Modeling the Space Shuttle Liquid Hydrogen Subsystem

Bemina Atanacio

April 2000

Model-Based Verification Project
Dependable Systems Upgrade Initiative

Technical Note
CMU/SEI-2000-TN-002

This material is based upon work supported under a National Science Foundation Graduate Fellowship. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation.

Contents

List of Figures

List of Tables

Abstract

This paper describes experiences with modeling the liquid hydrogen subsystem of the space shuttle. The Symbolic Model Verifier tool and the Software Cost Reduction tool set were used to model and specify the behavior of the system. The tools were then used to check for errors in the models. Modeling a problem from several different perspectives offers the chance to uncover discrepancies among different models and to understand the problem space enough to ask important questions about the behavior of the system.

Each tool presented different issues in modeling the problem. Both models and a breakdown of the time spent during this study are included as appendices.

1 Introduction

The liquid hydrogen (LH2) subsystem is a series of pipes and valves that controls the flow of liquid hydrogen into and out of the external tank of the space shuttle. The behavior of the system from start to lift-off was modeled as a finite state machine by an intern at NASA. This state machine model was translated into SMV (Symbolic Model Verifier) and SCR (Software Cost Reduction) notations.

There are two types of specifications: operational and property based. Operational specifications represent the system as a state machine. These types of specifications are less likely to omit required behavior [Heitmeyer 98b]. The modeling language used in the SMV tool is an example of an operational model. Property-based specifications express system properties as logical formulas. These specifications are concise and abstract, thereby minimizing implementation bias [Heitmeyer 98b]. The specification language used in the SCR tool set is an example of a property-based specification language.

The SMV tool is an automated model checker developed at Carnegie Mellon University. It uses ordered binary decision diagrams to explore and check the state space of models. The SCR language is based on the tabular function notations of Parnas and is a requirements-specification language; its automated tool set checks for consistency and completeness.

In this report, the experiences of modeling the LH2 subsystem with both tools are discussed. The issues that were encountered during these experiences are also described. The author's background in modeling includes a four month survey of several formal mathematical models and two months of learning SMV notation and its checker. The author was new to the SCR tool set at the beginning of this project, however she had been introduced briefly to function tables.

2 Liquid Hydrogen Subsystem

The liquid hydrogen (LH2) subsystem is responsible for filling the space shuttle’s external tank with liquid hydrogen [Wight 92]. It is also responsible for maintaining the level of liquid hydrogen in the tank prior to launch. The subsystem is made up of the launchpad liquid hydrogen storage tank, the space shuttle’s external tank, and transfer lines and valves.

The launchpad LH2 storage tank holds the liquid hydrogen until it is time to fill the external tank. The transfer lines carry the LH2 from the storage tank to the external tank. Valves control the flow of LH2 in the various transfer lines. The transfer lines are also equipped with gauges that monitor temperature and pressure.

A diagram of the LH2 subsystem state machine is shown in Figure 1 below.

