of gravity acting on the liquid in the path may cause flow from lower to higher **pressure**.

The quantity **head** is introduced in order to properly account for flow through non-level paths. In classical fluid dynamics, head represents the height assumed by a column of fluid open to the atmosphere (or to a vacuum). We ignore the component of **head** due to the velocity of the fluid, and qualitatively define head as the sum of **pressure** and **height**. It is meaningful to refer to the **head** of a contained-liquid, since this quantity does not change with depth. The liquid flow process will depend on the relative **head** at the two ends of the flow path; flow will always occur from higher to lower **head**.

### 3.3.3 Thermal Properties of Liquid Flow

The model of liquid flow presented so far has ignored the effects of **temperature** differences between the source and destination of the flow. Unless the flow process can influence the **heat** of the contained-liquids, the **heat** will remain constant even as the liquid objects appear and disappear.

There are two obvious approaches to modeling thermal properties in liquid flow. We may either consider both intrinsic and extrinsic thermal quantities (i.e., **temperature** and **heat**), or we may consider only the intrinsic quantity: **temperature**. Considering the latter approach first, **temperature** must be directly influenced by "heat flows" and other thermal processes; the quantity **heat** is not included in the model. Depending on the relative **temperatures** of the source and destination of liquid flow, the appropriate thermal mix-in process instantiates to directly influence (up or down) the **temperature** at the destination. The **temperature** at the source is uninfluenced. The direct influencer of **temperature** is a function of the difference in **temperatures**, as well as the **flow-rate** of the process instance (*pi*) and the **mass** of the contained-liquid at the destination:

$$\texttt{Thermal-Rate(pi)} = \frac{\texttt{Flow-Rate(pi)}}{\texttt{Mass(dest)}}(\texttt{Temp(source)} - \texttt{Temp(dest)})$$

This approach pays a high price for excluding **heat** from the model. It seems unnatural for a liquid flow or heat flow process to require knowledge of the amount of liquid at the destination. In addition, we lose the sense of a thermal flow; *temperature* is not moved from source to destination, nor does it obey conservation laws.

A more elegant alternative is to define the **temperature** of a contained-liquid as a ratio of **heat** and **mass**, which results in the following dependencies:

$$\texttt{temperature} \propto_{Q+} \texttt{heat}; \qquad \texttt{temperature} \propto_{Q-} \texttt{mass}$$

**Heat** and **mass** are directly influenced by various flow processes. This definition is consistent, but can often result in ambiguity. For example, both **heat** and **mass** are decreasing at the source of a liquid flow and increasing at the destination, so the net effect on the **temperatures** cannot be resolved given the ambiguous combination of the $\propto_{Q+}$ and $\propto_{Q-}$ .

This problem motivated the development of a technique for resolving ratios, which is described in detail in [2]. Basically the technique involves pairing up the influencers on numerator and denominator, and resolving the net influence of each pair in isolation. As long as no two pairs provide opposite influences, the derivative of the ratio will be unambiguously resolved.

Augmented with this technique, QPE is powerful enough to reason that the **temperature** at the source of a liquid flow remains constant, while the **temperature** at the destination behaves according to the difference in **temperatures**. This technique also solved another problem: recognizing that flow into an empty container results in a contained-liquid at the same **temperature** as the flow coming in. By requiring that a ZERO-**mass** contained-stuff have constant **temperature**, it follows that the initial **temperature** will be the same as that of the liquid flowing in (otherwise it would be changing, a contradiction). This constraint also covers cases of multiple flows of different **temperatures** into an empty container.

## 3.4 Pumped Flow

Pumps are used to drive fluid flow when gravity won't. The simplest qualitative model of a pump assumes a constant (positive) flow rate, as long as there is liquid in the source container to be pumped. This model corresponds to a positive–displacement pump.

Our domain model for pumps is based on the more common centrifugal pump, in which the flow rate depends on the pressure rise across the pump. The rate of flow decreases as the pressure rise increases, until some maximum pressure is reached. Given a sufficiently high pressure rise, the pump will have a "negative" flow.[2] The current model includes views to distinguish working, coasting and "losing" pumps, based on the pressure rise or drop across the pump. The thermal behavior of a pumped liquid is handled in the same way as in the liquid flow process described above.

## 3.5 Gasses

Many thermodynamic systems of interest involve gasses. Unfortunately, gasses introduce several new difficulties. Unlike liquids, which are incompressible, gasses expand to fill their container. In the process of expanding or compressing, gasses are subject to doing work or being worked upon. These processes affect the internal energy of the gas, which in turn affects its temperature and pressure. This section describes the qualitative model developed to account for the behaviors of gasses in fluid systems.

[2]This model of a pump is equivalent to a constant displacement pump in parallel with a (restricted) flow path.

### 3.5.1 Open and Closed Containers

Because gasses expand to fill their container, it is necessary to introduce a new distinction for containers, namely: *open* vs. *closed* containers.[3] An open container is only capable of containing liquid, and is exposed to the constant pressure of the atmosphere. A closed container may contain liquid or gas, or both. The pressure in a closed container may vary, as determined by the amount of contained-gas present and how hot it is.

### 3.5.2 Contained-Gasses—Ideal Gas Law

As with liquids, contained-gasses are modeled as a specialization of contained-stuffs. Contained-gasses share all the same quantities as contained-liquids, except that gasses do not have a `level`.

When a gas is sufficiently above its boiling point, its behavior is approximated by the ideal gas law:

$$PV = mRT = U$$

where P, V, m and T represent pressure, volume, mass and (absolute) temperature, respectively. R is the gas constant for the substance in question; U is the internal energy of the gas, which for simplicity will be referred to as **heat**.

Because QP theory requires a causal model, it is necessary to replace the constraint equation representation of the ideal gas law with a set of directed influences. The first step in constructing such a model is to identify the independent parameters among the quantities; these are the inputs to the causal chains. In QP theory, these are exactly the directly influenced quantities. As with liquids, it is reasonable to choose **mass** and **heat** as independent parameters, since there are clearly-identifiable processes which directly influence these quantities. In addition, **volume** is viewed as independent, since the **volume** of a contained-gas is determined by the **volume** of its container.[4]

Using these three quantities (i.e., heat, mass and volume) as independent parameters, we can solve for the remaining (dependent) quantities as follows:

$$P = U/V; \quad T = U/m;$$

The constant R is dropped from the representation, since it does not affect the qualitative behavior of a gas. Note that the definition of **temperature** for a contained-gas is the same as for contained-liquids and all other physical objects.

The expression for pressure may seem unintuitive, since it involves neither temperature nor mass. Intuitively when gas is added to a closed container, or when the contained-gas is heated, the pressure of the gas will increase. But in both cases heat is being added to the gas while its volume remains constant. The model predicts that if the amount of the gas could be increased while its heat is held constant (say by adding gas at absolute zero temperature), then the pressure would remain unchanged; this result does not conflict with an intuitive view based on a product of mass and temperature, since the temperature in the above case would be decreasing, and the net influence on pressure would be ambiguous.

### 3.5.3 Gas Flow and Expansion

The flow of gasses is in many respects analogous to liquid flow. The gas flow process is driven by a pressure drop across a fluid path, and influences the amount of contained-gas at the source and destination of the

[3] A familiar example of a closed container is a household pressure cooker.

[4] Expansion and compression processes have been developed to influence a container's **volume**.

flow. As with liquids, the dynamics of flow rate acceleration based on the inertia of the fluid are ignored, in favor of an equilibrium model of flow.

Unlike the model for liquid flow, gas flow involves an expansion of the fluid within the flow path. As the gas expands, it does work on its surroundings, at the expense of some of its internal (heat) energy. If the fluid path is in fact a turbine, then some of the energy of the gas may be converted to mechanical energy and used to do physical work. In this case, the energy of the gas arriving at the destination of the flow is less than the energy leaving the source. Since the mass flows are the same, the temperature of the gas must drop across the path. In addition, gas flowing through a constriction will be cooler as it flows faster, since it has exchanged thermal energy for kinetic energy.[5]

The process definition for gas flow through a restricted fluid path moves **mass** and **heat** from the source of the flow to the destination. The amount of **heat** moved is greater than the original **heat** content of the gas being moved, due to the work being done by the source on the destination. Thus the gas flow process tends to reduce the **temperature** at the source and increase the **temperature** at the destination.

### 3.5.4 Compressed Gas-Flow—Compression

Like liquids, gasses can be made to flow from lower to higher pressures, through a *compressor*. Compressors, like pumps, can be modeled in a variety of ways. The simplest model of a compressor has a constant flow rate; The process definition for this model would have no quantity conditions, and would be active whenever there is a contained-gas at the compressor's inlet.

A more realistic model of a compressor determines the flow rate as proportional to the *density* of the gas at the inlet. Properties at the outlet do not affect the flow rate in this model, which represents an ideal positive-displacement (eg. piston/cylinder) compressor.

