A Qualitative Model of the Space Shuttle Reaction Control System*

Herbert Kay
bert@cs.utexas.edu

April 8, 1994

Abstract

The Space Shuttle Reaction Control System (RCS) is a component of the space shuttle propulsion system that provides motion control for the shuttle orbiter. As part of a model-based monitoring and diagnosis system for the RCS, a qualitative model of the system was developed using the QSIM modeling language. This report describes the RCS model, the assumptions that went into it, and predictions that can be made with the model.

*This work has taken place at the Artificial Intelligence Research Branch of the NASA Ames Research Center and the Qualitative Reasoning Group at the Artificial Intelligence Laboratory, The University of Texas at Austin. Research of the Qualitative Reasoning Group is supported in part by NSF grants IRI-8905494 and IRI-8904454, by NASA grant NAG 2-507, by the Texas Advanced Research Program under grant no. 003658175, and by the Jet Propulsion Laboratory.

1 Introduction

The Space Shuttle Reaction Control System (RCS) is a component of the space shuttle propulsion system that provides motion control of the orbiter while it is in space. This report is a design document for a qualitative model of the RCS that was constructed in the QSIM modeling language.

The RCS model describes the structure and physical operation of the RCS with respect to propellant movement through the system. It is not concerned with simulating the movement of the spacecraft. This latter process is fairly well understood and is well modeled by conventional software for computing trajectories. A model of propellant movement that is useful for monitoring and diagnosis, however, is more problematic because it must remain accurate even in the presence of an imprecise description of the system (in part because "all plumbing leaks" and in part because the exact fluid path may vary from system to system) and incomplete knowledge about failures. A qualitative model can make accurate predictions (i.e., all real behaviors are predicted) in the face of this incompleteness whereas a purely numerical model, which must rely on precise values, cannot.

Part 2 of this report describes the physical layout and operation of the RCS in both normal and faulty modes. It also discusses important fault scenarios and their detection, isolation, and repair. Part 3 describes the QSIM model in detail with emphasis on the assumptions and design decisions that went into each component. Finally, Part 4 describes the user interface for the simulation model as well as the types of questions that the model is capable of answering. The model and user interface are available from the author at the University of Texas at Austin.

2 RCS Overview

The RCS is a set of maneuvering thrusters used primarily to make attitude adjustments to the space shuttle orbiter. Key tasks include the separation of the shuttle from the external fuel tank after launch and repositioning the shuttle for reentry. It also can serve as a replacement for one of the main engines in the event of main engine failure and can serve as a fuel dump under emergency conditions.

The RCS is composed of three subsystems – the forward, left, and right RCS located in the nose and aft sections of the shuttle (see Figure 1).

Each subsystem consists of a fuel tank, an oxidizer tank, and a set of 14 jets (16 jets for the Forward RCS). Jet usage and firing can be monitored and controlled from the ground, by the on-board computers, or by manually reading gauges and operating valves. Normally, jet firing is controlled by a computer that makes decisions on what jets to fire based on the

Figure 1: Location of RCS subsystems on the shuttle orbiter.

type of attitude adjustment and the jets currently available[1].

2.1 RCS Subsystem Layout and Operation

Figure 2 details the layout of a single RCS subsystem. All three subsystems have a similar structure. A subsystem is composed of two pressurization and propellant feed systems[2] – one for oxidizer and one for fuel. These two systems are independent and connect at the thruster chambers at the ends of each manifold. Each feed system consists of a helium tank (which pressurizes the propellant), a propellant tank, a set of manifolds and thrusters, and a set of regulators and valves that distribute the propellant and deliver it to one or more of the manifolds. Additionally, the left and right subsystems can be cross-connected (a so-called "crossfeed") so that they can share the same fuel and/or oxidizer tanks. They may also be connected to the main engine fuel and/or oxidizer tanks as well (a so-called "interconnect").

Each pressurization and propellant feed system contains temperature and pressure sensors in the helium and propellant tanks and pressure sensors in the manifolds (denoted by the "T" and "P" symbols in Figure 2). These sensors are used to monitor the RCS and assist in fault isolation as well as to calculate the quantity of propellant remaining.

Conceptually, each propellant feed system operates like an aerosol can. Pressure in the helium tank passes through a series of regulators that reduce the pressure to the desired working pressure (242 to 248 psig). This pressure is transferred to the propellant tank which in turn forces the propellant into the tank lines, past any open isolation valves, and into any open manifolds. Finally, if the jet corresponding to a manifold is on, the fuel and oxidizer from the pair of feed systems is pushed into the thrust chamber where they explode on contact and produce thrust.

The system is designed for maximum reconfigurability. Normally, only one of the paths from the helium tank to the propellant tank are open with the second one kept as a spare. Each of these paths has two pressure regulators in series to guard against a single regulator failure. The isolation and manifold valves below the propellant tanks are used to reconfigure around a variety of leaks. Finally, the crossfeed valves permit operation of the system even when there is a catastrophic failure above the tank isolation valves.

To prevent under and over pressures in the propellant tank, a check valve (which prevents backflow into the helium tank) and a relief valve (which prevents the propellant tank pressure from becoming too high) are placed between the regulators and the propellant tank.

Since the primary use of the RCS model is for diagnosis, it is important to understand the main failure modes of the RCS. These failures are outlined in the following paragraphs

[1] Jets may be unavailable due to failures or reconfiguration of the RCS.
[2] In this report, "propellant" refers to either fuel or oxidizer.

Figure 2: Layout of an RCS subsystem. One of the aft subsystems is shown.

(refer to Figure 3).

Leaks The locations of leaks are determined by the location of flow inhibition valves. Important leak locations are at

- The helium tank. A leak here will force the check valve to close. This causes a situation called "blowdown" to occur where the amount of usable propellant in the propellant tank is determined by the remaining pressure in the propellant tank (called "ullage"). If ullage pressure is low, then a potentially large amount of propellant will be trapped in the tank, unable to flow to the thrusters because there will not be enough pressure to push it out[3].

- The helium line. A leak here has similar consequences to a helium tank leak. If the leak is in one of the helium legs (between the primary regulator and the helium tank isolation valve), then that leg can be isolated and helium flow will pass through the other helium leg. If the leak cannot be isolated, then both helium isolation valves can be closed to maintain helium tank pressure. This places the system in a blowdown situation.

- The propellant tank. A leak here requires closing the tank isolation valves. Crossfeed from the other RCS is needed to operate the jets within the subsystem.

- The tank leg. This failure can be configured around, with the loss of fuel to some jets.

- The manifolds. A leak here can be isolated with the loss of only the effected manifold.

- The crossfeed line. A leak here reduces the reconfigurability of the system as crossfeeds cannot be performed.

Regulator failures Regulators can fail open or closed. Because of the series design of the helium line regulators and the fact that the regulators regulate at different pressures, there are three possible series regulator failures: failed closed (either regulator), primary failed open but secondary working, and both primary and secondary failed open.

[3]The thrusters require a minimum pressure differential across them to function. When the manifold pressure is too low (below 180 psig), the jets will no longer operate, even if the manifold is repressurized. This failure is known as jet "starvation." Once a jet has starved, it is unusable for the remainder of the mission.

Figure 3: Location of important leaks in an RCS subsystem.

Other failures These include electrical and computer failures as well as individual valve failures. These failures are not covered in the RCS model.

Monitoring and diagnosis problems can be broken into three phases typically described by the acronym "FDIR." The first phase is detection. Failures are detected by either the mission controllers on the ground, the astronauts, or, as a last resort, by the on-board computer's quantity monitor which detects discrepancies in the amounts of fuel and oxidizer consumed in each system. The second phase is isolation. If the fault is plumbing-related (either a leak or a regulator failure) the following two steps are performed. :

1. "Safe" the system.
   This involves closing valves starting at the thruster and working back to the helium tank. This prevents venting any section of the system unnecessarily.

2. Diagnose the problem.
   By reopening valves, pressure differences can be calculated to isolate the leak.

The third phase is reconfiguration. Based on the source of the problem and the remaining mission objectives, certain portions of the system are cut off from the propellant flow.

3 The QSIM Model

The QSIM model of the RCS was designed with monitoring applications in mind. Because our simulation method is qualitative, it is possible to construct an incompletely specified model (due to lack of knowledge about precise system parameters, for instance), but there is potential for it to generate infinite behavior trees. To be useful for monitoring, however, the model must be tractable so that the number of behaviors is minimized while guaranteeing that all real behaviors are predicted. At odds with tractability is the need for greater detail in the model to ensure adequate fault coverage. A design decision was made to favor tractability over fault coverage in order to get a working and useful model running quickly.

A schematic of the RCS model is shown in Figure 4. Since both propellant feed systems are similar in construction, only one of the systems is modeled. For a complete subsystem model, two of these models would be needed. A key tractability concern with multitank models is that chatter may occur in flows between the tanks, especially when there are leaks. Therefore, effort was made to constrain the flow variables as much as possible. In part, this was done by ignoring the qualitative direction of change and turning off automatic landmark creation of all flows. The model abstracts the two paths from the helium tank to the thrusters into a single path. This simplification also improves model tractability with little impact on fault coverage[4]. In addition, the relief valve and crossfeed connections have

[4]This is because there are no sensors which could distinguish faults on the basis of the helium leg that they occur in.

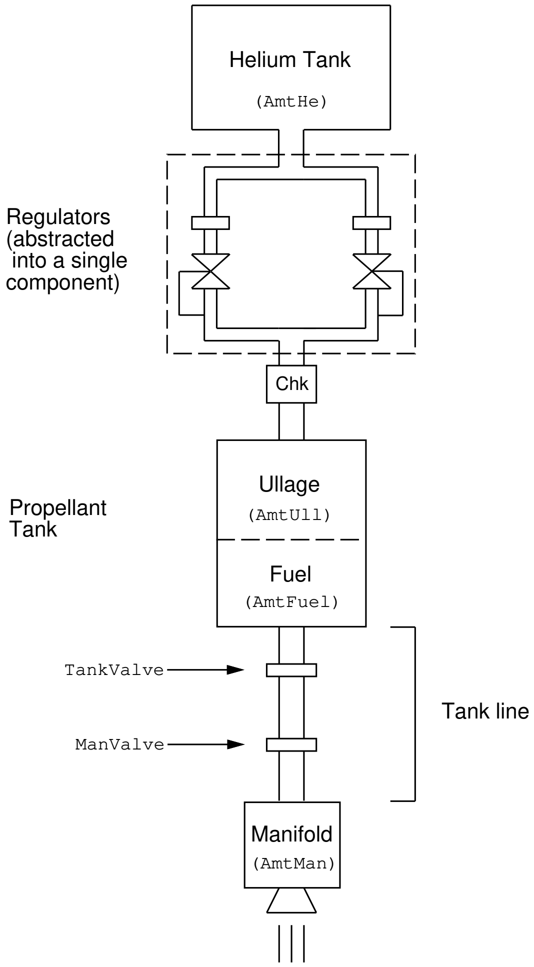


Figure 4: QSIM RCS subsystem model of the fuel pressurization and feed system.

been left out of the model.