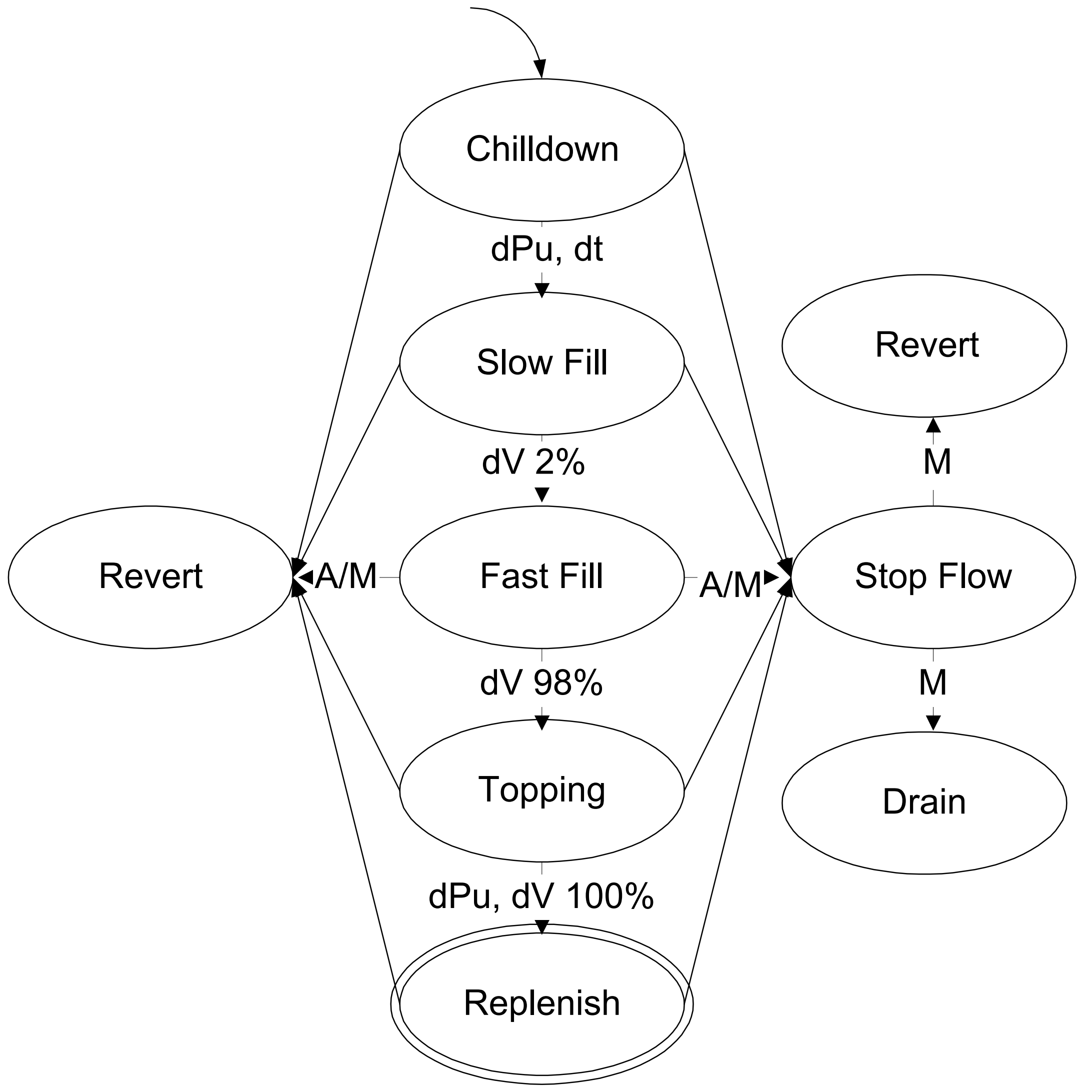

Figure 1: LH2 State Machine Diagram

Several issues during the modeling were encountered. One issue that arose was the author's lack of domain knowledge which prohibited her from interpreting the charts, diagrams, and plots that were included in the NASA intern's report. Therefore, there was a lot of potentially useful information in the report that could not be extracted and used in the models.

There was also a lack of information pertaining to the states of Revert, Stop Flow, and Drain and their transitions. They were depicted in the state machine diagram, but their semantics were not included in the NASA intern's report. This resulted in modeling the system without these states and transitions and reduced the amount of nondeterministic and potentially interesting behavior of the models.

3 The SMV Tool

The Symbolic Model Verifier (SMV) is a tool for checking finite state systems against computational tree logic (CTL) claims, which are called “specs” [McMillan 92]. SMV uses a symbolic model-checking algorithm, which is based on ordered binary decision diagrams (OBDDs), to check the validity of the CTL specs. The model of the system is written in a Pascal-like specification language. Clarke, et al., McMillan, and Srinivasan have written papers that explain the theory behind the SMV tool and how to use the tool [Clarke 94, McMillan 92, Srinivasan 98].

A few interesting points were discovered during the modeling of this system in the SMV tool. Although the model is specified as a “program” and can be thought of as one, it does not behave exactly as a program would. The SMV tool behaves as if the entire OBDD of the model were computed and then verifies the specification. This allows you to “see the future.” For example, you may assign a value to variable x, based on the *future* value of variable y. Figure 2 shows a small example of this in the SMV language. This ability of the tool was counter-intuitive until it became clear that the checking of the model is not exactly analogous to the running of a program; it is actually performed by manipulating equations. In the end, this feature of the SMV tool was not used in the model of the LH2 subsystem.

```
x: {one, two};
y: {four, five};
ASSIGN
init(x) := one;
init(y) := four;
next(x) :=
   case
      (next(y) = five): two;
      1: x;
   esac;
next(y) :=
   case
      y = four : five;
   esac;
```

Figure 2: Using the Next Value of y to Determine the Value of x

Time and volume were modeled as their own state machines or *modules* so that they could have nondeterministic behavior; their state machines could stay in any one state for a finite length of time and then move on to the next state. However, when a spec to check this behavior was created, the SMV tool always generated a counterexample and the state machine would stay in one state infinitely. To resolve this problem, fairness conditions for each state had to be added. The FAIRNESS keyword in the SMV tool signifies that the condition following it must hold infinitely often. In other words, in an infinite trace of the state machine's operation, there must never be a point at which the condition becomes false and stays false forever. The FAIRNESS keyword prevents the checker from "stalling" in one state forever.