## 3.6 Liquid–Gas Phase Transitions

Many thermodynamic cycles of interest—including modern air conditioners and power plants—involve phase changes between the liquid and the gaseous phase. Developing a realistic qualitative model for phase transitions proved to be particularly challenging.

The phase of a substance is primarily a function of its temperature; each phase exists only within a certain range of temperatures. When the temperature of a liquid reaches its upper limit—namely, the boiling point of the substance—then boiling begins to occur; when a gas is cooled to this same limit, condensation results. The boiling point of a substance is not constant but increases with pressure; thus boiling (and condensation) occur at a higher temperature in a pressurized vessel (such as a car's radiator or a pressure cooker) than in an open pan on a stove. Likewise, lukewarm water will boil in a vacuum, and superheated steam will condense when subjected to sufficiently high pressure.

### 3.6.1 Thermal Behavior of Phase Transitions

Correctly modeling the thermal aspects of a phase transition requires careful consideration. One useful technique for reasoning about a complex process is to decompose it into an equivalent sequence of simple events. For example, boiling may be decomposed in the following way:

1. An infinitesimal piece of liquid is selected as the next candidate to undergo the transition from liquid to gas. This infinitesimal piece of liquid is removed from the contained-liquid by subtracting out its mass and heat content from the corresponding properties of the contained-liquid.

[5] These properties were the basis for some of the early refrigerators.

2. In order to convert the piece of liquid into a piece of gas, additional heat—known as the *latent heat of vaporization*—is transferred to the piece of liquid.

3. As the phase transition proceeds, the piece-of-stuff expands, thereby expending energy (heat) as it does work on its surrounding contained-gas.

4. Finally, the piece of gas is added to the contained-gas by incrementing its mass and heat.

This reasoning technique has been similarly applied to the condensation process.

### 3.6.2 Evaporation and Boiling

An important modeling issue for the boiling process concerns the source of the latent heat of vaporation. One possible choice is to require the presence of an external heat source, whose `temperature` is above the boiling point. Another option is to take the heat from the surrounding contained-liquid. These two modeling choices result in very different predicted thermal behaviors for the contained-liquid. In the first case the `temperature` of the contained-liquid is unaffected, while in the second case it is negatively influenced.

A simple model of the boiling process is active whenever a contained-liquid is at its boiling point and has a heat flow into it. The rate of boiling is proportional to the heat flow rate into the liquid. The boiling process negatively influences the amount of liquid in the container and positively influences the amount of gas (of the same substance) in the container. This model corresponds to the first choice described above, where the boiling piece of liquid draws heat from an external heat source.

There are several problems with this model of boiling; for example, the model requires that the net heat flow into the liquid be available as a quantity. There may be multiple heat flows into and out of the liquid, and only if the net effect is positive will boiling occur. Even if this quantity were available, it would be incorrect to define the boiling rate solely in terms of the net heat flow. In fact it is possible to boil a liquid without adding any heat at all, simply by reducing the pressure and thus reducing the boiling point below the current temperature of the liquid.

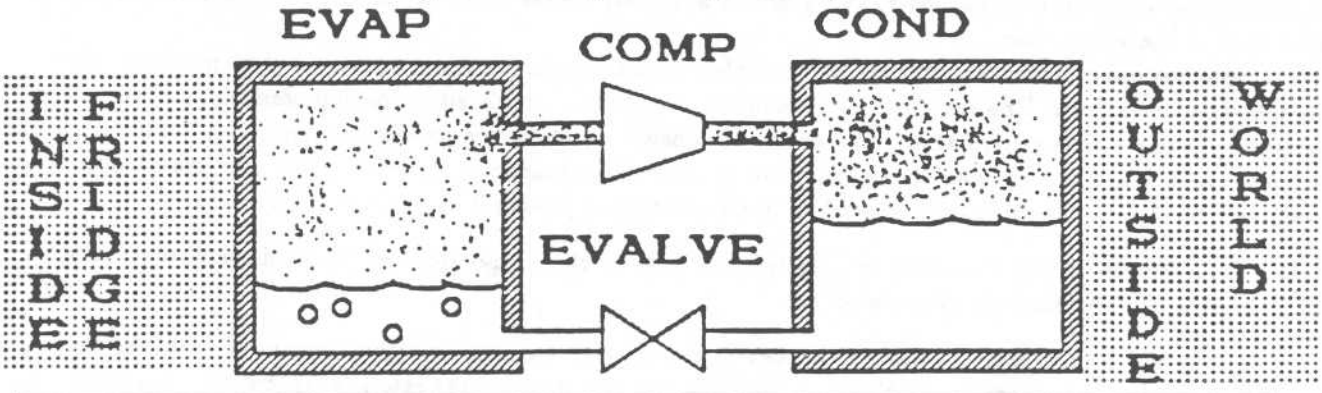
This problem is analogous to the overflow of a contained-liquid through the top of its open container. We would like to enforce that the `level` of the contained-liquid never exceed the `top-height` of the container, and that the overflow rate equals the net rate of flow entering a full container. As with boiling, it is possible to move the limit point of the overflow process, for example by raising or lowering a gate at the top of the container. Thus overflow can occur without any flow into the container. The problem here is that the requirement that the level (temperature) of the liquid never exceed its limit point is an idealization. In reality the fluid level *must* exceed the top height of the container for overflow to occur; the difference between these two quantities determines the rate of overflow.

This augmented model for overflow can be mapped back across the analogy to boiling. In the resulting model, the rate of boiling is proportional to the amount by which the temperature of the liquid exceeds its boiling point. This model in some ways improves on previous models; however, it is somewhat unintuitive.

To relieve the reader's unease, consider a slightly different perspective: a boiling liquid does not actually have a single temperature; rather it has a distribution of temperatures centered at some mean value. At the molecular level, some molecules will be moving faster than others. If we view the boiling process as Maxwell's demon grabbing and removing only the fastest molecules, then clearly the average temperature of the liquid is reduced as a result. Thus it seems reasonable for the boiling process to remove the required `heat` from the liquid.

Unfortunately, this model allows boiling to occur even when there is no heat flow into the liquid *and* the boiling point is constant. While this phenomenon may actually occur, it is of such short duration that we would prefer not to include it in our model. In a real boiling liquid, the removal of latent heat from the

Figure 1: A two-phase refrigeration system




liquid is sufficient to prevent it from heating up more than infinitesimally above the boiling point. Without *order of magnitude* reasoning, however, there is no way to capture this constraint.

The boiling process directly influences the **mass** and **heat** of the two contained-stuffs. The two influences on **mass** are equal and opposite, as required for conservation of matter. The latent heat of vaporization must be added to the liquid as it boils, and is assumed to flow from the contained-liquid. The **heat** of the contained-liquid is negatively influenced both by the removal of liquid and by the drain caused by the latent heat of vaporization, so the net influence on the liquid's temperature is negative. This provides a stabilizing influence on the boiling liquid by pushing its temperature back below the boiling point.

### 3.6.3 Condensation

The inverse of the evaporation process is condensation, the transformation of gas into liquid. The only difficult modeling issue for condensation is the recipient of the latent heat given up by the condensing gas. We choose the contained-gas, since it surrounds a piece of gas as it condenses.

The condensation process directly influences the **mass** and **heat** of the two contained-stuffs. The two influences on **mass** are equal and opposite, as required for conservation of matter. The same is not true for **heat**, since the latent heat of vaporization must be removed from the gas as it condenses, and is assumed to flow into the contained-gas. This latent heat more than compensates for the heat of the condensate leaving the contained-gas, so the net influence on the **heat** of the contained-gas is positive. In general the latent heat added to a condensing gas is sufficient to prevent it from cooling off more than infinitesimally below its boiling point. Again, order of magnitude reasoning would be required to conclude this fact.

## 4 An Example: Modeling a Refrigerator

The domain model described above has been applied to a variety of scenarios, ranging from a simple heat flow between two physobs to a complex refrigeration system. Modeling a refrigerator constitutes a significant test of the domain theory, since it involves most of the defined process types. A two-phase refrigerator involves liquid and gas flow, heat flow, and phase transitions between the liquid and gaseous states.

Figure 1 depicts the configuration for a simple two-phase refrigeration system. For simplicity, the evaporator and condenser coils have been modeled as closed-containers rather than path-type heat exchangers.

The contained-liquid in the evaporator and the contained-gas in the condenser are in thermal contact with their surroundings, so that heat flows can support the respective phase transitions. A compressor moves gas from the evaporator to the condenser, and a simple fluid path serves as an expansion valve, allowing liquid to return to the evaporator.

In order to maintain tractability for this complex model, the scenario was constrained to produce only steady-state behaviors. The resulting envisionment consists of a single situation representing the normal operating mode of the refrigerator. The situation consists of six active process instances: a liquid flow, a compressed gas flow, two heat flows, and one of each phase transition process type. The steady-state operation of the refrigerator can be described in terms of these processes as follows:

1. The pressure in the condenser is greater than that in the evaporator, so liquid flows through the expansion valve into the evaporator.