To introduce faults (as well as to model the regulator), the moded-constraint and discrete variables QSIM constructs were used. This permits multiple QDEs that differ in only a few constraints to be modeled as a single QDE with "switchable" constraints that depend on mode variables [Farquhar *et al.*, 1991].

The remainder of this section details sections of the RCS model. Refer to Appendix A for the full model.

Helium Tank The Helium Tank is modeled as a simple gas container with a fixed volume[5] and a single outlet. The amount of helium (the QSIM variable `AmtHe`) is a state variable. There are five landmarks in the quantity space of `AmtHe`. These have the following meaning:

- `AmtHeInit` – This is the maximum amount of helium that is present in the tank. This would be what it is filled with before launch.
- `AmtHeSReg` – This is the amount needed to provide a pressure equal to that produced at the output of the secondary regulator in the helium line. It is needed to determine the regulator mode.
- `AmtHePReg` – This is the amount needed to provide a pressure equal to that produced at the output of the primary regulator in the helium line. It is needed to determine the regulator mode.
- `AmtHeMin` – This is the amount needed to provide just enough pressure to keep the thruster from starving.
- `0` – A vacuum. This value should be unreachable.

In reality, the helium tank volume is a function of pressure, and pressure is a function of temperature. Since these effects are reasonably small, they have been left out of the model.

Helium tank leaks are modeled by a conductance `HeLeakS` and the pressure difference between the tank and the vacuum of space (`PDiffHeAndVac`).

Propellant Tank The propellant tank is modeled as a container with a liquid (the propellant, represented by `AmtFuel`) and a gas (the ullage represented by `AmtUll`). Gas enters the tank from the helium line and propellant exits through the tank line. The ullage and propellant are related by the fact that their volume remains a constant. Although the propellant is basically a liquid, it is still compressible, and it is therefore necessary to model its density `DenFuel`. There are four landmarks in the propellant and ullage quantity spaces.

[5] Volume is implicit in this component.

Below is a list of the landmarks for the ullage. The propellant landmarks are similarly named.

- `AmtUllMax` – This is the maximum amount of ullage in the propellant tank. Normally, the ullage will start out small and increase as more propellant is sent to the thrusters.
- `AmtUllNom` – This is the amount of ullage in the propellant tank at the start of the simulation.
- `AmtUllInit` – This is the starting amount of ullage in the propellant tank.
- `0` – This landmark should be unreachable.

There is also a W+ constraint between `PUll` and `nRT[AmtUll]` that acts as the order-of-magnitude constraint which states that the change in `PUll` is dominated by the change in `AmtUll` when the change in `AmtFuel` is small.

Propellant tank leaks are modeled by a conductance `FuelLeakS` and the pressure difference between the tank and vacuum (`PDiffFuelAndVac`) and they act as an additional flow of propellant out of the tank. This is not quite correct in that a leak should also cause depletion of ullage as well. This simplification was made since fuel loss is the more important of the two losses from the standpoint of the manifold.

Flow from Helium to Propellant Tank The flow from the helium tank to the propellant tank is idealized into a single path which is determined by the states of the valves and regulators in both legs. The idea here is that the pressure seen by the propellant tank is independent of the path by which the pressure reaches the tank. Since there is no sensor that can detect a leak in one path versus the other, the distinction of having two paths provides no diagnostic leverage. Including both paths does, however, decrease tractability because of the additional flow.

The path between the helium and propellant tank is further simplified by abstracting the valve and regulator pair (of the single path) into a single component. Lack of diagnostic sensors between the valves and regulators reduces the need to model them individually. This abstract component may then be thought of as a "super regulator" whose behavior is determined by the state of the valve and the fault models for the primary and secondary regulator. This state information is represented by a mode variable called `reg-mode` that selects the proper regulator pressure (called *regpress* below). Its values are:

- `open` – This signifies that both regulators in a helium line leg have failed open. The regulator works like a pipe.

- `closed` – Signifies that both helium isolation valves are closed and/or both regulator series have at least one failed closed regulator. There is no path from the helium tank to the propellant tank.
- `Preg` – Signifies that the regulator is regulating at the primary regulator pressure, assuming adequate pressure in the helium tank.
- `Sreg` – Signifies that the regulator is regulating at the secondary regulator pressure, assuming adequate pressure in the helium tank.

Originally, the abstract regulator was modeled using a second order mechanical model of a regulator a la deKleer and Brown [de Kleer and Brown, 1984]. This model, while tractable by itself, caused the total RCS model to become intractable. It was then replaced by a first-order mechanical model, but this proved intractable as well. In the final model, the regulator is modeled functionally by a set of operating modes. Switching between modes is handled by QSIM operating region transitions. The state of the pressure regulator is represented by

The abstract regulator is implemented as a device with three modes. Mode changes are implemented with QSIM region transitions. The modes of the regulator are defined by the variable `reg-model` as follows:

- `ideal-psource` – The regulator is maintaining the proper output pressure. In this mode, the flow through the regulator is a function of the flow out of the propellant tank. `PUll` is held steady at *regpress* and `PHe` > *regpress*. This is the normal operating mode of the RCS. Once `PHe` drops below *regpress* a region transition to `invisible` takes place.
- `invisible` – This mode is reachable only from `ideal-psource` when the helium pressure drops below *regpress*. In this mode, ullage pressure is monotonically related to the helium tank pressure and the flow through the helium line is equal to the flow out of the ullage. Additionally the pressure difference between the helium and ullage tank is zero. In effect, the helium and propellant tanks become one tank and the pressure regulator is thus invisible.
- `pipe` – In this mode, `PHe` > *regpress* but `PUll` < *regpress*. This situation could arise after a leak in the propellant tank has been repaired. The flow through the helium line is a function of the pressure difference between the helium tank and the ullage. Once `PUll` = *regpress*, the regulator model transitions back to `ideal-psource`.
- `closed` – In this mode, there is no flow through the regulator.

This multi-mode regulator violates the "no function in structure" rule [deKleer and Brown, 1984] since the flow through the regulator depends on the behavior of other components which are not directly connected (namely, the propellant outflow). Since the goal was not to build a truly generic model of a regulator, however, this regulator model is used because it greatly increases the tractability of the complete RCS model.

The moded regulator also introduces continuity problems in the RCS model. As defined, the `invisible` mode is incorrect since even at low pressures, there will be a small pressure difference across the regulator although the model assumes that there is none. This means that leaks in the helium tank which could change helium line flow are improperly modeled as there is no pressure drop across the regulator. This could be corrected by removing the invisible mode and using only the pipe mode, however, the model chatters because of the now underconstrained pressure difference. Since the model is primarily required to operate in the `ideal-psource` mode and there was no design information about behavior outside of this region, it was decided to make this simplification at the expense of some fault coverage.

Check Valve The check valve is modeled by a region transition to `check-valve-closed` whenever there is a backflow in the helium line. This transition sets `reg-mode` to `closed`, thereby isolating the helium tank. In the current model, there is no way to fault the check valve.

Manifold and Thruster The manifold is represented as a simple container of compressible fluid whose input comes from the tank lines and whose output goes to the thruster. It contains one state variable `AmtMan`. The thruster itself is represented as a conductance controlled by `Thruster` and outflow is a function of the pressure difference between the manifold and the vacuum of space. This model is not quite correct, as the thruster flow varies monotonically with thruster chamber pressure which is not modeled. Since the fuel density in the manifold may be different from the density in the propellant tank, the variable `DenFuelMan` is introduced. As with the propellant tank, a W+ constraint is added to guarantee that change in pressure difference across the manifold dominates over the change in fuel density. Leaks in the manifold are modeled by a fixed conductance `ManLeakS` and a pressure difference `PDiffManAndVac`.

Tank Line The tank line consists of a pair of valves represented by the discrete variables `TankValve` and `ManValve` that control whether there is a flow between the tank and manifold. Valve operation is discrete (either `open` or `closed`) and is modeled by a moded conductance constraint on the variable `TIValve` which represents the minimum conductance through both valves. `TankValve` prevents flow in either direction through the tank line, but `ManValve` prevents flow only from the line into the manifold, since backflow is possible through this valve. The tank line is modeled explicitly so that tank line leaks can be represented. A tank line leak gets flows from both the propellant tank and the manifolds depending on which valves are open. The leak is modeled by a fixed conductance `LineLeakS` and the pressure differences `PDiffTankAndVac` and `PDiffManAndVac` defined previously.

4 Interface Description

This section details the user interface to the RCS model and shows some simulation results from a perturbed equilibrium model and a non-equilibrium initial condition.

4.1 Simulation setup using `run-rcs`

From a user standpoint, the state of the RCS model is described by

- The position of the valves in each helium line.
- The fault mode of each regulator in each of the helium lines.
- The position of the tank and manifold valves.
- The state of the thruster.
- The pressures in the helium tank, propellant tank, and manifolds.
- The amount of propellant in the propellant tank.
- The hypothesized set of leaks.

As has been shown in the previous section, the QSIM model abstracts away much of this detail. To bridge this gap between descriptions, a configuration description function is used that maps an RCS user description into a QSIM model. The function `run-rcs` takes three arguments, an initial state (equilibrium or with `PUll` perturbed low), a list of configuration settings, and an initial pressure for the helium tank, and forms a QSIM state perturbed from equilibrium based on the settings[6]. Each setting is one of :

- A helium line descriptor.
  This is a tuple of (*name valve preg sreg*) where *name* is either `leg1` or `leg2`, *valve* describes the valve position for the leg (`open` or `closed`) and *preg* and *sreg* describe the state of the primary and secondary regulators. This state may be `working`, `failed-open`, or `failed-closed`. An unspecified leg defaults to (*leg* `open working working`).