Designating time as an integer variable with a large range (from 0 to 405 seconds) would cause a state explosion in the model. To abstract the information in time, the author chose to model it as having a finite number of enumerated states. Because only three times were important to the behavior of the system (60 s, 165 s, 405 s), the states "initial," "short," "medium," and "long" were created. Volume was modeled similarly with four abstract states: 0%, 2%, 98%, and 100%, and had a similar nondeterministic behavior and fairness problem. The volume module of the model is shown in Figure 3.

```
MODULE volume(system-state)
VAR
   state : {zero, two, ninety-eight, one-hundred};

ASSIGN
   init(state) := zero;

   next(state) :=
   case
   --Either stay in the same state or change states
   --spontaneously based on state of system
      (system-state = slow-fill) & (state = zero):
{zero, two};
      (system-state = fast-fill) & (state = two):
{two, ninety-eight};
      (system-state = topping-state) &
      (state = ninety-eight):
{ninety-eight, one-hundred};
      1 : state;
   esac;

FAIRNESS
   !(state = zero)
FAIRNESS
   !(state = two)
FAIRNESS
   !(state = ninety-eight)
```

Figure 3: Nondeterminism and Fairness in the Volume Module

The entire SMV model of the LH2 subsystem can be found in Appendix A.

4 The SCR and Spin Tools

The Software Cost Reduction (SCR) notation is based on the tabular function notations of Parnas [Heitmeyer 96]. It is not an operational modeling language; rather it is a requirements specification language. The behavior of the system is thought of as a state machine or a *mode class*. Each state is a *mode*, and transitions from mode to mode are through *events*. The SCR tool set checks specifications for consistency. A consistent specification has proper syntax, no type errors, no nondeterminism, and no missing cases. It must have all initial values either defined or able to be derived from the tables, and all modes must be reachable from the initial mode. Also, it should not contain circular definitions for variables [Heitmeyer 96]. In addition, the specification should be *disjoint* and have *full coverage* [Easterbrook 96]. Being disjoint means that no combination of conditions has conflicting actions specified for it. Full coverage means that an action is specified for each combination of failure conditions [Easterbrook 96].

The SCR method uses the notion of monitored and controlled variables. These categories correspond to input and output variables. All variables are required to be classified as either monitored or controlled. This explicit classification forced the author to look carefully at the system and determine how it should be modeled and at what level of abstraction. For example, one of the variables represented the volume of liquid hydrogen in the external tank. This volume can be considered monitored because its value determines which system valves to open and close. This volume can also be considered controlled because the system controls the volume of hydrogen in the tank. Volume was chosen to be a monitored variable because the level of abstraction was at the level of individual valves. The system was modeled as directly controlling the valves, and not the volume.

The SCR specifications in the SCR tool set documentation are written in deeply nested Backus-Naur form (BNF) notations and are confusing to read. To resolve this issue, the author had to look at example specifications written by other members of the Software Engineering Institute (SEI). Some of the BNFs were also rewritten by the author to remove the nesting. For example, Figure 4 shows the steps the author took to create one event in the event table for the cTopping variable, following the Syntax of Expressions table in the SCR tool set documentation [Kirby 97]. (Note: in the figure below, $\rightarrow^{n}$ signifies a transformation from one BNF to another in *n* steps.)

expression $\rightarrow^{1}$ o_event $\rightarrow^{1}$ a_event $\rightarrow^{1}$ simple_event $\rightarrow^{1}$ cond_event $\rightarrow^{1}$

event WHEN full_ifandonlyif $\rightarrow^{2}$

AT_TRUE LPAREN full_ifandonlyif close_expression WHEN
    full_implication ifandonlyif $\rightarrow^{4}$

@T(full_ifandonlyif) WHEN full_implication ifandonlyif $\rightarrow^{9}$

@T(true) WHEN full_implication ifandonlyif $\rightarrow^{1}$

@T(true) WHEN o_cond implication ifandonlyif $\rightarrow^{3}$

@T(true) WHEN LPAREN full_ifandonlyif close_expression ifandonlyif $\rightarrow^{8}$

@T(true) WHEN (relat_exp close_expression ifandonlyif $\rightarrow^{8}$

@T(true) WHEN (cChilldown = open)

Figure 4: Transformation of "expression" to "@T(true) WHEN (cChilldown = open)"

As you can see, the creation of one conditional event is very involved, especially for someone who does not know the syntax of the tool.

Nondeterminism is excluded purposely in SCR specifications because many systems that have nondeterministic behavior are underspecified. Because the SCR tool set checks for incomplete specifications, it prohibits nondeterministic behavior. The external tank vent valve in the LH2 subsystem is "unstable" in certain states of the system. This was easily modeled as nondeterministic behavior in the SMV tool. However, in the SCR tool set, this specification had to be left out and replaced with "stable" valve behavior (meaning the valve was left in the position it was in prior to the unstable states).

The SCR tool set provides a visual way to look for variable dependencies. The dependency graph screenshot shown in Figure 5 shows the connections between dependent variables. It also shows unspecified variables and variables with circular dependencies. The SCR tool set also provides textual warnings, but the graph is an easy way to scan for these possible errors.