2. The liquid immediately begins to evaporate, due to the low boiling point associated with the low pressure in the evaporator. The rate of liquid flow exactly matches the rate of evaporation, thus maintaining a constant amount of liquid in the evaporator. However, the heat carried into the evaporating liquid by the flow through the expansion valve is less than the heat taken away by the evaporated gas.

3. In order to maintain constant temperature, a heat flow process from the refrigerator interior must make up the difference. Thus the steady-state temperature of the liquid in the evaporator is lower than the inside temperature of the fridge.

4. The gas in the evaporator is compressed and moved into the condenser. The work done by the compressor raises the heat and temperature of the gas as it is compressed.

5. The gas is now hotter than room temperature, but below the higher boiling point in the high-pressure condenser. Condensation occurs.

6. As the gas condenses, it gives off heat, which flows into the room. The condensed liquid is now ready to flow through the expansion valve, thus completing the cycle.

This scenario represents one of the largest models run by QPE to date. Although it created only ten view instances and eight process instances, these resulted in 332 inequality relations among 173 numbers. QPE used about ten minutes of processor time on a Symbolics 3600 to produce the highly-constrained envisionment.

# 5 Problems Encountered, Lessons Learned

During the development of the model for thermdynamics described above, several recurring problems were encountered. A sampling of these are outlined here.

## 5.1 Changing Existence

Several problems arise when allowing for the appearance and disappearance of objects. For instance, because a contained-liquid does not exist when the amount of liquid in the container is ZERO, the liquid flow process is not allowed to refer to any properties of the contained-liquid at the destination. Otherwise flow could not be initiated into an empty container.

### 5.2 Causality

Dependencies in QP theory carry with them a causal direction. Qualitative proportionalities must run outward from directly influenced quantities to the other (indirectly) influenced quantities; since these relations are viewed as imputing causality, loops among the qualitative proportionalities are not allowed. *Qprop loops*, as they are affectionately known, sometimes creep into a domain model, and result in a hard error in QPE.

Qprop loops generally come in two varieties: *self–loops* and *cycle loops*. A self–loop occurs when a dependency is expressed in two different ways (eg. $X = Y$ & $Y = X$, or $V = IR$ & $I = V/R$). A cycle–loop results from a chain of dependencies around a cycle of entities, as might occur when relating node voltages and component voltages in an electronic circuit.

One way to avoid certain qprop loops is to avoid having multiple copies of the same quantity. For example, instead of having two equal and opposite forces, `force-on(B,A)` & `force-on(A,B)`, use a single quantity: `force-between(A,B)`, whose actual effect depends on the orientation of the two objects.

### 5.3 No Negation–by–Failure

QPE requires that certain facts be known in every situation. Examples include existence of quantities and conditions on individuals of views and processes. Ensuring that these facts are known can make an otherwise simple relation become very complicated. This is the source of one of the most common bugs among domain models for QPE.

## 6 Discussion

The refrigerator example described above clearly demonstrates the composability afforded by Qualitative Process theory and the domain descriptions presented in this paper. A simple structural description of a refrigerator is automatically expanded into a set of process instances and a qualitative behavioral description; this in turn may serve as the basis for generating other types of descriptions, or for solving a variety of engineering–type problems (see [1] for details).

This paper has discussed the issues involved in developing a large-scale qualitative model of a real-world domain. It is hoped that other researchers in Qualitative Reasoning may benefit from this discussion, by avoiding the kinds of mistakes made in developing the model.

## References


[1] Collins, J. and Forbus, K. "Reasoning about Fluids via Molecular Collections", in *Proceedings of the National Conference on Artificial Intelligence*, Seattle, July, 1987.

[2] Collins, J. "Qualitative Algebra in QPE", *in preparation.*

[3] de Kleer, J. and Brown, J. "A Qualitative Physics based on Confluences", *Artificial Intelligence*, **24**, 1984.

[4] Forbus, K. "Qualitative Process Theory" *Artificial Intelligence*, **24**, 1984.

[5] Forbus, K. "The Problem of Existence", in Proceedings of the Cognitive Science Society, 1985.

[6] Hayes, P. "Naive Physics 1: Ontology for Liquids", in Hobbs, J. and Moore, B. (Eds.), *Formal Theories of the Commonsense World*, Ablex Publishing Corporation, 1985.

[7] Kuipers, B. "Common Sense Causality: Deriving Behavior from Structure", *Artificial Intelligence*, **24**, 1984.

[8] Kuipers, B. "Abstraction by Time-Scale in Qualitative Simulation", in *Proceedings of the National Conference on Artificial Intelligence*, Seattle, July, 1987.

[9] Weld, D. "Switching Between Discrete and Continuous Process Models to Predict Genetic Activity", MIT Artificial Intelligence Lab TR-793, October, 1984.

[10] Williams, B. "Qualitative Analysis of MOS Circuits", *Artificial Intelligence*, **24**, 1984.

[11] Iwasaki, Y., and Simon, H. "Causality in Device Behavior", *Artificial Intelligence*, **29**, 1986.