[6] At the present time, the configuration and helium pressure arguments are ignored for the non-equilibrium initial state.

- Positions for the tank and manifold valves.
  These are tuples of (*valve position*) where *valve* is either `tankvalve` or `manvalve` and *position* is either `open` or `closed`. An unspecified valve defaults to `open`.

- Thruster state.
  This is a tuple of the form (`thruster` *state*) where *state* is either `0` or `max`. The default is `0`.

- Leak Locations.
  These are tuples of the form (*location yes-or-no*) where *location* is one of `HeTankLeak?`, `FuelTankLeak?`, `TankLineLeak?`, or `ManLeak?` and *yes-or-no* is one of `yes` or `no`. All leaks default to `no`.

The helium line descriptions are used to compute a single helium line model whose `reg-mode` is determined by the lower conductance path through the two lines. For the `equilibrium` initialization, simulation begins from an equilibrium state determined from a system with no leaks or regulator failures, a closed thruster, and a nominal amount of helium, ullage, and propellant. The keyword argument `:PHe` selects an initial helium tank pressure. The default is `PHeSreg`, i.e., the helium pressure is at the pressure needed to regulate at the secondary regulator pressure. This value is used to shorten the number of simulation steps when running the model for the nominal case of `reg-mode` = `preg`. For the `low-PUll` initial state, simulation begins from a non-equilibrium state where `PUll` is perturbed below its equilibrium value.

4.2 Sample simulation runs

This section describes several runs of the RCS model under a variety of scenarios. In general, these scenarios predict all real behaviors, however, models that assume helium tank leaks may be incorrect when `PHe` < *regpress*. This is because the regulator model is not always accurate when the conditions for the `ideal-psource` model are violated.

Only single faults are simulated in the sample runs. Multiple faults can be simulated by simply defining them in the configuration description. Since the QDE describes only one pressurization and propellant feed system, the simulation runs reflect faults in only one of the fuel or oxidizer systems. Since these systems are independent in terms of material flow, a simulation of an entire RCS subsystem would involve two separate simulations of each pressurization and propellant feed system. Different fault combinations could then be simulated in each subsystem.

- Table 1 describes the faults that are covered by these examples with respect to the set of all faults of interest.

Fault	Coverage
No faults	Covered (Figures 5 and 6)
He Tank leak	Covered (Figure 8)
He Line leak	Inexpressible
Propellant Tank leak	Expressible
Primary Regulator failure (either leg)	Covered (Figure 10)
Secondary Regulator failure (either leg)	Expressible
Check valve failure	Inexpressible
Manifold leak	Expressible
Tank line leak	Covered (Figure 9)
Crossfeed leak	Inexpressible
Lo Ullage Pressure	Covered (Figure 11)

Table 1: Fault coverage of the RCS sample simulation runs. Each entry in the table lists a fault type and whether the fault is covered by an example, expressible but not covered, or inexpressible with the current model. Note that fault combinations are covered if the individual faults are covered.

- Figure 5 show the equilibrium state of the system.

- Figure 6 show the system under normal operation with the thruster on. There are 15 behaviors (see Figure 7) all of which show fuel and helium depleting. The main difference is what gives out first – pressure head or fuel. Note that an ullage pressure of less than `PMin` terminates simulation due to jet starvation.

- Figure 8 shows the effect of a helium leak with the thruster off and a helium tank leak. Here, the check valve prevents ullage from backflowing into the helium tank. There is only one behavior. With the thrusters on, this model is identical to Figure 6. This model is not correct once `PUll` drops below `PUllPreg`, however, because the the transition from `ideal-psource` under the conditions of this scenario is not correct. The problem is that `PUll` will not necessarily follow `PHe` because the check valve will prevent flow from the ullage to the helium tank (which is caused by the helium tank leak). This could be addressed by improving the regulator model (for instance, by adding more modes to `reg-mode`).

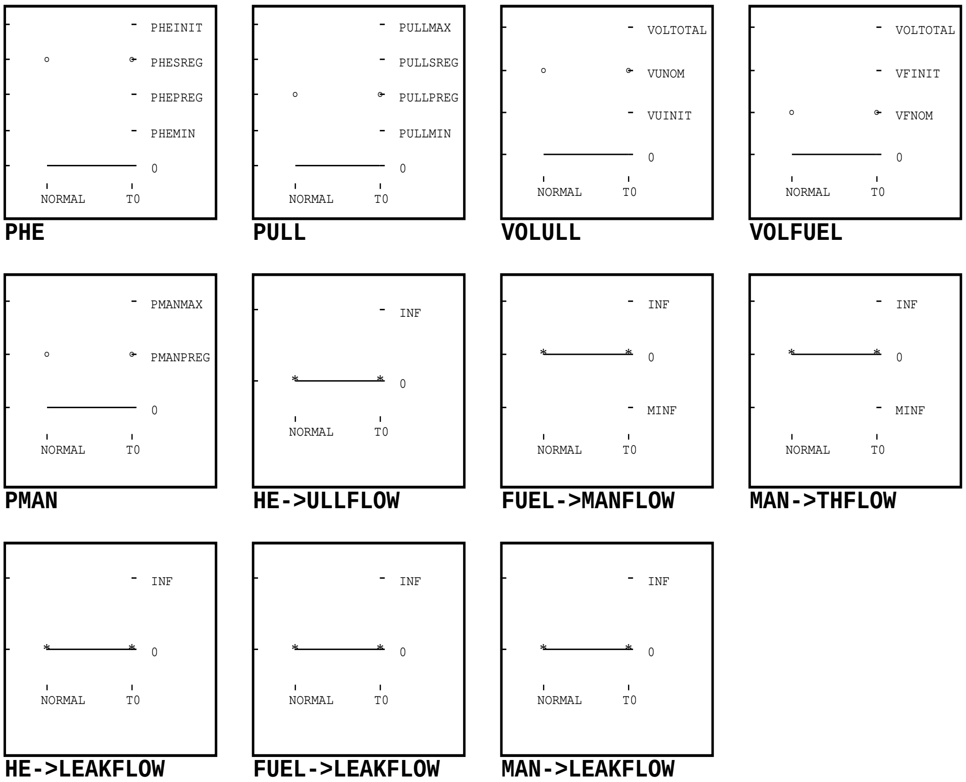
- Figure 9 shows a system with a tank line leak with thrusters off and the manifold closed. Because the manifold valve is a "one-way" valve, the line leak will bleed off the pressure here. Because both the manifold and propellant tanks are feeding the leak, there is chatter in the pressure difference between the tank and manifold. When the tank valve is closed, this chatter disappears.

- Figure 10 shows the effects of the primary regulator failing open. Initially, the regulator acts like a pipe to bring the ullage pressure up to the new regulated pressure `PUllSreg`. The 41 behaviors are distinguished by whether the all fuel is pushed out of the propellant tank or whether the helium tank pressure drops below the secondary regulator pressure. Since the difference between `PUllPreg` and `PUllSreg` is small and the density of the propellant is greater than the density of the helium, quantitative information could rule out many of these behaviors.

- Figure 11 shows the effects of a low pressure in the ullage after a helium line leg is opened. This situation could appear after a leak has been repaired and the system is switched from "safe" to working. There are 33 behaviors. As in the previous example, these are distinguished by by whether the fuel or helium gets pushed out as a result of the sudden change in pressure caused by opening the helium line valve.

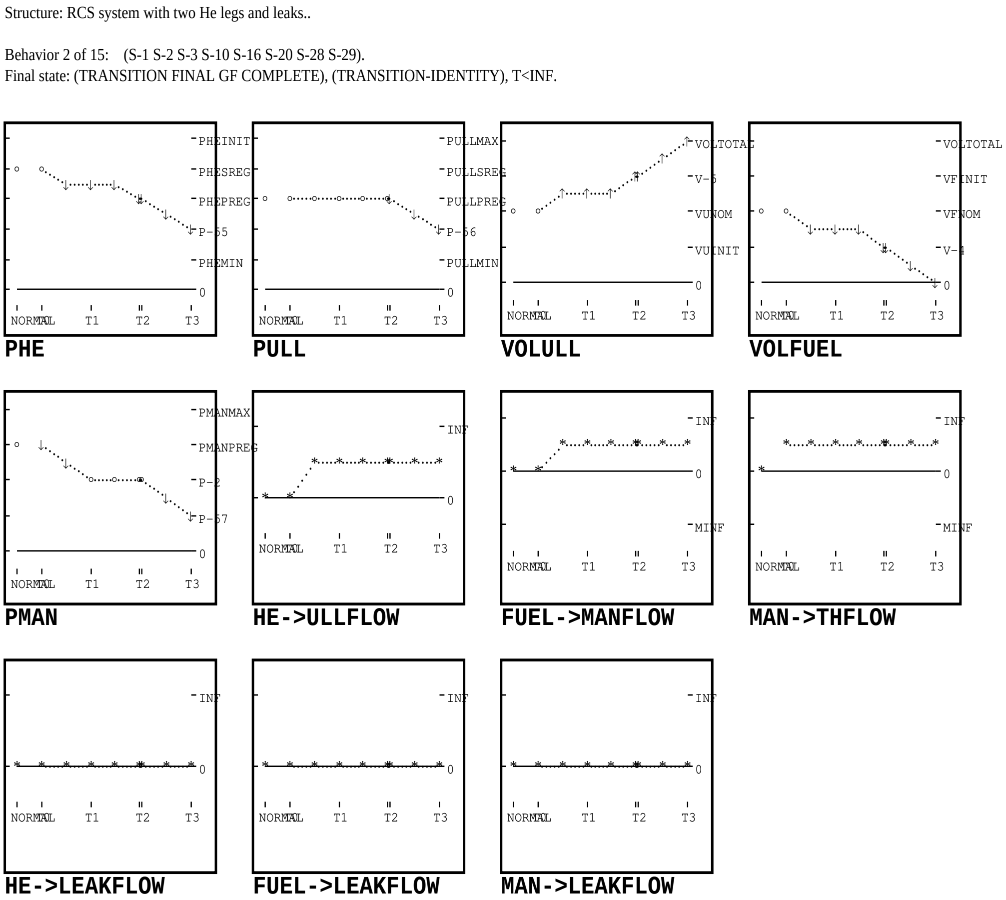
Structure: RCS system with two He legs and leaks..