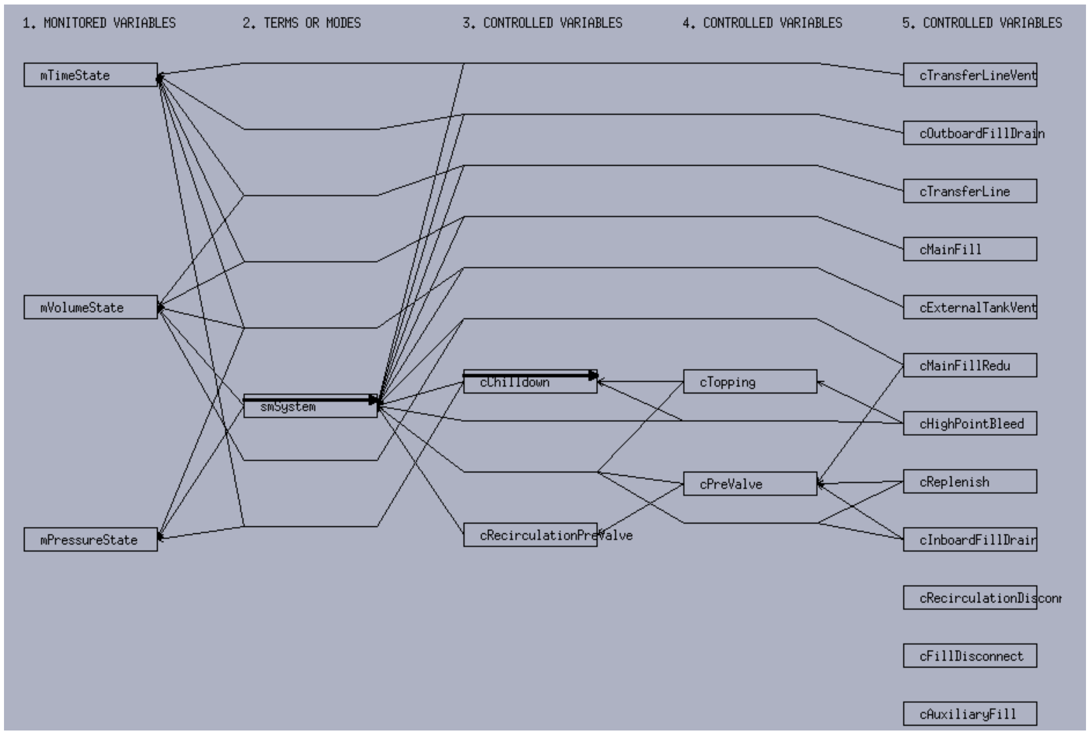

Figure 5: Screenshot of Dependency Graph for the LH2 Specification

The controlled variables have dependency arrows leading backwards through other controlled variables and the terms to the monitored variables. Arrows from left to right are darker because they may be part of a circular dependency.

The SCR tool set facilitates the verification of the specification against claims by using Xspin, which is the graphical version of the modeling tool Spin, developed at Bell Laboratories. The Spin tool models in a language called PROMELA (PROcess MEta LAnguage). Like in the SMV tool, claims are written to describe specific behaviors of the system and verified automatically by the tool. Unlike in the SMV tool, claims are written in linear temporal logic (LTL) rather than CTL. Many expressions written in CTL can be translated into LTL and vice versa. The Spin tool translates the PROMELA model into a C-code prototype and runs it, thereby verifying the specs on the fly [Spin].

The author did not have to learn PROMELA, because the SCR tool set automatically translates tables into PROMELA. Developing claims was simple, since similar specs had already been developed in the SMV tool, and CTL can be converted to LTL easily. There were some troubles running the tool and verifying the specs. The SCR tool set allows variables to be of the type “real,” but the Spin tool does not. All of the “real” variables had to be changed to integers. This change did not solve all of the problems in running the Spin tool. Because the “integer” type variables had large ranges, the resulting state space was very large. The computer used did not have enough memory to explore the entire model. To solve this problem, the integer variables were replaced with enumerated ones. This is one of the abstraction methods discussed by Heitmeyer, et al. [Heitmeyer 98a]. Only then was the state space small enough for the Spin tool to check thoroughly. As with the SMV tool, all claims were valid.

The SCR specification of the LH2 subsystem can be found in Appendix B.

5 Conclusions

The goal of this study was to document the experiences of modeling a system in different languages and using different tools. There are several benefits to modeling in multiple languages. On one hand, multiple models help in developing a greater understanding of the problem space, eliciting questions about the model to be posed to domain experts, and catching unexpected system failures and behaviors. On the other hand, there is a time and effort cost in learning new tools and languages, and in keeping all the models up-to-date and consistent with each other and with the system implementation.