# APPENDIX

## Thermodynamics Domain Theory

```
;;; -*- Mode: Lisp; Syntax: Common-lisp; ; Package: QPE -*-

 ;;;; Physobs:
 ;;; All physical objects are endowed with certain unalienable properties:

      ;; AMOUNTS:
(defQuantity-Type Mass Individual)
(defQuantity-Type Heat Individual)
      ;; TEMPERATURES:
(defQuantity-Type Temperature Individual)
(defQuantity-Type Tboil Individual)
      ;; PRESSURES:
(defQuantity-Type Pressure Individual)
      ;; VOLUMES:
(defQuantity-Type Volume individual)

(defentity physob
  (simple-physob ?self)
  ;;; Main characteristic is that it has a number of quantities:
  (quantity (Mass ?self))
  (quantity (Temperature ?self))
  (quantity (Pressure ?self))
  (quantity (Volume ?self))
  (quantity (Tboil ?self))
  ;;; There are a few state-independent relationships:
  (Q= (temperature ?self) (/+ (Heat ?self) (Mass ?self)))
  (not (less-than (A (Heat ?self)) ZERO))
  (not (less-than (A (Mass ?self)) ZERO))
  (greater-than (A (Temperature ?self)) ZERO))

(defentity simple-physob  ; used for simple heat-flow examples.
  (quantity (heat ?self)))

(adb:rule :intern (((physob ?ob) . :TRUE))
          (adb:rnogood (((equal-to (a (heat ?ob)) ZERO) . :FALSE)
                        ((equal-to (a (amount-of ?ob)) ZERO) . :TRUE))))
```



```
;;; -*- Mode: Lisp Package: QPE; Syntax: Common-lisp; -*-

 ;;;; Heat Flow

 ;;;Types of Quantities:
      ;; TEMPERATURES:
(defQuantity-Type Temperature Individual)
(defQuantity-Type Temp-diff Individual)
      ;; RATES:
(defQuantity-Type Heat-Flow-Rate Individual)
      ;; COEFFICIENTS:
(defQuantity-Type Thermal-Conductance Individual)

 ;; Entities:
(defentity Temperature-source
  (simple-physob ?self)
  (quantity (temperature ?self)))

(defentity Temperature-sink
  (simple-physob ?self)
  (quantity (temperature ?self)))

(defentity Heat-Path
  (quantity (thermal-conductance ?self))
  (greater-than (A (thermal-conductance ?self)) ZERO))

 ;; Processes:
(defprocess (Heat-Flow ?src ?dst ?path)
  Individuals ((?src :type simple-physob)
               (?dst :type simple-physob)
               (?path :type Heat-Path
                      :conditions (Heat-Connection ?path ?src ?dst)))
  Preconditions ((heat-aligned ?path))
  QuantityConditions ((greater-than (A (temperature ?src))
                                    (A (temperature ?dst))))
  Relations ((quantity Temp-diff)
             (quantity Heat-flow-rate)
             (Q= Temp-diff (- (temperature ?src) (temperature ?dst)))
             (Q= Heat-Flow-Rate (*+ Temp-diff (thermal-conductance ?path))))
  Influences ((I+ (Heat ?dst) (A Heat-flow-rate))
              (I- (Heat ?src) (A Heat-flow-rate))))
```

```
;;; Replenish an infinite source:
(defprocess (Heat-Replenish ?hf)
  Individuals ((?src :type Temperature-source)
               (?hf :type (Process-Instance Heat-Flow)
                    :conditions (?hf SRC ?src)))
  QuantityConditions ((active ?hf))
  Influences ((I+ (Heat ?src) (A (Heat-Flow-Rate ?hf)))))

;; Relieve an infinite sink:
(defprocess (Heat-Relieve ?hf)
  Individuals ((?dst :type Temperature-sink)
               (?hf :type (Process-Instance Heat-Flow)
                    :conditions (?hf DST ?dst)))
  QuantityConditions ((active ?hf))
  Influences ((I- (Heat ?dst) (A (Heat-Flow-Rate ?hf)))))
```



```
;;; -*- Mode: Lisp; Syntax: Common-lisp; Package: QPE -*-
;;;; Contained Stuffs
;;; Contained-liquids & contained-gasses are generalised to contained-stuffs.

;;; USES: Physob

;; AMOUNTS:
(defQuantity-Type Amount-of-in Constant Constant Individual)
(defQuantity-Type Head Individual)

;; LEVELS:
(defQuantity-Type Level Individual)
(defQuantity-Type Fluid-Level Individual)
(defQuantity-Type Bottom-Height Individual)
(defQuantity-Type Top-height Individual)
(defQuantity-Type Height Individual)

;; Rule for Amount-of-in:
(Rule :intern (((distinguish existence) . :TRUE)
               ((container ?c) . :TRUE)
               ((substance ?s) . :TRUE)
               ((state ?st) . :TRUE))
      (adb:rassert! ((quantity (amount-of-in ?s ?st ?c)) . :TRUE))
      (adb:rassert! ((less-than (a (amount-of-in ?s ?st ?c)) zero) . :FALSE)))

(Rule :intern (((Consider all contained-stuffs) . :TRUE)
;              ((distinguish existence) . :FALSE))
               ((container ?c) . :TRUE)
               ((substance ?s) . :TRUE)
               ((state ?st) . :TRUE))
      (adb:rassert! ((contained-stuff (C-S ?s ?st ?c)) . :TRUE)))

;; Assertions for Atmospheric Pressure:
(assert! '((exists ATMOSPHERE) . :TRUE))
(assert! '((quantity (pressure ATMOSPHERE)) . :TRUE))
(assert! '((greater-than (A (pressure ATMOSPHERE)) ZERO) . :TRUE))
```

```
;;; Entities:

;;; Define Contained-Stuffs:
(defentity (Contained-Stuff (C-S ?sub ?st ?can))
  (physob (C-S ?sub ?st ?can))
  (((C-S ?sub ?st ?can) SUBSTANCE ?sub) . :TRUE)
  (((C-S ?sub ?st ?can) STATE ?st) . :TRUE)
  (((C-S ?sub ?st ?can) CONTAINER ?can) . :TRUE)
  (function-spec tboil-fun
    (Qprop (Tboil (C-S ?sub ?st ?can))
           (pressure (C-S ?sub ?st ?can)))))

;;; Define Contained-liquids:
(defentity (Contained-Stuff (C-S ?sub liquid ?can))
  (Contained-Liquid (C-S ?sub liquid ?can)))

(defentity (Contained-Liquid (C-S ?sub liquid ?can))
    ;;; Liquids have a novel quantity, level.
  (quantity (Level (C-S ?sub liquid ?can)))
  (Qprop (level (C-S ?sub liquid ?can))
         (Volume (C-S ?sub liquid ?can)))
  (Ordered-Correspondence
    ((A (level (C-S ?sub liquid ?can))) (A (bottom-height ?can)))
    ((A (volume (C-S ?sub liquid ?can))) ZERO))
  (Ordered-Correspondence
    ((A (level (C-S ?sub liquid ?can))) (A (top-height ?can)))
    ((A (volume (C-S ?sub liquid ?can))) (A (volume ?can))))
    ;;; Relate volume to amount-of.
  (Qprop (volume (C-S ?sub liquid ?can))
         (amount-of (C-S ?sub liquid ?can)))
  (Ordered-Correspondence
    ((A (volume (C-S ?sub liquid ?can))) ZERO)
    ((A (amount-of (C-S ?sub liquid ?can))) ZERO))
  (not (greater-than (A (volume (C-S ?sub liquid ?can)))
                     (A (volume ?can))))
  ;;; Inherit pressure & head from container:
  (Q= (pressure (C-S ?sub liquid ?can)) (pressure ?can))
  (quantity (head (C-S ?sub liquid ?can)))
  (Q= (head (C-S ?sub liquid ?can)) (head ?can))
  (Q= (fluid-level ?can) (level (C-S ?sub liquid ?can))))
```



```
;;; Define contained gasses:
(defentity (Contained-Stuff (C-S ?sub gas ?can))
     (Contained-Gas (C-S ?sub gas ?can)))

(defentity (Contained-Gas (C-S ?sub gas ?can))
   ;;; The pressure of a container is the same as that of its contained gas:
   (Q= (pressure ?can) (pressure (C-S ?sub gas ?can)))
   ;;; The next Q= expresses the ideal gas law: PV = mRT = HEAT
   (Q= (pressure (C-S ?sub gas ?can)) (/O+ (heat (C-S ?sub gas ?can))
                                          (volume (C-S ?sub gas ?can)))))

;;; Define Open and Closed Containers:
(defentity container
   ;;; Simple container geometry only has heights of bottoms and tops
   (quantity (bottom-height ?self))
   (quantity (top-height ?self))
   (greater-than (A (top-height ?self)) (A (bottom-height ?self)))
   (quantity (head ?self))
   (quantity (fluid-level ?self))
   (not (less-than (A (fluid-level ?self)) (A (bottom-height ?self))))
   (not (less-than (A (head ?self)) (A (fluid-level ?self))))
   (Q= (head ?self) (+ (fluid-level ?self) (pressure ?self)))
   (quantity (volume ?self))
   (greater-than (A (volume ?self)) ZERO)
   (quantity (pressure ?self))
   (not (less-than (A (pressure ?self)) ZERO)))

(defentity Open-Container
   (container ?self)
   (Q= (pressure ?self) (pressure ATMOSPHERE)))

(defentity Closed-Container
   (container ?self))

;;; View for creating Contained-liquids and gasses:

(defview (Contained-Stuff (C-S ?sub ?st ?can))
   Individuals ((?can :type container)
                (?sub :type substance)
                (?st  ;:type state
                      :conditions (state ?st) (distinguish existence)))
   Preconditions ((Can-Contain-Substance ?can ?sub ?st))
   QuantityConditions ((greater-than (A (Amount-of-in ?sub ?st ?can)) ZERO))
   Relations ((there-is-unique (C-S ?sub ?st ?can))
              (Q= (amount-of (C-S ?sub ?st ?can)) (amount-of-in ?sub ?st ?can))))
```

```
; Views for full & empty containers:

(rule :intern (((container ?can) . :TRUE))
  (adb:rassert! ((Full ?can) . :FALSE))) ; Temporary patch

(defview (Empty ?can)
  Individuals ((?can :type container)