Behavior 1 of 1: (S-1).
Final state: (GF COMPLETE QUIESCENT), (NIL), NIL.




```
(run-rcs :init 'equilibrium
         :PHe 'PHeSreg
         :configuration '((leg1 closed working working)
                          (leg2 open   working working)
                          (tankvalve open) (manvalve open)
                          (thruster 0)))
```

Figure 5: RCS simulation (and function call) with all components working and thrusters off. This is an equilibrium state.




```
(run-rcs :init 'equilibrium
         :PHe 'PHeSreg
         :configuration '((leg1 closed working working)
                          (leg2 open   working working)
                          (tankvalve open) (manvalve open)
                          (thruster max)))
```

Figure 6: RCS simulation with all components working and thrusters on. Note that `PUll` is constant until the helium tank pressure drops to the point where it can no longer provide enough flow to the regulator. At this point, the regulator model switches to `invisible`.

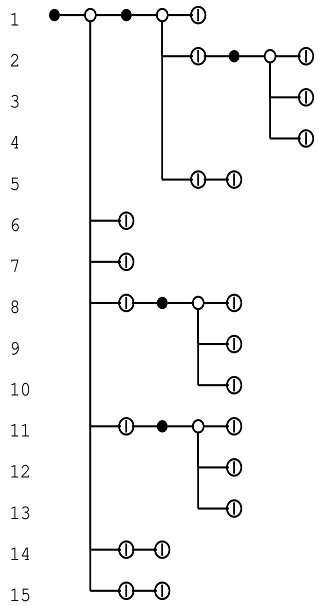
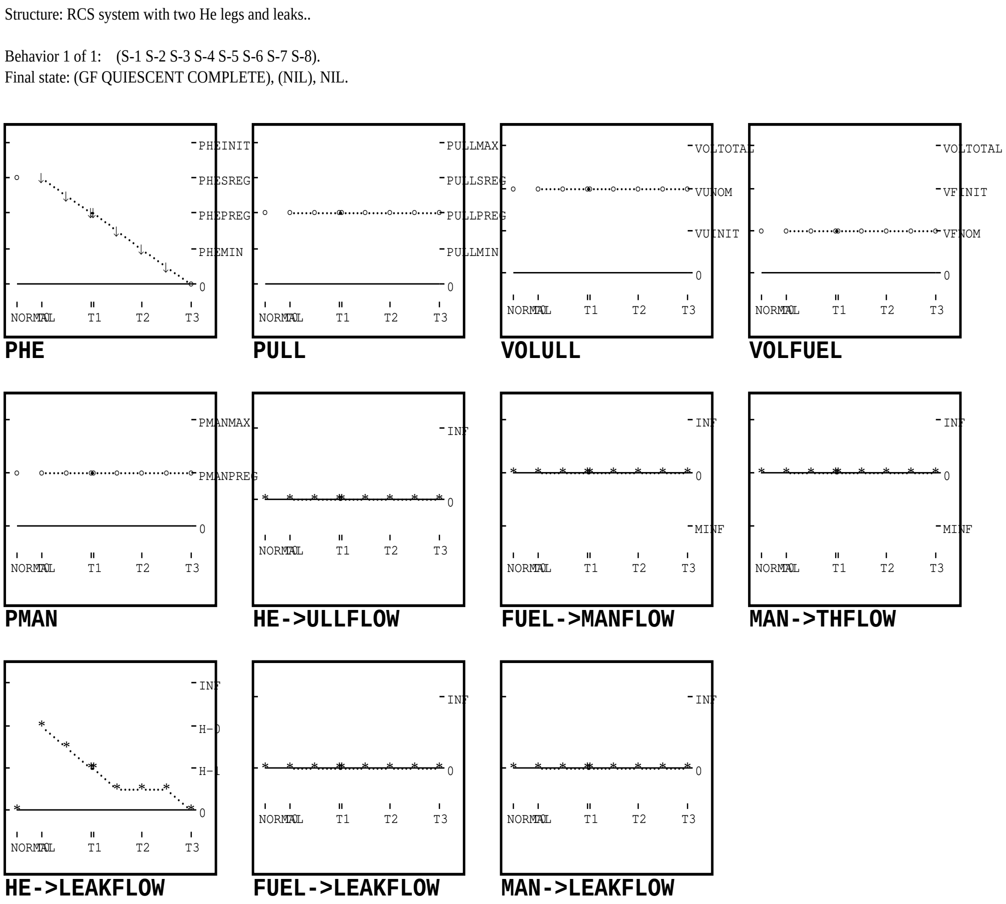


Figure 7: The behavior tree for Figure 6. The difference in behaviors is due to branching on the events `PHe` reaches `PUllPreg`, `VolFuel` reaches 0, and `PUll` reaches `PUllMin`.

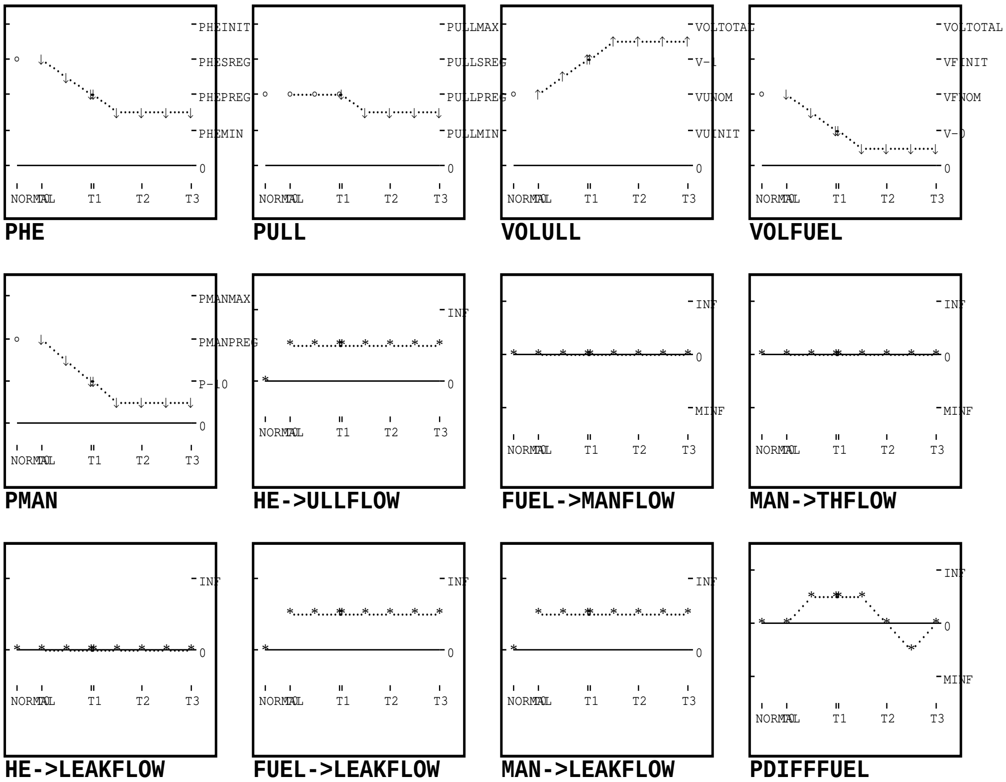



```
(run-rcs :init 'equilibrium
         :PHe 'PHeSreg
         :configuration '((leg1 closed working working)
                          (leg2 open   working working)
                          (HeTankLEak? yes) (HeLeakS max)
                          (tankvalve open) (manvalve open)
                          (thruster 0)))
```

Figure 8: Simulation of a helium tank leak with the thrusters off. Note that there is a region transition at `T1` that simulates the operation of the check valve to prevent backflow into the helium tank.

Structure: RCS system with two He legs and leaks..

Behavior 11 of 33: (S-1 S-2 S-6 S-11 S-18 S-22 S-38 S-47).
Final state: (GF COMPLETE), (NIL), T<INF.

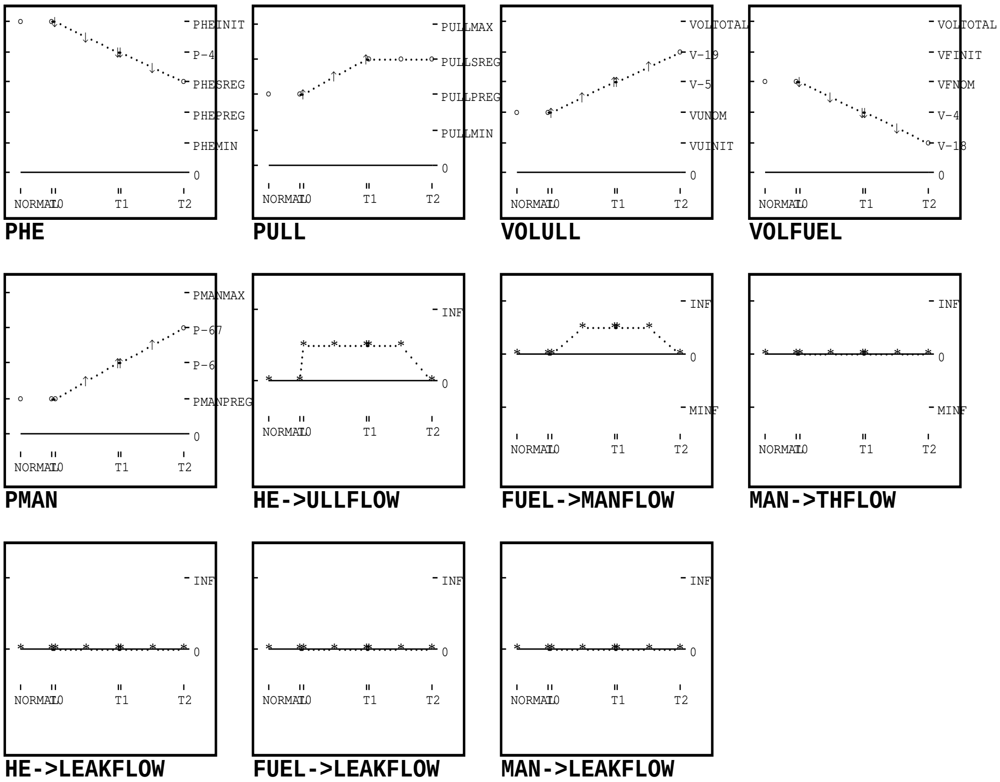



```
(run-rcs :init 'equilibrium
         :PHe 'PHeSreg
         :configuration '((leg1 closed working working)
                          (leg2 open   working working)
                          (TankLineLeak? yes) (LineLeakS max)
                          (tankvalve open) (manvalve closed)
                          (thruster 0)))
```

Figure 9: Simulation of a tank line leak with the thrusters off but `TankValve` open and `ManValve` closed. Note that there is chatter in `PDiffFuel`. If both valves were closed, this chatter would disappear and only one behavior would be generated.