By having more than one model, you gain a greater understanding of the problem space and the behavior of the system. Easterbrook, et al., mention that it is not the end product of modeling that is most important; the experience of modeling is [Easterbrook 96]. Heitmeyer, et al., mention that by examining the problem from both operational and property-based viewpoints, discrepancies in both specifications can be detected [Heitmeyer 98a]. "Carefully designed redundancy in specifications can be useful" [Heitmeyer 98a]. However, time must be spent keeping all of the models synchronized with each other and the system implementation. If all of the models overlap in a large portion of the problem, making one change in the system implementation can require updating all of the models. Carefully designed redundancy may offer benefits; however, complete redundancy is wasteful and unnecessary.

Building models elicits questions to be posed to domain experts. As the author modeled in the SMV tool, there was not enough information about certain states and their transitions, such as the "stop flow" state. This gap in the model helped to formulate important questions to be asked of domain experts. The use of monitored and controlled variables in the SCR tool set forced the author to critically examine the level of abstraction in the model.

It takes time and effort to learn a new tool and to learn the syntax of a new modeling language. See Appendix C for a breakdown of the author's time. A lot of effort (31%) was spent in mentor meetings and group presentations. Seven percent of the effort was spent modeling in the SMV tool. This low figure is attributable to the author's familiarity with that tool. Sixteen percent of the effort was spent learning and modeling in the SCR tool set because of the author's lack of familiarity with the tool and the language syntax.

Modeling tools are much more efficient than theorem-proving techniques at detecting errors and providing counterexamples to claims because these tools search the state space automatically and completely. When used early in the development cycle (before code has been written and sometimes before requirements have been completed) modeling can catch unexpected system behavior or failures before their cost of repair increases as the cycle continues [Heitmeyer 96].

Formal model checking is often compared to the more traditional method of verification, theorem proving. Model checking is cheaper in time and effort than theorem proving [Easterbrook 96]. Model checking is automated, can completely search a state space, and does not require extensive training in logic and higher mathematics. The author's background includes high-school-level predicate logic, college-freshman-level first-order logic, and two months in CTL. The counterexamples generated by these automated tools are also useful artifacts because they can be used as "bug reports" and because the root causes of problems can be found and corrected easily. For example, in Appendix A, the SMV model uses specifications that prove true. They could be negated easily to produce a "witness" to the correct system behavior [Clarke 94]. In the example shown in Figure 6, the spec "`AF(LH2.state = topping-state -> AF(vol-gauge.state = one-hundred))`" is negated to "`EG (!(LH2.state = topping-state -> AF (vol-gauge.state = one-hundred)))`." The output is also shown.

```
-- specification EG (!(LH2.state = topping-state -> AF
vo... is false
-- as demonstrated by the following execution sequence
state 2.1:
LH2.state = chilldown-state
LH2.chilldown = closed
LH2.transfer-line = closed
LH2.transfer-line-vent = open
LH2.main-fill = closed
LH2.outboard-fill-drain = closed
LH2.external-tank-vent = closed
LH2.main-fill-redu = closed
LH2.auxiliary-fill = closed
LH2.inboard-fill-drain = open
LH2.topping = closed
LH2.high-point-bleed = closed
LH2.fill-disconnect = open
LH2.recirculation-disconnect = open
LH2.pre-valve = open
LH2.recirculation = closed
LH2.replenish = closed
vol-gauge.state = zero
timer.state = initial
press-gauge.state = initial
```

Figure 6: Output of a Negated Property to Create a Witness

Although there are drawbacks to using multiple models, they are outweighed by the benefits: automated verification and insight into the problem space. In modeling the LH2 subsystem, no major problems were found. The experiences gained with the modeling tools can increase developers' software engineering knowledge and mature their engineering discipline.

References

[Clarke 94]	Clarke, E.; Grumberg, O.; McMillan, K.; & Zhao, X. *Efficient Generation of Counterexamples and Witnesses in Symbolic Model Checking* (CMU-CS-94-204). Pittsburgh, PA: Carnegie Mellon University, 1994.
[Easterbrook 96]	Easterbrook, S.; & Callahan, J. *Formal Methods for V&V of Partial Specifications: An Experience Report* (CSRP 443). Brighton: University of Sussex at Brighton, 1996.
[Heitmeyer 96]	Heitmeyer, C. L.; Jeffords, R. D.; & Labaw, B. G. "Automated Consistency Checking of Requirements Specifications." *ACM Transactions on Software Engineering and Methodology* 5, 3 (July 1996): 231-261.
[Heitmeyer 98a]	Heitmeyer, C.; Kirby Jr., J.; Labaw, B.; Archer, M.; & Bharadwaj, R. "Using Abstraction and Model Checking to Detect Safety Violations in Requirements Specifications." *IEEE Transactions on Software Engineering* 24, 11 (November 1998): 927-947.
[Heitmeyer 98b]	Heitmeyer, C.; & Jeffords, R. Automatic Generation of State Invariants from Requirements Specifications. *Proceedings of the Sixth International Symposium on Foundations of Software Engineering*, Orlando, FL, November 3-5, 1998.
[Kirby 97]	Kirby Jr., J. "SCR* Tool set: The User Guide." Draft, 1997.
[McMillan 92]	McMillan, K. L. "The SMV System." Available WWW: <URL: http://www.cs.cmu.edu/~modelcheck /smv.html>.
[Spin]	"Spin—Formal Verification Web page, Bell Laboratories." <URL: http://netlib.bell-labs.com/netlib/spin/whatispin.html>