               (?sub ;:type substance
                     :conditions (substance ?sub) (distinguish empty containers)))
  QuantityConditions ((equal-to (A (amount-of-in ?sub liquid ?can)) ZERO))
  Relations ((allow empty containers)
             (equal-to (A (fluid-level ?can)) (A (bottom-height ?can)))
             (qprop (fluid-level ?can) (amount-of-in ?sub liquid ?can))))

(rule :in (((allow empty containers) . :FALSE))
      (adb::rassert! ((distinguish empty containers) . :TRUE)))

(defview (Evacuated ?can)
  Individuals ((?can :type closed-container
                     :conditions (not (full ?can))
                                 (distinguish evacuated containers))
               (?sub :type substance))
  QuantityConditions ((equal-to (A (amount-of-in ?sub gas ?can)) ZERO)
                      (not (equal-to (A (volume (C-S ?sub LIQUID ?can)))
                                     (A (volume ?can)))))
  Relations ((allow evacuated containers)
             (equal-to (A (pressure ?can)) zero)
             (qprop (pressure ?can) (amount-of-in ?sub gas ?can))))

(rule :in (((allow evacuated containers) . :FALSE))
      (rassert! ((distinguish evacuated containers) . :TRUE)))

(defview (Full ?can)
  Individuals ((?can ;:type container
                     :conditions (container ?can)
                                 (distinguish full containers))
               (?sub :type substance)
               (?cl :bind (C-S ?sub LIQUID ?can)))
  QuantityConditions ((equal-to (A (volume ?cl)) (A (volume ?can))))
  Relations ((allow full containers)))

(rule :in (((allow full containers) . :false))
      (rassert! ((distinguish full containers) . :TRUE)))
```



```
;;; Rules for defining volume of a contained-gas
(rule :intern (((Contained-Gas ?C-G) . :TRUE)
               ((?C-G CONTAINER ?can) . :TRUE)
               ((?C-G SUBSTANCE ?sub) . :TRUE)
               ((Empty ?can) . :TRUE))
  (rjustify ((gas-only ?C-G ?sub ?can) . :TRUE)
            (((contained-gas ?C-G) . :TRUE)
             ((empty ?can) . :TRUE))
            :EMPTY-CAN)
  (rjustify ((gas-only ?C-G ?sub ?can) . :FALSE)
            (((contained-gas ?C-G) . :FALSE))
            :OUT-OF-GAS)
  (rjustify ((gas-only ?C-G ?sub ?can) . :FALSE)
            (((empty ?can) . :FALSE))
            :NOT-EMPTY))

(defpredicate (gas-only ?C-G ?sub ?can)
  (equal-to (A (volume ?C-G)) (A (volume ?can)))
; (Qprop- (volume ?C-G) (amount-of-in ?sub liquid ?can)) This line was
  )       ; commented out so volume of gas can be directly influenced.

(rule :intern (((Contained-Gas ?C-G) . :TRUE)
                ((?C-G CONTAINER ?can) . :TRUE)
                ((Contained-Liquid ?C-L) . :TRUE)
                ((?C-L CONTAINER ?can) . :TRUE))
  (rjustify ((Gas-And-Liquid ?C-G ?C-L ?can) . :TRUE)
            (((contained-gas ?C-G) . :TRUE)
             ((contained-liquid ?C-L) . :TRUE))
            :GAS-AND-LIQUID)
  (rjustify ((Gas-And-Liquid ?C-G ?C-L ?can) . :FALSE)
            (((contained-gas ?C-G) . :FALSE))
            :OUT-OF-GAS)
  (rjustify ((Gas-And-Liquid ?C-G ?C-L ?can) . :FALSE)
            (((contained-liquid ?C-L) . :FALSE))
            :OUT-OF-LIQUID))

(defpredicate (gas-and-liquid ?C-G ?C-L ?can)
   (+Qrel (volume ?can) (volume ?C-G) (volume ?C-L)))
; Why not: ( Q= (volume ?C-G) (- (volume ?can) (volume ?C-L))) ????
; ANSWER: So volume of gas can be directly influenced. Needed for Resolving Ratios.

;;; Rule to deduce temperature of newly-formed contained-stuff:
(rule :intern (((contained-stuff ?C-S) . :TRUE))
  (rjustify ((s (d (temperature ?C-S))) . 0)
            (((equal-to (A (amount-of ?C-S)) ZERO) . :TRUE))
            :INITIAL-TEMPERATURE-LAW))
```

```
;;; -*- Mode: Lisp; Syntax: Common-lisp; Package: QPE -*-

;;;; Domain information for PORTALS:

;;; Quantity types:

(defQuantity-Type Fluid-level Individual)
(defQuantity-Type Max-Height Individual)
(defQuantity-Type Bottom-Height Individual)
(defQuantity-Type Top-height Individual)
(defQuantity-Type Height Individual)
;(defQuantity-Type Submerged-Depth Individual)
(defQuantity-Type Head Individual)

;;; Define Portals:
(defentity Portal
  (quantity (temperature ?self))
  (quantity (pressure ?self))
  (quantity (head ?self))
  (quantity (height ?self)))

(defpredicate (Portal-of ?portal ?can)
  (portal ?portal)
  (container ?can)
  (not (greater-than (A (bottom-height ?can)) (A (height ?portal))))
  (not (less-than (A (top-height ?can)) (A (height ?portal))))
  (Q= (head ?portal) (head ?can)))

(defpredicate (Dry-Portal-of ?portal ?can)
  (Q= (pressure ?portal) (pressure ?can)))

(defpredicate Submerged-Portal
  (Q= (pressure ?self) (- (head ?self) (height ?self))))

(Rule :intern (((dry-portal ?portal) . :TRUE)
               ((portal-of ?portal ?can) . :TRUE))
      (rjustify ((Dry-Portal-of ?portal ?can) . :TRUE)
                (((dry-portal ?portal) . :TRUE))
                :DRY-PORTAL)
      (rjustify ((Dry-Portal-of ?portal ?can) . :FALSE)
                (((dry-portal ?portal) . :FALSE))
                :NOT-DRY-PORTAL))
```



```
;; Install function-specs between portals sharing a path (or a container?):
;(adb:rule (((Portal-of ?portal ?can) . :TRUE))
;  (rassert! ((function-spec-pred ?can ?portal) . :TRUE)))

(adb:rule :in (((fluid-connection ?F-P ?p1 ?p2) . :TRUE))
  (rassert! ((generic-fluid-connection ?F-P ?p1 ?p2) . :TRUE)))

(adb:rule :in (((pump (pump ?p1 ?p2)) . :TRUE))
  (let ((?name (intern (format nil "PUMP-FROM-~A-TO-~A" ?p1 ?p2))))
    (rassert! ((generic-fluid-connection ?name ?p1 ?p2) . :TRUE))))

(adb:rule :in (((compressor (comp ?p1 ?p2)) . :TRUE))
  (let ((?name (intern (format nil "COMPRESSOR-FROM-~A-TO-~A" ?p1 ?p2))))
    (rassert! ((generic-fluid-connection ?name ?p1 ?p2) . :TRUE))))

'''(adb:rule (((function-spec-pred ?name ?can) . :TRUE))
  (multiple-value-bind (ignore ?full-name ?value ?arguments)
      (parse-explicit-function
        '(,(intern (format nil "CAN-HEAD-FUN-~A" ?name))
          (Qprop (head ,?can) (fluid-level ,?can))
          (Qprop (head ,?can) (pressure ,?can))))
    (rassert! ((has-function ?can ?full-name) . :TRUE))
    (eval '(install-explicit-function-specs ',?full-name ',?value ',?arguments))))

;; Install function-specs between containers sharing a path:
(rule :intern (((generic-fluid-connection ?name ?p1 ?p2) . :TRUE)
               ((Portal-of ?p1 ?can1) . :TRUE)
               ((Portal-of ?p2 ?can2) . :TRUE))
      ;; Install "function-specs" between the heads for two connected containers:
  (rassert! ((RCorrespondence
              ((A (head ?can1)) (A (head ?can2)))
              ((A (fluid-level ?can1)) (A (fluid-level ?can2)))
              ((A (pressure ?can1)) (A (pressure ?can2)))) . :TRUE))
  (rassert! ((RCorrespondence
              ((D (head ?can1)) (D (head ?can2)))
              ((D (fluid-level ?can1)) (D (fluid-level ?can2)))
              ((D (pressure ?can1)) (D (pressure ?can2)))) . :TRUE))
   ;; Now do the same for the pressures of the two portals:
  (rassert! ((RCorrespondence
              ((A (pressure ?p1)) (A (pressure ?p2)))
              ((A (head ?p1)) (A (head ?p2)))
              ((A (height ?p1)) (A (height ?p2)))) . :TRUE))
  (rassert! ((RCorrespondence   ;heights never change.
              ((D (pressure ?p1)) (D (pressure ?p2)))
              ((D (head ?p1)) (D (head ?p2)))) . :TRUE)))
```

```
;;; Views for Portals:
(defview (Submerged-Portal ?portal)
  Individuals ((?can :type container)
               (?c-l :type contained-liquid
                     :form (C-S ?sub LIQUID ?can))
               (?portal :type portal
                        :conditions (portal-of ?portal ?can)
                                    (distinguish submerged portals)))
  QuantityConditions ((greater-than (A (level ?c-l)) (A (height ?portal))))
  Relations ((Q= (fluid-level ?portal) (level ?c-l))))

(defview (Not-Quite-Submerged-Portal ?portal)
  Individuals ((?can :type container)
               (?c-l :type contained-liquid
                     :form (C-S ?sub LIQUID ?can))
               (?portal :type portal
                        :conditions (portal-of ?portal ?can)
                                    (distinguish submerged portals)))
  QuantityConditions ((not (greater-than (A (level ?c-l)) (A (height ?portal)))))
  Relations ((Q= (fluid-level ?portal) (level ?c-l))))