Structure: RCS system with two He legs and leaks..

Behavior 16 of 41: (S-1 S-2 S-3 S-4 S-19 S-24 S-41).
Final state: (GF QUIESCENT COMPLETE), (NIL), NIL.

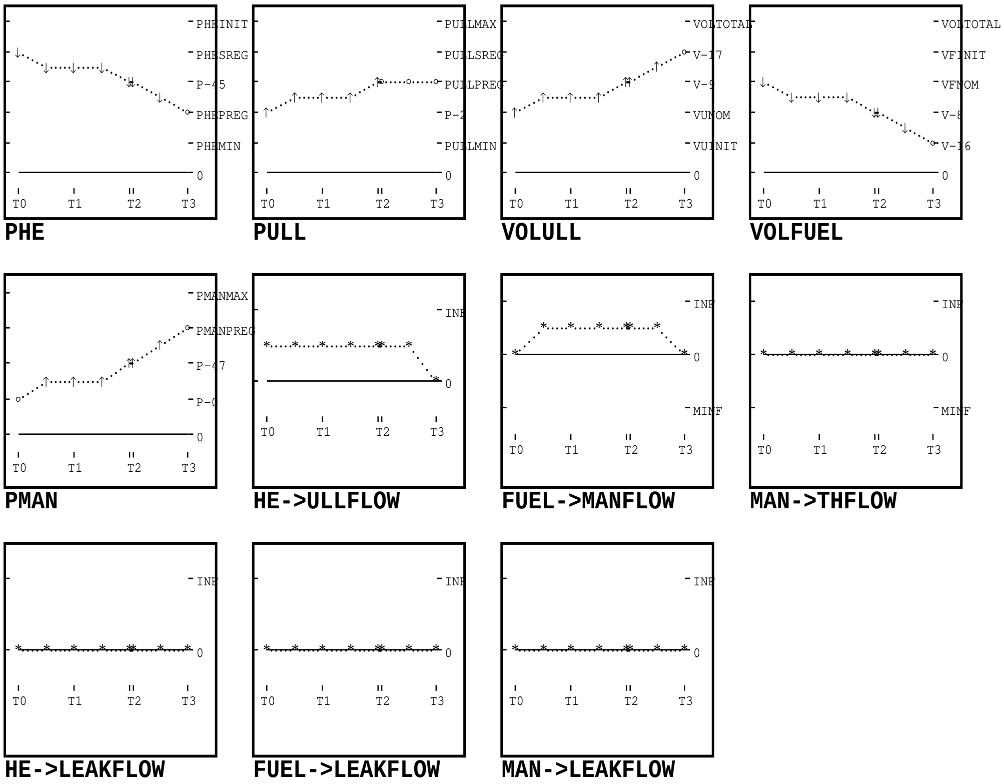



```
(run-rcs :init 'equilibrium
         :Phe 'PHeInit
         :configuration '((leg1 closed working working)
                          (leg2 open   failed-open working)
                          (tankvalve open) (manvalve open)
                          (thruster 0)))
```

Figure 10: Simulation of a primary regulator failure with the thrusters off. Initially, the regulator behaves like a pipe to bring PUll up to the secondary regulator pressure PUllSreg. At this point, the regulator switches back to the ideal-psource mode. There are 41 behaviors for this model, most of the distinctions being caused by uncertainty as to whether bringing PUll up will empty the propellant tank or the helium tank.

Structure: RCS system with two He legs and leaks..

Behavior 9 of 33: (S-0 S-1 S-3 S-9 S-12 S-29 S-33 S-45).
Final state: (GF QUIESCENT COMPLETE), (NIL), NIL.




```
(run-rcs :init 'low-PUll)
```

Figure 11: Simulation of opening the helium line valve when the ullage pressure is below the regulated pressure. As in Figure 10, the regulator behaves like a pipe to bring `PUll` up to pressure (in this case `PUllPreg`). At this point, the regulator switches back to the `ideal-psource` mode. There are 33 behaviors for this model, most of the distinctions being caused by uncertainty as to whether bringing `PUll` up will empty the propellant tank or the helium tank.

5 Follow-on Work

The current RCS model meets the requirements of tractable simulation and adequate fault coverage for a wide variety of operational scenarios. Future work on this model will center on providing better fault coverage without sacrificing tractability. Key enhancements are :

- Improving the component descriptions. The current model is somewhat brittle because the regulator model violates the no function in structure rule. By replacing this with a more realistic controller-based model, it should apply over a wider range of scenarios. Additionally, the helium tank model does not include the effects of temperature on pressure and pressure on volume (it turns out that the tank itself can expand). Finally, the models of valves and regulators do not allow for failures other than fully open or fully closed. These improvements will probably negatively impact tractability.

- Adding more of the RCS to the model. Certain components, such as the propellant tank relief valve and thruster chamber dynamics are unmodeled. Without them, certain types of faults cannot be simulated. The current model abstracts also away the multiple thrusters attached to each system and completely ignores the crossfeed systems. Since there are leaks and sensors that apply here, the model should distinguish these items to provide better coverage. Because the RCS model is large, it is difficult to add new components by direct modification of the QDE. However, since the components of the model and their connections are well-understood, a logical next step would be to reimplement the model using a component-connection ontology like CC [Franke and Dvorak, 1990].

- Generic tractability improvements. One way to improve tractability is to include quantitative information about the RCS and then use semiquantitative methods [Kuipers and Berleant, 1988, Kay and Kuipers, 1992] to refute impossible behaviors and tighten the predictions of the remaining ones. Another approach is to use behavior aggregation techniques to abstract irrelevant distinctions in the behaviors [Fouche and Kuipers, 1991].

There are also other similar systems in the space shuttle that are candidates for this type of modeling. In particular, the Orbital Maneuvering System is designed in much the same way as the RCS. Much of the model-building expertise that went into constructing the RCS model could therefore be applied to this system as well.

References

[de Kleer and Brown, 1984] Johan de Kleer and John Seely Brown. A qualitative physics based on confluences. *Artificial Intelligence*, 24:7–83, 1984.

[Farquhar *et al.*, 1991] Adam Farquhar, Benjamin Kuipers, Jeff Rickel, David Throop, and The Qualitative Reasoning Group. Qsim: The program and its use. Technical Report AI90–123, Artificial Intelligence Laboratory, University of Texas at Austin, Austin, Texas 78712, October 1991.

[Fouche and Kuipers, 1991] Pierre Fouche and Benjamin Kuipers. Abstracting irrelevant distinctions in qualitative simulation. In *Working Papers from the Fifth International Workshop on Qualitative Reasoning about Physical Systems*, pages 171–185, Austin, Texas, 1991.

[Franke and Dvorak, 1990] David Franke and Daniel Dvorak. CC: Component-connection models for qualitative simulation – a user's guide. Technical Report AI90–126, Artificial Intelligence Laboratory, University of Texas at Austin, Austin, Texas 78712, January 1990.

[Kay and Kuipers, 1992] Herbert Kay and Benjamin Kuipers. Numerical behavior envelopes for qualitative models. In *Working Papers from the Sixth International Workshop on Qualitative Reasoning*, pages 252–267, Edinburgh, Scotland, 1992.

[Kuipers and Berleant, 1988] Benjamin Kuipers and Daniel Berleant. Using incomplete quantitative knowledge in qualitative reasoning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 324–329, 1988.