[Srinivasan 98]	Srinivasan, G. R. & Gluch, D. P. *A Study of Practice Issues in Model-Based Verification Using the Symbolic Model Verifier (SMV)* (CMU/SEI-98-TR-013, ADA358751). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1998. Available WWW: <URL: http://www.sei/publications/documents/98.reports /98tr013/98tr013abstract.html>.
[Wight 92]	Wight, L. *Finite-State Analysis Shuttle LH2 Subsystem*. National Aeronautics and Space Administration, July 1992.

Appendix A: SMV Model and Specifications

```
MODULE system (volume, pressure, time)

VAR
 state: {chilldown-state, slow-fill, fast-fill, topping-state,
replenish-state, revert-state, stop-flow, drain};
 chilldown: {open, closed};
 transfer-line: {open, closed};
 transfer-line-vent: {open, closed};
 main-fill: {open, closed};
 outboard-fill-drain: {open, closed};
 external-tank-vent: {open, closed};
 main-fill-redu: {open, closed};
 auxiliary-fill: {open, closed};
 inboard-fill-drain: {open, closed};
 topping: {open, closed};
 high-point-bleed: {open, closed};
 fill-disconnect: {open, closed};
 recirculation-disconnect: {open, closed};
 pre-valve: {open, closed};
 recirculation: {open, closed};
 replenish: {open, closed};

ASSIGN
 init(state) := chilldown-state;
 init(chilldown) := closed;
 init(transfer-line) := closed;
 init(transfer-line-vent) := open;
 init(main-fill) := closed;
 init(outboard-fill-drain) := closed;
 init(external-tank-vent) := closed;
 init(main-fill-redu) := closed;
 init(auxiliary-fill) := closed;
 init(inboard-fill-drain) := open;
 init(topping) := closed;
 init(high-point-bleed) := closed;
 init(fill-disconnect) := open;
 init(recirculation-disconnect) := open;
 init(pre-valve) := open;
 init(recirculation) := closed;
 init(replenish) := closed;

 next(state) :=
 case
   (state = chilldown-state) & (pressure = forty-three) :
slow-fill;
   (state = slow-fill) & (volume = two) : fast-fill;
   (state = fast-fill) & (volume = ninety-eight) :
topping-state;
   (state = topping-state) & (volume = one-hundred) :
replenish-state;
   1: state;
 esac;

 next(chilldown) :=
 case
   (time = medium) &
(state = chilldown-state): open;
   (state = chilldown-state) & (pressure = forty-three):
closed;
   (state = slow-fill) & (chilldown = closed): open;
   1 : chilldown;
 esac;

 next(transfer-line) :=
 case
   (time = medium & state = chilldown-state) |
(volume = two & state = slow-fill): open;
   (time = long & state = chilldown-state) |
(volume = ninety-eight & state = fast-fill): closed;
   1 : transfer-line;
 esac;

 next(transfer-line-vent) :=
 case
   (time = short & state = chilldown-state): closed;
   1 : transfer-line-vent;
 esac;

 next(main-fill) :=
 case
```

```
  (time = medium) & (state = chilldown-state): open;
  (volume = one-hundred) & (state = topping-state): closed;
  1 : main-fill;
esac;

next(outboard-fill-drain) :=
case
  (time = short) & (state = chilldown-state): open;
  1 : outboard-fill-drain;
esac;

next(external-tank-vent) :=
case
  (time = short & state = chilldown-state) |
(volume = ninety-eight & state = fast-fill): open;
  (pressure = forty-three) & (state = chilldown): closed;
  --valve position is unstable in the following statements
  --is substate 2 of slow-fill dependent on the chilldown
  --and topping valves?
  (state = slow-fill) & (chilldown = open) &
(topping = open): {open, closed};
  (state = fast-fill): {open, closed};
  --end of nondeterminism
  1: external-tank-vent;
esac;

next(main-fill-redu) :=
case
  (pre-valve = closed) & (state = fast-fill): open;
  (volume = one-hundred) & (state = topping-state) : closed;
  1: main-fill-redu;
esac;

next(auxiliary-fill) :=
case