(defview (Very-Dry-Portal ?portal)
  Individuals ((?can :type container)
               (?portal :type portal
                        :conditions (portal-of ?portal ?can)
                                    (distinguish submerged portals)))
  QuantityConditions ((equal-to (A (amount-of-in water liquid ?can)) ZERO))
  Relations ((equal-to (A (fluid-level ?portal)) (A (bottom-height ?can)))))

;;; These two rules replace the three views above. Only good for portals at the top or bottom of the
can.
(Rule :intern (((distinguish submerged portals) . :TRUE)
               ((portal ?portal) . :TRUE)
               ((container ?can) . :TRUE)
               ((portal-of ?portal ?can) . :TRUE))
      (rjustify ((Dry-Portal ?portal) . :TRUE)
                (((Very-Dry-Portal ?portal) . :TRUE))
                :BONE-DRY)
      (rjustify ((Dry-Portal ?portal) . :TRUE)
                (((Not-Quite-Submerged-Portal ?portal) . :TRUE))
                :SURFS-UP)
      (rjustify ((Dry-Portal ?portal) . :FALSE)
                (((Not-Quite-Submerged-Portal ?portal) . :FALSE)
                 ((Very-Dry-Portal ?portal) . :FALSE))
                :MUST-BE-WET-BY-NOW))
```



```
(Rule :intern (((distinguish submerged portals) . :FALSE)
               ((portal ?portal) . :TRUE)
               ((container ?can) . :TRUE)
               ((portal-of ?portal ?can) . :TRUE)
               ((substance ?sub) . :TRUE))
      (rjustify ((Dry-Portal ?portal) . :TRUE)
                (((Empty ?can) . :TRUE))
                :NO-LIQUID)
      (rjustify ((Dry-Portal ?portal) . :TRUE)
                (((equal-to (a (height ?portal)) (a (top-height ?can))) . :TRUE)
                 ((Full ?can) . :FALSE))
                :NOT-FULL)
      (rjustify ((Dry-Portal ?portal) . :FALSE)
                (((Submerged-Portal ?portal) . :TRUE))
                :WET-NOT-DRY)
      (rjustify ((Submerged-Portal ?portal) . :TRUE)
                (((equal-to (a (bottom-height ?can))
                            (a (height ?portal))) . :TRUE)
                 ((greater-than (a (amount-of-in ?sub LIQUID ?can))
                                ZERO) . :TRUE))
                :SUBMERGED)
      (rjustify ((Submerged-Portal ?portal) . :TRUE)
                (((Full ?can) . :TRUE))
                :FULL-CAN)
      (rjustify ((Submerged-Portal ?portal) . :FALSE)
                (((Dry-Portal ?portal) . :TRUE))
                :DRY-NOT-WET)
      (rnogood (((Submerged-Portal ?portal) . :FALSE)
                ((Dry-Portal ?portal) . :FALSE))
               :WET-XOR-DRY))
```

```
;;;; -*- Mode: Lisp; Syntax: Common-lisp; Fonts: CPTFONT,TR12I; Package: QPE -*-

 ;;;; Domain theory for Liquid Flow:

 ;;; USES: Physob, Contained-stuff, Portals.

 ;;; Quantities:

 ;; RATES:
(defQuantity-Type Flow-Rate Individual)
(defQuantity-Type Heat-Flow-Rate Individual)
 ;; LEVELS:
(defQuantity-Type Max-Height Individual)
 ;; COEFFICIENTS:
(defQuantity-Type Conductance Individual)

 ;;; Define fluid path:
(defentity Fluid-Path
  (quantity (max-height ?self))
  (quantity (conductance ?self))
  (not (less-than (A (conductance ?self)) ZERO)))

(rule :INTERN (((fluid-connection ?path ?port1 ?port2) . :TRUE)
               ((portal-of ?port1 ?can1) . :TRUE)
               ((portal-of ?port2 ?can2) . :TRUE))
   (rassert! ((container-path ?can1 ?path) . :TRUE))
   (rassert! ((container-path ?can2 ?path) . :TRUE)))
```



```
;;;; Process vocabulary
(defprocess (Liquid-flow ?src-port ?dst-port ?path)
  Individuals ((?src-port :type submerged-portal)
               (?src-can :type container
                         :conditions (portal-of ?src-port ?src-can))
               (?dst-port :type portal)
               (?dst-can :type container
                         :conditions (portal-of ?dst-port ?dst-can))
               (?path :type Fluid-Path
                      :conditions (Fluid-Connection ?path ?src-port ?dst-port))
               (?sub :type substance)
               (?src-cl :type contained-liquid
                        :form (C-S ?sub LIQUID ?src-can))
               (?dst-cl :bind (C-S ?sub LIQUID ?dst-can)))
  Preconditions ((aligned ?path))
  QuantityConditions ((greater-than (A (head ?src-port))
                                    (A (head ?dst-port))))
  Relations ((quantity flow-rate)
             (quantity heat-flow-rate)
             (Q= flow-rate (- (head ?src-port) (head ?dst-port)))
             (Q= heat-flow-rate (*+ flow-rate (temperature ?src-cl)))
             (Q= (temperature ?src-port) (temperature ?src-cl))
             (Q= (temperature ?dst-port) (temperature ?src-port)))
  Influences ((I- (Amount-of-in ?sub liquid ?src-can) (A flow-rate))
              (I+ (Amount-of-in ?sub liquid ?dst-can) (A flow-rate))
              (I- (heat ?src-cl) (A heat-flow-rate))
              (I+ (heat ?dst-cl) (A heat-flow-rate))))
```

```
(defview (Same-temp-flow ?lf)
  Individuals ((?src-cl :type contained-liquid
                        :form (C-S ?sub LIQUID ?src-can))
               (?dst-cl :type contained-liquid
                        :form (C-S ?sub LIQUID ?dst-can))
               (?lf :type (process-instance liquid-flow)
                    :conditions (?lf SRC-CL ?src-cl)
                                (?lf DST-CL ?dst-cl)
                                (distinguish flow temperatures)))
  QuantityConditions ((active ?lf)
                      (equal-to (A (temperature ?src-cl))
                                (A (temperature ?dst-cl)))))

(defview (Hot-to-Cold-flow ?lf)
  Individuals ((?src-cl :type contained-liquid
                        :form (C-S ?sub LIQUID ?src-can))
               (?dst-cl :type contained-liquid
                        :form (C-S ?sub LIQUID ?dst-can))
               (?lf :type (process-instance liquid-flow)
                    :conditions (?lf SRC-CL ?src-cl)
                                (?lf DST-CL ?dst-cl)
                                (distinguish flow temperatures)))
  QuantityConditions ((active ?lf)
                      (greater-than (A (temperature ?src-cl))
                                    (A (temperature ?dst-cl))))
  Relations ((allow temperature differences)))

(defview (Cold-to-Hot-flow ?lf)
  Individuals ((?src-cl :type contained-liquid
                        :form (C-S ?sub LIQUID ?src-can))
               (?dst-cl :type contained-liquid
                        :form (C-S ?sub LIQUID ?dst-can))
               (?lf :type (process-instance liquid-flow)
                    :conditions (?lf SRC-CL ?src-cl)
                                (?lf DST-CL ?dst-cl)
                                (distinguish flow temperatures)))
  QuantityConditions ((active ?lf)
                      (less-than (A (temperature ?src-cl))
                                 (A (temperature ?dst-cl))))
  Relations ((allow temperature differences)))

(rule :in (((allow temperature differences) . :FALSE))
      (rassert! ((distinguish flow temperatures) . :TRUE)))
```



```
;;; -*- Mode: Lisp; Syntax: Common-lisp; Package: QPE -*-

;;;; Domain theory for Pumped Liquid Flow:
;;; USES: Physob, Contained-stuff, Portals.

;;; Quantities:
(defQuantity-Type Max-Head Individual)
(defQuantity-Type Max-Flow Individual)
(defQuantity-Type Flow-Rate Individual)
(defQuantity-Type Heat-Flow-Rate Individual)

;;; Entities and predicates:
(defpredicate (Pump (pump ?src-port ?dst-port))
  (Q= (head (pump ?src-port ?dst-port))
      (- (head ?dst-port) (head ?src-port)))
  (((Pump ?src-port ?dst-port) SRC-PORT ?src-port) . :TRUE)
  (((Pump ?src-port ?dst-port) DST-PORT ?dst-port) . :TRUE))

(defentity Pump
  (quantity (max-flow ?self))
  (greater-than (A (max-flow ?self)) ZERO)
  (quantity (max-head ?self))
  (greater-than (A (max-head ?self)) ZERO)
  (quantity (head ?self)))

;;; Views:
(defview (working-pump ?pump)
  Individuals ((?pump :type pump
                      :form (pump ?src-port ?dst-port))
               (?pf :type (Process-instance pumped-flow)
                    :conditions (?pf PUMP ?pump)
                                (distinguish working pump)))
  QuantityConditions ((Active ?pf)
                      (greater-than (A (head ?dst-port))
                                    (A (head ?src-port)))))

(defview (coasting-pump ?pump)
  Individuals ((?pump :type pump
                      :form (pump ?src-port ?dst-port))
               (?pf :type (Process-instance pumped-flow)