A The Complete QSIM RCS Model

This section is a listing of the file rcs-model.lisp which is available from the author.

```
;;; -*- Syntax: Common-lisp; Package: qsim -*-

(in-package 'qsim)

; A model of the RCS from He tank through to Thruster
;
; The system is modeled as a four-variable system :
;   AmtHe   - the amount of helium in the Helium Tank
;   AmtFuel - the amount of fuel/oxidizer in the fuel/oxidizer tank
;   AmtUll  - the amount of Ullage in the fuel/oxidizer tank
;             (i.e., the amount of Helium in the fuel/oxidizer tank)
;   AmtMan  - the amount of fuel in the manifold
;
; The model covers this section of the RCS
;
```

```
;         -------------
;        |   He Tank   |
;        |             |
;         -------------
;           |      |
;           |      |
;        Valve1  Valve2
;           |      |
;         Sreg1  Sreg2
;           |      |
;         Preg1  Preg2
;           |      |
;           |      |
;            ------
;               |
;               |
;          CheckValve
;               |
;               |
;         -------------
;        |  Ullage     |
;        |~~~~~~~~~~~~~|
;        |  Fuel       |
;         -------------
;               |
;           TValve
;               |
;           MValve
;               |
;            -----
;           | Man-|
;           | fold|
;            -----
;             / \
;            /   \  <--- Thruster
;
;
;
; Assumptions :
;   - The amount of propellant vapor in the fuel/oxidizer tank is negligable
;     and is therefore not modeled.
;   - The pressure regulator internals are not modeled.  Instead, the function
;     of the regulator is modeled as a set of modes as follows :
;         o  ideal-psource : The regulator is maintaining the proper output
```

```
;                            pressure.  The flow thru the regulator is a
;                            function of the flow out of the load.  This is
;                            when PHe > <regpress> and PUll = <regpress>.
;         o  invisible     : This is the transition state from ideal-psource
;                            when PHe drops below <regpress>.  Here PUll follows
;                             PHe and flow is a function of flow out of the load
;                            (i.e. it is as if the regulator were not there).
;         o  closed        : This is the case where there is no path between the
;                            helium and fuel tanks either because of failure or
;                            because the isolation valves are closed.
;                            Here the pressures are independent of one another
;                            and there is no flow.
;         o  pipe          : Here PHe >= <regpress> and PUll < <regpress>.  The
;                            flow is a function of PDiffRegs.
;     In addition to the above regulator models, there is also a mode that
;     determines the value for Preg.  This value can be :
;         o  preg   : The primary regulator determines the pressure.
;         o  sreg   : The secondary regulator determines the pressure.
;         o  open   : There is a path between the regulator input and output.
;         o  closed : There is no path between the regulator input and output.
;     To further simplify things, the pair of legs is merged into a single
;     entity.  The rationale here is that there is no way to measure the
;     relative flows in each leg, so combining the flows does not affect
;     diagnosis.
;   - The amount of gas in the regulator segments is negligible.  This allows us
;     to model the isolation valve and both regulators in a leg as a single
;     resistive device.
;   - The system is assumed to be isothermal.
;   - Helium tank expansion under pressure is assumed to be negligable.

;;; Things that remain to be done :
;;; - Add heline leak.
;;; - Add Q2 info for flows (using Hellerstein method of curve fitting for M+).
;;; - Add a PMin lmark for PMan.  This is the pressure at which the jets starve.
;;; - Add a model for the thruster that reflects the change in chamber pressure.
;;; - Model the relief valve.
;;; - Model crossfeeds (assume other system is an infinite source of fuel?).

(define-QDE rcs-two-legs-leaks-noheline
   (text "RCS system with two He legs and leaks.")
   (quantity-spaces
      ;; Units for quantities :
      ;;  amounts   - lbs
      ;;  volumes   - in^3
```

```
;;  pressures - psia
;;  flows     - lbs/sec
;; He Tank
(AmtHe      (0 AmtHeMin AmtHePreg AmtHeSreg AmtHeInit)) ; Amt of Helium
                                                        ; in He Tank
(PHe        (0 PHeMin PHePreg PHeSreg PHeInit)) ; Pressure in He tank
(dAmtHe     (minf 0 inf))                       ; change in amt of He

;; He tank->regulator flows
(He->UllFlow (0 inf))                   ; flow from He tank to He line
                                        ; (sum of leg flows)

;; Regulators
(PDiffRegs  (minf 0 inf))               ; pressure diff across the regs

;; Ullage
(AmtUll      (0 AmtUllInit AmtUllNom AmtUllMax)) ; Amt ullage in prop Tank
(nrt[AmtUll] (0 nRTInit nRTNom nRTMax))          ; Amt in correct units
(PUll        (0 PUllMin PUllPreg  PUllSreg PUllMax))  ; Pressure in Fu tank
                                                      ; Min => jet failure

;; Fuel tank (other than Ullage)
(AmtFuel    (0 AmtFNom AmtFInit AmtFMax)) ; Amount of fuel in prop tank
(VolFuel    (0 VFNom VFInit   VolTotal))  ; Volume of fuel
(VolUll     (0 VUInit VUNom   VolTotal))  ; Volume of Ullage
(VolTotal   (VolTotal))                   ; Total vol of prop tank
(DenFuel    (0 DFPreg DFMax))      ; fuel density (a function of pressure)
(dAmtFuel   (minf 0 inf))          ; change in fuel in tank
(Fuel->ManFlow  (minf 0 inf))      ; flow proptank -> manifold
(PDiffFuel  (minf 0 inf))          ; PUll-PMan
(TIValve    (0 Max))               ; combined tank & man vlve conductance

;; Manifold and thruster
(AmtMan     (0 AmtPreg AmtMax))    ; Amount of fuel in manifold
(dAmtMan    (minf 0 inf))          ; change in manifold quantity
(PMan       (0 PManPreg PManMax))  ; Manifold pressure
(Man->ThFlow (minf 0 inf))         ; flow Manifold -> Thruster
(PDiffMan   (minf 0 PD* inf))      ; PMan-PVac

(DenFuelMan (0 inf))               ; Density of fuel in manifold
(DPFuel     (0 inf))               ; Density*PDiffMan
(f[DPFuel]  (0 inf))               ; CDa*f[DPFuel] = flow in lbs/sec
(Thruster   (0 Max))               ; Thruster opening
```

```
   (PVac       (0 inf))              ; Pressure of space

   ;; Leak variables

   ;; He tank leak
   (He->Leakflow (0 inf))
   (HeLeakS      (0 max))            ; leak conductance
   (PDiffHeAndVac (0 inf))           ; pressure diff between tank and vac
   (HeOutflow    (0 inf))            ; flow out of the He tank
   ; He Line leak can be included as below

   ;; Fuel Tank (&line leak)
   (Fuel->Leakflow (0 inf))          ; total leak flow (From tank and line)
   (FuelLeak       (0 inf))          ; tank leak flow
   (Fuel->LineLeak (0 inf))          ; leak flow from line
   (FuelLeakS      (0 max))          ; tank leak conductance
   (LineLeakS      (0 max))          ; line leak condctnce (used by manfld too)
   (PDiffFuelAndVac (0 inf))         ; pressure diff between tank and vac
   (FuelOutflow     (0 inf))         ; total flow out of fuel tank

   ;; Manifold (&line leak)
   (Man->Leakflow  (0 inf))          ; total leak flow (From man and line)
   (ManLeak        (0 inf))          ; manifold leak flow
   (Man->LineLeak  (0 inf))          ; leak flow from line
   (ManLeakS       (0 max))          ; manifold leak conductance
   (PDiffManAndVac (0 inf))          ; pressure diff between man and vac
   (ManOutflow     (minf 0 inf))     ; total flow out of manifold
   (MManOutflow    (minf 0 inf))     ; -ManOutflow
   )
(discrete-variables
   (reg-model (ideal-psource invisible closed pipe))
   (reg-mode  (open sreg preg closed))
   (TankValve (open closed))               ; Fuel Tank -> Tank Line Valve
   (ManValve  (open closed))               ; Tank Line -> Manifold Valve

   ;; Leak loacations
   (HeTankLeak?   (yes no))
   (FuelTankLeak? (yes no))
   (TankLineLeak? (yes no))
   (ManLeak?      (yes no))
   )
(constraints
   ;; Helium tank
   ((d/dt AmtHe dAmtHe))
```

```
((minus dAmtHe HeOutflow))
((M+ AmtHe PHe)   (0 0)  (AmtHeMin PHeMin) (AmtHePreg PHePreg)
                  (AmtHeSreg PHeSreg)
                  (AmtHeInit PHeInit))

;; Flow from HeTank to Ull
(mode (or (reg-model ideal-psource) (reg-model invisible))
  ((M+ FuelOutflow He->UllFlow) (0 0) (inf inf)))
(mode (reg-model pipe)
  ((M+ PDiffRegs He->UllFlow) (0 0) (inf inf)))
(mode (reg-model closed)
  ((zero-std He->UllFlow)))

;; He tank leak stuff
((add He->UllFlow He->LeakFlow HeOutFlow))
((mult HeLeakS PDiffHeAndVac He->LeakFlow))
((add PDiffHeAndVac PVac PHe))
(mode (HeTankLeak? yes)
  ((constant HeLeakS)))
(mode (HeTankLeak? no)
  ((zero-std HeLeakS)))

((add PDiffRegs PUll PHe) (0 PUllMin PHeMin) (0 PUllPreg PHePreg)
                          (0 PUllSreg PHeSreg))

;; PUll determined by most tightly closed regulator
(mode (and (reg-model ideal-psource)
           (or (reg-mode preg) (reg-mode sreg)))
  ((constant PUll)))
(mode (reg-model invisible)
  ((M+ PHe PUll) (0 0) (PHeMin PUllMin) (PHePreg PUllPreg)
                       (PHeSreg PUllSreg) (PHeInit PUllMax)))
(mode (reg-model invisible)
  ((zero-std PDiffRegs))) ; make sure PUll QDir is not set

;; Propellant tank - ullage
((d/dt AmtUll He->UllFlow))

;; Propellant tank - fuel flow
((d/dt AmtFuel dAmtFuel))
((minus FuelOutflow dAmtFuel))
((add PDiffFuel PMan PUll) (0 PManPreg PUllPreg))
((mult PDiffFuel TIValve Fuel->ManFlow))
(mode (and (TankValve open) (ManValve open))
```

```
  ((constant TIValve)))
(mode (or (TankValve closed) (ManValve closed))
  ((zero-std TIValve)))

;; Fuel Leaks
((add Fuel->Leakflow Fuel->Manflow FuelOutflow))
((add FuelLeak Fuel->LineLeak Fuel->Leakflow))
((add PDiffFuelAndVac PVac PUll))
((mult FuelLeakS PDiffFuelAndVac FuelLeak))
(mode (and (TankLineLeak? yes) (TankValve open))
  ((mult LineLeaks PDiffFuelAndVac Fuel->LineLeak)))
(mode (or (TankLineLeak? no) (TankValve closed))
  ((zero-std Fuel->LineLeak)))
(mode (FuelTankLeak? yes)
  ((constant FuelLeakS)))
(mode (FuelTankLeak? no)
  ((zero-std FuelLeakS)))

;; Fuel amount and volume
((M+ AmtUll nRT[AmtUll]) (0 0) (AmtUllInit nRTInit)
                         (AmtUllNom nRTNom) (AmtUllMax nRTMax))
((mult PUll VolUll nRT[AmtUll]) (0 0 0) (PUllPReg VUInit nRTInit)
                           (PUllPReg VUNom nRTNom)
                           (PUllMax VolTotal nRTMax))
((M+ PUll DenFuel) (0 0) (PUllPreg DFPreg) (PUllMax DFMax))
((mult DenFuel VolFuel AmtFuel) (DFPreg VFInit AmtFInit)
                                (DFPreg VFNom AmtFNom))