  --This valve has no transitions listed
  1: auxiliary-fill;
esac;

next(inboard-fill-drain) :=
case
  (pre-valve = closed) & (state = fast-fill) : closed;
  1: inboard-fill-drain;
```

```
 esac;

 next(topping) :=
 case
   (chilldown = open) & (state = slow-fill): open;
   1: topping;
 esac;

 next(high-point-bleed) :=
 case
   (state = slow-fill) & (chilldown = open) &
(topping = open) : open;
   1: high-point-bleed;
 esac;

 next(fill-disconnect) :=
 case
   --This valve has no transitions listed
   1: fill-disconnect;
 esac;

 next(recirculation-disconnect) :=
 case
   --This valve has no transitions listed
   1: recirculation-disconnect;
 esac;

 next(pre-valve) :=
 case
   (state = fast-fill) & (recirculation = open): closed;
   1: pre-valve;
 esac;

 next(recirculation) :=
 case
   (state = fast-fill): open;
   1: recirculation;
 esac;

 next(replenish) :=
 case
   (state = fast-fill) & (pre-valve = closed): open;
   1: replenish;
 esac;

MODULE volume(system-state)
VAR
   state : {zero, two, ninety-eight, one-hundred};

ASSIGN
   init(state) := zero;

   next(state) :=
   case
   --Either stay in the same state or change states
   --spontaneously based on state of system
      (system-state = slow-fill) & (state = zero): {zero, two};
      (system-state = fast-fill) & (state = two): {two, ninety-
eight};
      (system-state = topping-state) & (state = ninety-eight):
{ninety-eight, one-hundred};
      1 : state;
   esac;

FAIRNESS
   !(state = zero)
FAIRNESS
   !(state = two)
FAIRNESS
   !(state = ninety-eight)

MODULE time(system-state, trans-line-vent, chilldown-valve,
trans-line)
VAR
   state : {initial, short, medium, long};

ASSIGN
   init(state) := initial;

   next(state) :=
   case
      --Either stay in the same state or change states
      --spontaneously based on state of certain valves
      (trans-line-vent = open) & (state = initial):
{initial, short};
      (chilldown-valve = closed) & (state = short):
```

```
{short, medium};
      (trans-line = open) & (state = medium): {medium, long};
      1 : state;
   esac;

FAIRNESS
   !(state = initial)
FAIRNESS
   !(state = short)
FAIRNESS
   !(state = medium)

MODULE pressure(system-state, chilldown-valve)
VAR
   state : {initial, forty-three};

ASSIGN
   init(state) := initial;

   next(state) :=
   case
      --Either stay in the same state or change states
      --spontaneously based on state of certain valves and
      --the system state
      (chilldown-valve = closed) &
(system-state = chilldown-state): {initial, forty-three};
      --Not quite true. Pressure may go up or down in later
      --states, but may not be important to model.
      1 : state;
   esac;


MODULE main
VAR
   LH2: system(vol-gauge.state, press-gauge.state,
timer.state);
   vol-gauge: volume(LH2.state);
   timer: time(LH2.state, LH2.transfer-line-vent,
LH2.chilldown, LH2.transfer-line);
   press-gauge: pressure (LH2.state, LH2.chilldown);

--Sanity specs here
SPEC
```

```
   --Make sure pressure changes state
   AF(press-gauge.state = initial &
AF(press-gauge.state = forty-three))

SPEC
   --Make sure time progresses
   AF(timer.state = initial & AF(timer.state = medium &
AF(timer.state = long)))

SPEC
   --Make sure volume progresses
   AF(vol-gauge.state = zero & AF(vol-gauge.state = two &
      AF(vol-gauge.state = ninety-eight &
AF(vol-gauge.state = one-hundred))))

--Important specs here

SPEC
   --ET is filled to the 100% level during topping
   AF(LH2.state = topping-state ->
AF(vol-gauge.state = one-hundred))

SPEC
   --Level of the LH2 is maintained at 100% during Replenish
   AF(LH2.state = replenish-state &
AF(vol-gauge.state = one-hundred))
```