                    :conditions (?pf PUMP ?pump)
                                (distinguish working pump)))
  QuantityConditions ((Active ?pf)
                      (less-than (A (head ?dst-port))
                                 (A (head ?src-port))))
  Relations ((allow coasting pump)))
```

```
(rule :in (((allow coasting pump) . :FALSE))
      (reassert! ((distinguish working pump). :TRUE)))

;;; Processes:
(defprocess (Pumped-Flow ?pump)
  Individuals ((?pump :type pump
                      :form (pump ?src-port ?dst-port))
               (?src-can :type container
                         :conditions (portal-of ?src-port ?src-can))
               (?dst-can :type container
                         :conditions (portal-of ?dst-port ?dst-can))
               (?src-cl :type contained-liquid
                        :form (C-S ?sub LIQUID ?src-can))
               (?dst-cl :bind (C-S ?sub LIQUID ?dst-can)))
  QuantityConditions ((greater-than (A (amount-of-in ?sub liquid ?src-can))
                                    ZERO)
                      (less-than (A (head ?pump)) (A (max-head ?pump))))
  Relations ((quantity flow-rate)
             (Quantity heat-flow-rate)
             (Qprop- flow-rate (head ?pump))
             (Ordered-Correspondence
               ((A (max-flow ?pump)) (A flow-rate))
               ((A (head ?pump)) ZERO))
             (Ordered-Correspondence
               ((A flow-rate) ZERO)
               ((A (max-head ?pump)) (A (head ?pump))))
             (Q= (temperature ?src-port) (temperature ?src-cl))
             (Q= (temperature ?dst-port) (temperature ?src-port))
             (Q= heat-flow-rate (** flow-rate (temperature ?src-cl))))
  Influences ((I- (amount-of-in ?sub liquid ?src-can) (A flow-rate))
              (I+ (amount-of-in ?sub liquid ?dst-can) (A flow-rate))
              (I- (Heat ?src-cl) (A heat-flow-rate))
              (I+ (Heat ?dst-cl) (A heat-flow-rate))))
```



```
(defprocess (Losing-Pumped-Flow ?pump)
  Individuals ((?pump :type pump
                      :form (pump ?src-port ?dst-port))
               (?src-can :type container
                         :conditions (portal-of ?src-port ?src-can))
               (?dst-can :type container
                         :conditions (portal-of ?dst-port ?dst-can))
               (?src-cl :type contained-liquid
                        :form (C-S ?sub LIQUID ?src-can))
               (?dst-cl :bind (C-S ?sub LIQUID ?dst-can)))
  QuantityConditions ((greater-than (A (head ?pump))
                                    (A (max-head ?pump))))
  Relations ((quantity flow-rate)
             (Quantity heat-flow-rate)
             (Qprop flow-rate (head ?pump))
             (Ordered-Correspondence
               ((A flow-rate) ZERO)
               ((A (head ?pump)) (A (max-head ?pump))))
             (Q= (temperature ?src-port) (temperature ?src-cl))
             (Q= (temperature ?dst-port) (temperature ?src-port))
             (Q= heat-flow-rate (** flow-rate (temperature ?src-cl))))
  Influences ((I- (amount-of-in ?sub liquid ?src-can) (A flow-rate))
              (I+ (amount-of-in ?sub liquid ?dst-can) (A flow-rate))
              (I- (Heat ?src-cl) (A heat-flow-rate))
              (I+ (Heat ?dst-cl) (A heat-flow-rate))))
```

```
(defview (pumping-same-temp ?pf)
  Individuals ((?src-cl :type contained-liquid
                        :form (C-S ?sub LIQUID ?src-can))
               (?dst-cl :type contained-liquid
                        :form (C-S ?sub LIQUID ?dst-can))
               (?pf :type (process-instance pumped-flow)
                    :conditions (?pf SRC-CL ?src-cl)
                                (?pf DST-CL ?dst-cl)
                                (distinguish flow temperatures)))
  QuantityConditions ((active ?pf)
                      (equal-to (A (temperature ?src-cl))
                                (A (temperature ?dst-cl)))))

(defview (pumping-hot-to-cold ?pf)
  Individuals ((?src-cl :type contained-liquid
                        :form (C-S ?sub LIQUID ?src-can))
               (?dst-cl :type contained-liquid
                        :form (C-S ?sub LIQUID ?dst-can))
               (?pf :type (process-instance pumped-flow)
                    :conditions (?pf SRC-CL ?src-cl)
                                (?pf DST-CL ?dst-cl)
                                (distinguish flow temperatures)))
  QuantityConditions ((active ?pf)
                      (greater-than (A (temperature ?src-cl))
                                    (A (temperature ?dst-cl))))
  Relations ((allow temperature differences)))

(defview (pumping-cold-to-hot ?pf)
  Individuals ((?src-cl :type contained-liquid
                        :form (C-S ?sub LIQUID ?src-can))
               (?dst-cl :type contained-liquid
                        :form (C-S ?sub LIQUID ?dst-can))
               (?pf :type (process-instance pumped-flow)
                    :conditions (?pf SRC-CL ?src-cl)
                                (?pf DST-CL ?dst-cl)
                                (distinguish flow temperatures)))
  QuantityConditions ((active ?pf)
                      (less-than (A (temperature ?src-cl))
                                 (A (temperature ?dst-cl))))
  Relations ((allow temperature differences)))

(rule :in (((allow temperature differences) . :FALSE))
      (rassert! ((distinguish flow temperatures) . :TRUE)))
```



```
;;; -*- Mode: Lisp Package: USER; Syntax: Common-lisp; Package: QPE -*-

;;;; Domain theory for Gas Flow:

;;; USES: Physob, Contained-stuff, Portals.

;;; Quantities:

;; RATES:
(defQuantity-Type Flow-Rate Individual)
(defQuantity-Type Heat-Flow-Rate Individual)

;;; Entities:
(defentity Fluid-Path
  (quantity max-height ?self)
  (quantity heat ?self)
  (equal-to (D (heat ?self)) ZERO)
  (quantity temperature ?self)
  (greater-than (A (temperature ?self)) ZERO))
```

```
;;;; Process vocabulary:
;;;----------------------------------------------------------------------
(defprocess (Gas-flow ?src-port ?dst-port)
  Individuals ((?src-port :type dry-portal
                          :conditions (portal-of ?src-port ?src-can))
               (?dst-port :type dry-portal
                          :conditions (portal-of ?dst-port ?dst-can))
               (?path :type Fluid-Path
                      :conditions (Fluid-Connection ?path ?src-port ?dst-port))
               (?sub :type substance)
               (?src-cg :type contained-gas
                        :form (C-S ?sub GAS ?src-can))
               (?dst-cg :bind (C-S ?sub GAS ?dst-can)))
  Preconditions ((aligned ?path))
  QuantityConditions ((greater-than (A (pressure ?src-port))
                                    (A (pressure ?dst-port))))
  Relations ((Quantity flow-rate)
             (Quantity heat-flow-rate)
             (Quantity temperature)
             (Q= flow-rate (- (pressure ?src-port) (pressure ?dst-port)))
             (Q= (temperature ?src-port) (temperature ?src-cg))
             (Qprop (temperature ?dst-port) (temperature ?src-port))
             ;(Qprop- (temperature ?dst-port) flow-rate)
             (less-than (A (temperature ?dst-port)) (A (temperature ?src-port)))
             (Q= heat-flow-rate (*+ flow-rate temperature))
             (greater-than (A temperature) (A (temperature ?src-port)))
             (Qprop temperature (temperature ?src-port)))
  Influences ((I- (Amount-of-in ?sub gas ?src-can) (A flow-rate))
              (I+ (Amount-of-in ?sub gas ?dst-can) (A flow-rate))
              (I- (Heat ?src-cg) (A heat-flow-rate))
              (I+ (Heat (C-S ?sub GAS ?dst-can)) (A heat-flow-rate))))
```



```
;;; -*- Mode: Lisp Package: QPE; Syntax: Common-lisp; -*-

;;;; Domain theory for Compressed Gas Flow:

;;; USES: Physob, Contained-stuff, Portals.

;;; Quantities:
(defQuantity-Type Flow-Rate Individual)
(defQuantity-Type pressure Individual)
(defQuantity-Type Max-Pressure Individual)

;;; Entities and Predicates:
(defentity Compressor
  (quantity (max-pressure ?self))
  (greater-than (A (max-pressure ?self)) ZERO)
  (quantity (pressure ?self)))

(defpredicate (Compressor (comp ?src-port ?dst-port))
  (Q= (pressure (comp ?src-port ?dst-port))
      (- (pressure ?dst-port)
         (pressure ?src-port)))
  (((Comp ?src-port ?dst-port) SRC-PORT ?src-port) . :TRUE)
  (((Comp ?src-port ?dst-port) DST-PORT ?dst-port) . :TRUE))