((add VolFuel VolUll VolTotal) (VolTotal 0 VolTotal)
                               (0 VolTotal VolTotal)
                               (VFNom VUNom VolTotal)
                               (VFInit VUInit VolTotal)
                               )
;; When there is no real flow (other than due to compression in the Man)
;; PUll should follow nRT[AmtUll]
(mode (and (reg-model pipe) (Fuel->Leakflow 0)
           (and (tivalve max)
                (and (thruster 0) (Man->Leakflow 0))))
  ((W+ PUll nRT[AmtUll])))

;; Manifold
((d/dt AmtMan dAmtMan))
((M+ AmtMan PMan) (0 0) (AmtPreg PManPreg) (AmtMax PManMax))
((add PDiffMan PVac PMan) (PD* 0 PManPreg))
((mult PDiffMan DenFuelMan DPFuel))
```

```
   ((M+ PMan DenFuelMan) (0 0) (PManMax inf))
   ((M+ DPFuel f[DPFuel]) (0 0) (inf inf))
   ((mult Thruster f[DPFuel] Man->ThFlow))
   ((add Fuel->ManFlow MManOutflow dAmtMan))
   ((W+ PDiffMan DPFuel))

   ;; Manifold leaks
   ((add Man->Leakflow Man->ThFlow ManOutflow))
   ((minus MManOutflow ManOutflow))
   ((add ManLeak Man->LineLeak Man->Leakflow))
   ((add PDiffManAndVac PVac PMan))
   ((mult ManLeakS PDiffManAndVac ManLeak))
   (mode (and (TankLineLeak? yes)) ;  (ManValve open) Manifold leaks one way
     ((mult LineLeaks PDiffManAndVac Man->LineLeak)))
   (mode (or (TankLineLeak? no))   ; (ManValve closed)
     ((zero-std Man->LineLeak)))
   (mode (ManLeak? yes)
     ((constant ManLeakS)))
   (mode (ManLeak? no)
     ((zero-std ManLeakS)))

   ;; Tank line leak conductance
   (mode (TankLineLeak? yes)
     ((constant LineLeakS)))
   (mode (TankLineLeak? no)
     ((zero-std LineLeakS)))

   ((constant VolTotal))
   ((constant PVac))
   ((constant Thruster))
   )
;; Simulation stops if He->Ullflow indicates flow into the he tank
;; or PUll reaches the jet failure level. [This should actually be
;; measured at the manifold since these two pressures could differ.]
(transitions ((He->UllFlow (0 dec)) -> t) ; reverse flow (switch to closed?)
             ((PUll        (PUllMin dec)) -> t) ; jet failure - stop sim
             ((and (PDiffRegs (0 dec)) (not (He->UllFlow (0 nil))))
              -> reg-set-low-pressure)
             ((and (PUll (PUllPreg nil)) (reg-model (ideal-psource std))
                   (reg-mode (sreg std)))
              -> sreg-pipe)
             ((and (PDiffRegs (0 dec)) (He->UllFlow (0 nil))
                   (not (reg-model (closed std))))
              -> check-valve-closed)
```

```
                ((and (PUll (PUllPreg inc)) (reg-model (pipe std))
                      (reg-mode (preg std)))
                 -> ideal-psource-preg)
                ((and (PUll (PUllSreg inc)) (reg-model (pipe std))
                      (reg-mode (sreg std)))
                 -> ideal-psource-sreg)
                )
   ;; Ignore qdirs on all flows
   (ignore-qdirs dAmtHe dAmtFuel PDiffFuel DPFuel f[DPFuel] Fuel->ManFlow
                   He->UllFlow Man->ThFlow PDiffMan dAmtMan
                  MManOutflow  ManOutflow FuelOutflow ; DenFuel
                  PDiffHeAndVac He->LeakFlow HeOutflow
                  Fuel->Leakflow Man->Leakflow Fuel->LineLeak Man->LineLeak
                  FuelLeak ManLeak
                 )
   ;; No landmarks for flows
   (no-new-landmarks dAmtHe He->UllFlow HeOutflow PdiffManAndVac PDiffFuelAndVac
                     PDiffFuel Fuel->ManFlow
                     dAmtFuel dAmtMan Man->ThFlow
                     FuelOutflow ManOutflow MManOutflow
                     Man->LeakFlow Fuel->LeakFlow Man->LineLeak
                     )

     ;; A nice layout for showing only the important variables.
     (layout (PHe PUll VolUll VolFuel nil )
             (PMan He->Ullflow Fuel->Manflow Man->Thflow nil)
             (He->Leakflow Fuel->Leakflow Man->Leakflow nil nil))
     ;; A layout that shows all the variables.
;    (layout (AmtHe PHe dAmtHe He->UllFlow PDiffRegs PVac)
;           (nil nil nil AmtUll nrt[AmtUll] PUll)
;           (AmtFuel VolFuel VolUll VolTotal DenFuel dAmtFuel)
;           (Fuel->ManFlow PDiffFuel TIValve AmtMan dAmtMan PMan)
;           (Man->ThFlow MManFlow PDiffMan DPFuel f[DPFuel] Thruster)
;           (PDiffHeAndVac He->LeakFlow Fuel->Leakflow Man->Leakflow HeLeakS HeOutflow))
   )


;;; Operating Region Transitions

;;; Transition region for low pressure regulator operation.
;;; The reg-model mode becomes invisible which makes the
;;; PDiffRegs = (0 std) and (M+ PHe PUll), i.e., PHe = PUll.
;;;
(defun reg-set-low-pressure (state)
```

```
  (format t "~%-> reg-set-low-pressure")
  (create-transition-state :from-state state
                           :to-qde rcs-two-legs-leaks-noheline
                           :assert '((reg-model (invisible std)))
                           :inherit-qmag :rest
                           :inherit-qdir '(PHe PMan tivalve pvac voltotal)
                           ))

;;; Transition region for reg-model ideal-psource -> pipe.
;;;
(defun sreg-pipe (state)
  (format t "~%-> sreg-pipe")
  (create-transition-state :from-state state
                           :to-qde rcs-two-legs-leaks-noheline
                           :assert '((reg-model (pipe std)))
                           :inherit-qmag #'(lambda (varname)
                                             (if (not (member varname
                                                              '(He->Ullflow
                                                                HeOutflow
                                                                dAmtHe
                                                                reg-model)))
                                                 t
                                               nil))
                           :inherit-qdir '(PMan tivalve pvac voltotal)
                           ))


;;; Transition region for check valve operation.
;;; The check valve will disconnect the HeTank from the Fuel Tank.
;;;
(defun check-valve-closed (state)
  (format t "~%-> check-valve-closed")
  (create-transition-state :from-state state
                           :to-qde rcs-two-legs-leaks-noheline
                           :assert '((reg-model (closed std)))
                           :inherit-qmag :rest
                           :inherit-qdir :rest
                           ))


;;; Switching from a pipe to an ideal psource with primary regulator working.
;;;
(defun ideal-psource-preg (state)
  (format t "~%-> ideal-psource-preg")
```

```lisp
  (create-transition-state :from-state state
                           :to-qde rcs-two-legs-leaks-noheline
                           :assert '((reg-model (ideal-psource std))
                                     (PUll (PUllPreg std)))
                           :inherit-qmag #'ideal-psource-qmag-inheritor
                           :inherit-qdir '(tivalve pvac voltotal)))

;;; Switching from a pipe to an ideal psource with primary regulator failed open
;;;
(defun ideal-psource-sreg (state)
  (format t "~%-> ideal-psource-sreg")
  (create-transition-state :from-state state
                           :to-qde rcs-two-legs-leaks-noheline
                           :assert '((reg-model (ideal-psource std))
                                     (PUll (PUllSreg std)))
                           :inherit-qmag #'ideal-psource-qmag-inheritor
                           :inherit-qdir '(tivalve pvac voltotal)))

;;; Lets us selectively inherit qmags
;;;
(defun ideal-psource-qmag-inheritor (varname)
  (if (member varname '(He->UllFlow HeOutflow dAmtHe))
      nil
      t))




;;; Simulation functions.

;;; Run an RCS simulation using the QDE rcs-two-legs-leaks-noheline.
;;; Inputs: r      - a list of (var qmag) pairs (with assumed std qdir) or
;;;                  entries of the form (leg1/2 valvepos pregcond sregcond)
;;;                  where valvepos is open or closed
;;;                  and   regcond  is working failed-open or failed-closed
;;;
(defun rcs-two-legs-leaks-noheline (r &key (PHe 'PHeSreg))
  (qsim-cleanup)
  (let* ((*dpy-model-mode* t)
         (*state-limit* 50)
         (norm (make-new-state :from-qde rcs-two-legs-leaks-noheline
                               :assert-values `((PHe   (,PHe std))
                                                (PUll  (PUllPReg std))
                                                (VolUll (VUNom std))
```

```lisp
                                                       (AmtFuel (AmtFNom nil))
                                                       (thruster (0 std))
                                                       (PVac     (0 std))
                                                       (VolTotal (VolTotal std))

                                                       (PMan     (PManPReg std))
                                                       (reg-model (ideal-psource std))
                                                       (reg-mode  (preg std))
                                                       (tankValve (open std))
                                                       (manValve  (open std))
                                                       (TIValve   (max std))

                                                       (HeTankLeak? (no std))
                                                       (FuelTankLeak? (no std))
                                                       (TankLineLeak? (no std))
                                                       (ManLeak? (no std))
                                                       )))
       init)
    (declare (special *dpy-model-mode*))

    (setq init (make-new-state :from-state norm
                               :inherit '(PHe PUll AmtFuel PVac VolTotal PMan)
                               :assert-values (determine-model-mode r)))

    (qsim init)
    (qsim-display init :reference-states `((normal ,norm)))))


;;; This function preprocesses the valve and failure settings so that the proper
;;; model modes are set.  The real work is computing reg-model and reg-mode from
;;; the individual leg statuses.
;;; Inputs: r      - a list of (var qmag) pairs (with assumed std qdir) or
;;;                  entries of the form (leg1/2 valvepos pregcond sregcond)
;;;                  where valvepos is open or closed
;;;                  and   regcond  is working failed-open or failed-closed
;;; Returns: a list of assert-values for make-new-state.
;;;
(defun determine-model-mode (r)
  (let ((model-values nil)
        (reg-values nil))
    (declare (special *dpy-model-mode*))
    (multiple-value-setq (model-values reg-values)
      (process-model-args r))
    (setf reg-values (process-reg-values reg-values))
```

```
    (setf model-values (apply-default-args model-values))
    (setf model-values (add-inferred-args model-values))
    (setf model-values (mapcar #'(lambda (arg) `(,(first arg) (,(second arg) std)))