Appendix B: SCR Model

Name	Base Type	Units	Legal Values
yPressure	Float	Psia	[0.0, 43.7]
yTime	Integer	second	[0, 405]
yValve	Enumerated	N/A	open, closed
yVolume	Integer	%	[0, 100]

Table 1: Type Dictionary

Name	Modes	Initial Mode	Table?	Comment
smSystem	sChilldown, sSlowFill, sFastFill, sTopping, sReplenish	sChilldown	Yes	models the LH2 system

Table 2: Mode Class Dictionary

Name	Type	Initial Value	Accuracy	Comment
mPressureState	yPressure	0.0	N/A	Tried modeling this as another mode class, but had trouble with crossing modes. Does this start at 0?
mTimeState	yTime	0	N/A	Tried previously as a mode class
mVolumeState	yVolume	0	N/A	Tried previously as a mode class

Table 3: Monitored Variable Dictionary

Name	Type	Initial Value	Accuracy
cAuxiliaryFill	yValve	closed	N/A
cChilldown	yValve	closed	N/A
cExternalTankVent	yValve	closed	N/A
cFillDisconnect	yValve	open	N/A
cHighPointBleed	yValve	closed	N/A
cInboardFillDrain	yValve	open	N/A
cMainFill	yValve	closed	N/A
cMainFillRedu	yValve	closed	N/A
cOutboardFillDrain	yValve	closed	N/A
cPreValve	yValve	open	N/A
cRecirculationDisconnect	yValve	open	N/A
cRecirculationPreValve	yValve	closed	N/A
cReplenish	yValve	closed	N/A
cTopping	yValve	closed	N/A
cTransferLine	yValve	closed	N/A
cTransferLineVent	yValve	open	N/A

Table 4: Controlled Variable Dictionary

Source Mode	Events	Destination Mode
sChilldown	@T(mPressureState = 43.7)	sSlowFill
sSlowFill	@T(mVolumeState = 2)	sFastFill
sFastFill	@T(mVolumeState = 98)	sTopping
sTopping	@T(mVolumeState = 100)	sReplenish

Table 5: Mode Transition Table for smSystem

Modes	Events
sChilldown	@T(mTimeState = 60)
	closed

Table 6: Event Table for cTransferLineVent

Modes	Events	
sChilldown	@T(mTimeState = 165)	@T(mTimeState = 405)
sSlowFill	@T(mVolumeState = 2)	NEVER
sFastFill	NEVER	@T(mVolumeState = 98)
	open	closed

Table 7: Event Table for cTransferLine

Modes	Events
sSlowFill	@T(true) WHEN (cChilldown = open)
	open

Table 8: Event Table for cTopping

Modes	Events
sFastFill	@T(true) WHEN (cPreValve = closed)
	open

Table 9: Event Table for cReplenish

Modes	Events
sFastFill	@T(INMODE)
	open

Table 10: Event Table for cRecirculationPreValve

Modes	Events
sFastFill	@T(true) WHEN (cRecirculationPreValve = open)
	closed

Table 11: Event Table for cPreValve

Modes	Events
sChilldown	@T(mTimeState = 60)
	open

Table 12: Event Table for cOutboardFillDrain

Modes	Events	
sFastFill	@T(true) WHEN (cPreValve = closed)	NEVER
sTopping	NEVER	@T(mVolumeState = 100)
	open	closed

Table 13: Event Table for cMainFillRedu

Modes	Events	
sChilldown	@T(mTimeState = 165)	NEVER
sTopping	NEVER	@T(mVolumeState = 100)
	open	closed

Table 14: Event Table for cMainFill

Modes	Events
sFastFill	@T(true) WHEN (cPreValve = closed)
	Closed

Table 15: Event Table for cInboardFillDrain

Modes	Events
sSlowFill	@T(true) WHEN (cChilldown = open AND cTopping = open)
	open

Table 16: Event Table for cHighPointBleed

Modes	Events	
sChilldown	@T(mTimeState = 60)	@T(mPressureState = 43.7)
sFastFill	@T(mVolumeState = 98)	NEVER
	open	closed

Table 17: Event Table for cExternalTankVent

Modes	Events	
sChilldown	@T(mTimeState = 165)	@T(mPressureState = 43.7)
sSlowFill	@T(true) WHEN (cChilldown = closed)	NEVER
	open	closed

Table 18: Event Table for cChilldown

Appendix C: Time Summary

Task	**Time (min)**	**Time (hrs)**	**% Total**
Reading NASA intern's report	43	0.7	1.9
Searching and reading literature	283	4.7	11.9
Planning	118	2	5.0
Setting up the computer (and other overhead)	90	1.5	3.8
Creating statecharts	73	1.2	3.1
Creating SMV model	133	2.2	5.6
Creating SCR model	189	3.2	7.8
Learning SCR	95	1.6	4.0
Writing report	338	5.6	14.3
Attending meetings and presentations	662	11	28
Preparing and giving final presentation	315	5.3	13.3
Using spin	30	0.5	1.3
Total	**2369**	**39.5**	**100**

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY (LEAVE BLANK)		2. REPORT DATE April 2000	3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE Modeling the Space Shuttle Liquid Hydrogen Subsystem			5. FUNDING NUMBERS C — F19628-95-C-0003
6. AUTHOR(S) Bemina Atanacio			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213			8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2000-TN-002
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/DIB 5 Eglin Street Hanscom AFB, MA 01731-2116			10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES			
12.A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS			12.B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

This paper describes experiences with modeling the liquid hydrogen subsystem of the space shuttle. The Symbolic Model Verifier tool and the Software Cost Reduction tool set were used to model and specify the behavior of the system. The tools were then used to check for errors in the models. Modeling a problem from several different perspectives offers the chance to uncover discrepancies among different models and to understand the problem space enough to ask important questions about the behavior of the system.

Each tool presented different issues in modeling the problem. Both models and a breakdown of the time spent during this study are included as appendices.

14. SUBJECT TERMS space shuttle, National Aeronautics and Space Administration, NASA, symbolic model verifier, software cost reduction, liquid hydrogen subsystem			15. NUMBER OF PAGES <41 pp.> 16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102