(defview (working-compressor ?comp)
  Individuals ((?src-port :type portal)
               (?dst-port :type portal)
               (?comp :type compressor
                      :form (comp ?src-port ?dst-port))
               (?cf :type (Process-instance compressor-flow)
                    :conditions (?cf CONP ?comp)))
  QuantityConditions ((Active ?cf)
                      (greater-than (A (pressure ?dst-port))
                                    (A (pressure ?src-port)))))

(defview (coasting-compressor ?comp)
  Individuals ((?src-port :type portal)
               (?dst-port :type portal)
               (?comp :type compressor
                      :form (comp ?src-port ?dst-port))
               (?cf :type (Process-instance compressor-flow)
                    :conditions (?cf CONP ?comp)))
  QuantityConditions ((Active ?cf)
                      (less-than (A (pressure ?dst-port))
                                 (A (pressure ?src-port)))))
```

```
;;;; Process vocabulary:
;;;----------;-----------------------------------------------------------------
(defprocess (Compressor-flow ?comp)
  Individuals ((?src-port :type dry-portal
                          :conditions (portal-of ?src-port ?src-can))
               (?dst-port :type dry-portal
                          :conditions (portal-of ?dst-port ?dst-can))
               (?comp :type Compressor
                      :form (comp ?src-port ?dst-port))
               (?sub :type substance)
               (?src-cg :type contained-gas
                        :form (C-S ?sub GAS ?src-can))
               (?dst-cg :bind (C-S ?sub GAS ?dst-can)))
  QuantityConditions ((less-than (A (pressure ?comp))
                                 (A (max-pressure ?comp))))
  Relations ((Quantity flow-rate)
             (Quantity heat-flow-rate)
             (Q= flow-rate (- (max-pressure ?comp) (pressure ?comp)))
             (Q= heat-flow-rate (*+ (flow-rate ?cf) (temperature ?src-cg))))
  Influences ((I- (Amount-of-in ?sub GAS ?src-can) (A flow-rate))
              (I+ (Amount-of-in ?sub GAS ?dst-can) (A flow-rate))
              (I+ (Heat (C-S ?sub GAS ?dst-can)) (A heat-flow-rate))
              (I- (Heat ?src-cg) (A heat-flow-rate))))
```



```
;;; -*- Mode: Lisp Package: QPE; Syntax: Common-lisp; -*-

;;;; Domain theory for Boiling:

;;; USES: Physob, Contained-stuff.

;;; Quantities:

(defQuantity-Type Generation-Rate Individual)
(defQuantity-Type Heat-Flow-Rate Individual)
(defQuantity-Type Tempressure Individual)
(defQuantity-Type Temp-Diff Individual)

;;;; Process vocabulary:
(defprocess (Boiling ?C-L)
  Individuals ((?sub :type substance)
               (?can :type container
                     :conditions (boiling-allowed-in ?can))
               (?C-L :type Contained-Liquid
                     :form (C-S ?sub LIQUID ?can))
               (?C-G :bind (C-S ?sub GAS ?can)))
  QuantityConditions ((greater-than (A (temperature ?C-L)) (A (tboil ?C-L))))
  Relations ((quantity generation-rate)
             (quantity temp-diff)
             (quantity Heat-Flow-Rate)
             (quantity Tempressure)
             (Q= temp-diff (- (temperature ?C-L) (tboil ?C-L)))
             (Q= generation-rate (*+ temp-diff (amount-of ?C-L)))
             (Qprop Tempressure (temperature ?C-L))
             (greater-than (A Tempressure) (A (temperature ?C-L)))
             (not (less-than (A Tempressure) (A (temperature ?C-G))))
             (not (equal-to (A Tempressure) (A (temperature ?C-G))))
             (greater-than (A Tempressure) (A (pressure ?C-G)))
             (Q= Heat-Flow-Rate (*+ (generation-rate ?boil) Tempressure)))
  Influences ((I- (Amount-of-in ?sub liquid ?can) (A generation-rate))
              (I+ (Amount-of-in ?sub gas ?can) (A generation-rate))
              (I- (heat ?C-L) (A Heat-Flow-Rate))
              (I+ (heat ?C-G) (A Heat-Flow-Rate))
              (I+ (volume ?C-G) (A (generation-rate ?boil)))))
```

```
(defprocess (Condensation ?C-G)
  Individuals ((?sub :type substance)
               (?can :type container
                     (condensation-allowed-in ?can))
               (?C-G :type contained-gas
                     :form (C-S ?sub GAS ?can))
               (?C-L :bind (C-S ?sub LIQUID ?can)))
  QuantityConditions ((less-than (A (temperature ?C-G)) (A (tboil ?C-G))))
  Relations ((quantity generation-rate)
             (quantity temp-diff)
             (quantity Heat-Flow-Rate)
             (quantity Tempressure)
             (Q= temp-diff (- (tboil ?C-G) (Temperature ?C-G)))
             (Q= generation-rate (** Temp-Diff (Mass ?C-G)))
             (Qprop Tempressure (Temperature ?C-G))
             (less-than (A Temperessure) (A (temperature ?C-G)))
             (less-than (A Temperessure) (A (temperature ?C-L)))
             (less-than (A Temperessure) (A (pressure ?C-L)))
             (Q= Heat-Flow-Rate (** (generation-rate ?boil) Temperessure)))
  Influences ((I- (Amount-of-in ?sub gas ?can) (A generation-rate))
              (I+ (Amount-of-in ?sub liquid ?can) (A generation-rate))
              (I- (heat ?C-G) (A Heat-Flow-Rate))
              (I+ (heat ?C-L) (A Heat-Flow-Rate))
              (I- (volume ?C-L) (A (generation-rate ?boil))))))
```



```
;; Rule for HEAT-CONNECTION given TOUCHES:
(adb:rule :intern (((heat-path ?path) . :TRUE)
                   ((container ?can) . :TRUE)
                   ((contained-stuff ?C-S) . :TRUE)
                   ((?C-S CONTAINER ?can) . :TRUE)
                   ((Heat-Connection ?path ?src (?part ?can)) . :TRUE))
  (rjustify ((Heat-Connection ?path ?src ?C-S) . :TRUE)
            (((heat-path ?path) . :TRUE)
             ((contained-stuff ?C-S) . :TRUE)
             ((?C-S CONTAINER ?can) . :TRUE)
             ((Heat-Connection ?path ?src (?part ?can)) . :TRUE)
             ((Touches ?C-S (?part ?can)) . :TRUE)))
  (rjustify ((Heat-Connection ?path ?src ?C-S) . :FALSE)
            (((heat-path ?path) . :TRUE)
             ((contained-stuff ?C-S) . :TRUE)
             ((?C-S CONTAINER ?can) . :TRUE)
             ((Heat-Connection ?path ?src (?part ?can)) . :TRUE)
             ((Touches ?C-S (?part ?can)) . :FALSE))))

; Now go the other way; from the can out.
(adb:rule :intern (((heat-path ?path) . :TRUE)
                   ((container ?can) . :TRUE)
                   ((contained-stuff ?C-S) . :TRUE)
                   ((?C-S CONTAINER ?can) . :TRUE)
                   ((Heat-Connection ?path (?part ?can) ?dst) . :TRUE))
  (rjustify ((Heat-Connection ?path ?C-S ?dst) . :TRUE)
            (((heat-path ?path) . :TRUE)
             ((contained-stuff ?C-S) . :TRUE)
             ((?C-S CONTAINER ?can) . :TRUE)
             ((Heat-Connection ?path (?part ?can) ?dst) . :TRUE)
             ((Touches ?C-S (?part ?can)) . :TRUE)))
  (rjustify ((Heat-Connection ?path ?C-S ?dst) . :FALSE)
            (((heat-path ?path) . :TRUE)
             ((contained-stuff ?C-S) . :TRUE)
             ((?C-S CONTAINER ?can) . :TRUE)
             ((Heat-Connection ?path (?part ?can) ?dst) . :TRUE)
             ((Touches ?C-S (?part ?can)) . :FALSE))))
```

```
;; Rule for TOUCHES-BOTTOM:
(adb:rule. :intern (((container ?can) . :TRUE)
                    ((substance ?sub) . :TRUE))
  (rjustify ((Touches (C-S ?sub LIQUID ?can) (Bottom ?can)) . :TRUE)
            (((greater-than (A (amount-of-in ?sub LIQUID ?can)) ZERO) . :TRUE)))
;  (rassert ((Touches (C-S ?sub GAS ?can) (Bottom ?can)) . :FALSE))
  (rjustify ((Touches (C-S ?sub GAS ?can) (Bottom ?can)) . :FALSE)
            (((Touches (C-S ?sub LIQUID ?can) (Bottom ?can)) . :TRUE)))
  (rjustify ((Touches (C-S ?sub GAS ?can) (Bottom ?can)) . :TRUE)
            (((equal-to (A (amount-of-in ?sub liquid ?can)) ZERO) . :TRUE)))
  (rjustify ((Touches (C-S ?sub LIQUID ?can) (Bottom ?can)) . :FALSE)
            (((Touches (C-S ?sub GAS ?can) (Bottom ?can)) . :TRUE))))

;; Rule for TOUCHES-TOP:
(adb:rule :intern (((container ?can) . :TRUE)
                   ((substance ?sub) . :TRUE))
  (rjustify ((Touches (C-S ?sub GAS ?can) (Top ?can)) . :TRUE)
            (((greater-than (A (amount-of-in ?sub GAS ?can)) ZERO) . :TRUE)))
  (rjustify ((Touches (C-S ?sub GAS ?can) (Top ?can)) . :FALSE)
            (((equal-to (A (amount-of-in ?sub GAS ?can)) ZERO) . :TRUE)))
  (rassert ((Touches (C-S ?sub liquid ?can) (Top ?can)) . :FALSE)))
```