                               (append reg-values model-values)))
    (when *dpy-model-mode* (format t "~%Mode: ~a" model-values))
    model-values))

;;; Process the argument.
;;; This returns #1: a list of (model-var qmag) and #2: the leg entries
;;;
(defun process-model-args (args)
    (let ((model-values nil)
          (reg-values nil))
      (dolist (arg args)
        (cond
          ((and (member (first arg) '(leg1 leg2)) (= (length arg) 4))
           (push arg reg-values))
          (t
           (push arg model-values))))
      (values model-values reg-values)))

(defun process-reg-values (reg-vals)
  (let ((default '(open working working))
        (leg1     (assoc 'leg1 reg-vals))
        (leg2     (assoc 'leg2 reg-vals)))
    (when (null leg1) (setf leg1 (cons 'leg1 defaults)))
    (when (null leg2) (setf leg2 (cons 'leg2 defaults)))

    (cond
      ((and (leg-closed-p leg1) (leg-closed-p leg2))
       `((reg-model closed) (reg-mode closed)))
      ((or (leg-open-p leg1) (leg-open-p leg2))
       `((reg-model pipe) (reg-mode open)))
      ((or (leg-sreg-p leg1) (leg-sreg-p leg2))
       `((reg-model ideal-psource) (reg-mode sreg)))
      ((or (leg-preg-p leg1) (leg-preg-p leg2))
       `((reg-model ideal-psource) (reg-mode preg)))
      (t
       (error "Don't know this set of legs (~a ~a)" leg1 leg2)))))

(defmacro leg-name (leg)  `(first ,leg))
(defmacro leg-valve (leg) `(second ,leg))
(defmacro leg-preg (leg)  `(third ,leg))
(defmacro leg-sreg (leg)  `(fourth ,leg))
```

```lisp
(defun leg-closed-p (leg)
  (or (eq (leg-valve leg) 'closed)
      (eq (leg-preg leg) 'failed-closed)
      (eq (leg-sreg leg) 'failed-closed)))

(defun leg-open-p (leg)
  (and (eq (leg-valve leg) 'open)
       (eq (leg-preg leg) 'failed-open)
       (eq (leg-sreg leg) 'failed-open)))

(defun leg-sreg-p (leg)
  (and (eq (leg-valve leg) 'open)
       (eq (leg-sreg leg) 'working)
       (eq (leg-preg leg) 'failed-open)))

(defun leg-preg-p (leg)
  (and (eq (leg-valve leg) 'open)
       (not (eq (leg-sreg leg) 'failed-closed))
       (eq (leg-preg leg) 'working)))



(defun apply-default-args (args)
  (let ((defaults '((thruster 0)
;                    (PUll PUllPreg)
                     (TankValve open)
                     (ManValve open)
                     (HeTankLeak? no)
                     (FuelTankLeak? no)
                     (TankLineLeak? no)
                     (ManLeak? no)

        )))
    (mapc #'(lambda (def-arg)
              (when (not (assoc (car def-arg) args))
                (push def-arg args)))
          defaults)
    args))

(defun add-inferred-args (args)
  (when (and (eq (lookup 'tankvalve args) 'open)
             (eq (lookup 'manvalve args) 'open)
             (not (lookup 'tivalve args)))
```

```
    (push `(tivalve max) args))
  (dolist (arg '((HeTankLeak? HeLeakS)
                 (FuelTankLeak? FuelLeakS)
                 (TankLineLeak? LineLeakS)
                 (ManLeak? ManLeakS)))
    (when (and (eq (lookup (car arg) args) 'yes)
               (not (lookup (cadr arg) args)))
      (push `(,(cadr arg) max) args))
    (when (and (eq (lookup (car arg) args) 'no)
               (not (lookup (cadr arg) args)))
      (push `(,(cadr arg) 0) args))
    )
  args)


;;; This function provides a test for "pipe" regulator mode.
;;; Since the system does not start in equilibrium, we can't use the other
;;; simulation function.
;;; The state is set up so that there is a high pressure in the He Tank and
;;; a pressure lower than PUllPreg in the Fuel Tank.
;;;
(defun rcs-two-legs-leaks-noheline-lopull ()
  (qsim-cleanup)
  (let* ((*dpy-model-mode* t)
         (*state-limit* 60)
         (init (make-new-state :from-qde rcs-two-legs-leaks-noheline
                               :assert-values '((PHe   (PHeSreg nil))
                                                (PUll  ((PUllMin PUllPReg) nil))
                                                (AmtUll ((AmtUllInit AmtUllNom) nil))
                                                (VolUll (VUNom nil))
                                                (VolFuel (VFNom nil))
                                                (thruster (0 std))
                                                (PVac     (0 std))
                                                (VolTotal (VolTotal std))

                                                (PMan     ((0 PManPReg) nil))
                                                (PDiffFuel (0 nil))
                                                (reg-model (pipe std))
                                                (reg-mode  (preg std))
                                                (TIValve   (max std))
                                                (tankValve (open std))
                                                (manValve  (open std))
```

```
                                                (HeTankLeak? (no std))
                                                (FuelTankLeak? (no std))
                                                (TankLineLeak? (no std))
                                                (ManLeak? (no std))

                                                )))
       )
    (declare (special *dpy-model-mode*))
    (qsim init)
    (qsim-display init)))



;;; Top-level simulation call and test cases.

;;; This is the top-level call to the rcs simulator.
;;; Inputs:  init           - The initial starting point.  This is either
;;;                           equilibrium or low-PU11.
;;;          configuration  - A list of (var qmag) pairs (with assumed std qdir)
;;;                           or entries of the form
;;;                           (leg1/2 valvepos pregcond sregcond) where valvepos
;;;                           is open or closed and regcond is working,
;;;                           failed-open, or failed-closed.  Note that this
;;;                           argument is used only for qde
;;;                           rcs-two-legs-leaks-noheline.
;;;                           For the (var qmag) entries, valid values are
;;;                           VAR             QMAGS (default/alt1/alt2/...)
;;;                           ---------       ------------------------------
;;;                           Thruster        0/max
;;;                           PU11            PU11Preg/other PU11 landmark name
;;;                           TankValve       open/closed
;;;                           ManValve        open/closed
;;;                           HeTankLeak?     no/yes
;;;                           FuelTankLeak?   no/yes
;;;                           TankLineLeak?   no/yes
;;;                           ManLeak?        no/yes
;;;          phe            - The initial pressure landmark for the PHe variable.
;;;                           This argument is used only for qde
;;;                           rcs-two-legs-leaks-noheline.
;;; Returns: nothing, but runs the appropriate QDE with the specified
;;;          configuration.
;;;
(defun run-rcs (&key (init 'equilibrium)
                     (configuration nil)
```

```
                   (PHe 'PHeSreg))
  (case init
    (equilibrium
     (rcs-two-legs-leaks-noheline configuration :Phe Phe))
    (low-PUll
     (rcs-two-legs-leaks-noheline-lopull))
    (T
     (format t "~&No such qde as ~a" qde))))

#|
;;; Test cases.  To run a test case, evaluate the corresponding call.

;;; Test case for an unchanging system.
;;; 1 behavior (quiescent point)
(run-rcs :init 'equilibrium
         :PHe 'PHeSreg
         :configuration '((leg1 closed working working)
                          (leg2 open   working working)
                          (tankvalve open) (manvalve open)
                          (thruster 0)))


;;; Test case for an unchanging system with thrust on.
;;; 15 behaviors
(run-rcs :init 'equilibrium
         :PHe 'PHeSreg
         :configuration '((leg1 closed working working)
                          (leg2 open   working working)
                          (tankvalve open) (manvalve open)
                          (thruster max)))


;;; Test case for a system with thruster off and a hetank leak.
;;; 1 behavior
(run-rcs :init 'equilibrium
         :PHe 'PHeSreg
         :configuration '((leg1 closed working working)
                          (leg2 open   working working)
                          (HeTankLeak? yes) (HeLeakS max)
                          (tankvalve open) (manvalve open)
                          (thruster 0)))

;;; Test case for a system with thruster on and a hetank leak.
;;; This model does not adequately model the behavior of the regulator
```

```
;;; after the helium pressure drops below the regulator pressure.
;;; 15 behaviors
(run-rcs :init 'equilibrium
         :PHe 'PHeSreg
         :configuration '((leg1 closed working working)
                          (leg2 open   working working)
                          (HeTankLeak? yes) (HeLeakS max)
                          (tankvalve open) (manvalve open)
                          (thruster max)))

;;; Test for a tank line leak with thrusters off and manvalve closed.
;;; 33+ behaviors with a state limit of 50.
(run-rcs :init 'equilibrium
         :Phe 'PHeSreg
         :configuration '((leg1 closed working working)
                          (leg2 open   working working)
                          (TankLineLeak? yes) (LineLeakS max)
                          (tankvalve open) (manvalve closed)
                          (thruster 0)))

;;; Test for a tank line leak with thrusters off tankvalve and manvalve closed.
;;; By closing the valves, there is no branching on PDiffFuel.
;;; 1 behavior.
(run-rcs :init 'equilibrium
         :Phe 'PHeSreg
         :configuration '((leg1 closed working working)
                          (leg2 open   working working)
                          (TankLineLeak? yes) (LineLeakS max)
                          (tankvalve closed) (manvalve closed)
                          (thruster 0)))

;;; Primary regulator fails open.
;;; 41 behaviors.
;;;
(run-rcs :init 'equilibrium
         :Phe 'PHeInit
         :configuration '((leg1 closed working working)
                          (leg2 open   failed-open working)
                          (tankvalve open) (manvalve open)
                          (thruster 0)))


;;; Test for a tank line leak with thrusters off and manvalve open.
;;; 39+ behaviors with a state limit of 50
```

```
(run-rcs :init 'equilibrium
         :Phe 'PHeSreg
         :configuration '((leg1 closed working working)
                          (leg2 open   working working)
                          (TankLineLeak? yes) (LineLeakS max)
                          (tankvalve open) (manvalve open)
                          (thruster 0)))

;;; Test case for leg closed and lo pressure in ullage, then leg open.
;;; Thruster is off.
;;; 33 behaviors
(run-rcs :init 'low-PUll)

|#
```