Finite-State Analysis of Space Shuttle Contingency Guidance Requirements

Judith Crow
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

SRI Technical Report SRI-CSL-95-17
Also Available as NASA Contractor Report 4741

July 15, 1996

Abstract

We describe a finite-state analysis of the mode-sequencing requirements for the Contingency Three-Engines-Out Guidance function of the Space Shuttle flight control system.

Contents

Chapter 1

Introduction

The project documented in this report was undertaken in the context of a research program in formal methods [BCC+95] and forms part of a study intended to demonstrate that formal specification and verification can enhance the clarity, precision, and comprehension of requirements specifications for space applications. The project has focused on the mode-sequencing requirements for *Contingency Three-Engines-Out* (3E/O), a function of the contingency guidance component of the Space Shuttle flight-control system. The analysis and accompanying documentation are presented here in two chapters: the first containing an informal description of the 3E/O requirements and the second containing a discussion of the finite-state analysis of these requirements. A final chapter summarizes our work and considers directions for future research.

1.1 Motivation

Although the quality of Space Shuttle flight software is generally regarded as exemplary among NASA software development projects, much of the quality assurance activity in early lifecycle phases remains a manual exercise lacking well-defined methods or techniques [NASA93, p. 22]. Shuttle flight software is complex and life-critical. Software upgrades to accommodate new missions such as the recent MIR docking, new capabilities such as Global Positioning System navigation, and improved algorithms such as the newly automated three-engine-out contingency abort maneuvers (3E/O) are continually introduced. Such upgrades underscore a need recognized in the NASA community and in a recent assessment of Shuttle flight software development, for "state-of-the-art technology" and "leading-edge methodologies" to meet the demands of software development for increasingly large and complex systems [NAS93, p. 91]. The 3E/O project described in this report represents an attempt to explore productive tools and pragmatic strategies to address this need.

1.2 Acknowledgments

The informal description of 3E/O, and the finite-state analysis that follows, are based on our interpretation of the requirements documented in [Roc94]. We have derived our understanding of these requirements from this document, a short note on 3E/O region selection by Bill Kaufman (Loral), and from several discussions with Ron Avery (Loral), who offered valuable insight into the occasionally inscrutable requirements document and provided detailed comments on an earlier draft of this report. David Hamilton (formerly with Loral) initially identified 3E/O as a potentially interesting application and provided access to documentation and 3E/O expertise during early phases of the work. The guidance provided by John Rushby (SRI) and by our technical monitor, Rick Butler (NASA Langley Research Center), was also extremely valuable. Naturally, we take the blame for any misconceptions that remain. We hope that by giving a precise specification of our understanding, those with greater familiarity and understanding of the true requirements will be able to identify our errors and help us develop a precise, understandable, and accurate analysis that will inform future applications of formal methods to space applications, as well as future use and modification of [Roc94].

Chapter 2

Informal Description of Three-Engines-Out

The informal description of 3E/O covers three topics: general background providing an overview of the Space Shuttle's physical structure and contingency abort procedures, 3E/O functionality including a more detailed description of the two main 3E/O Contingency Guidance functions, and a final section on the process currently used to analyze Space Shuttle flight software requirements.

We begin with an informal description of the requirements document itself. Shuttle flight software requirements are documented as Functional Subsystem Software Requirements (FSSRs)— low-level software requirements specifications written in English prose with strong implementation biases, typically including material in the form of pseudo-code, tables, digrams, or flowcharts. The 3E/O requirements specification is written largely in English prose and pseudo-code, accompanied by tables and by flowcharts diagrammed in a notation unlike any used elsewhere in the modern computer science literature. For example, the control flow proceeds both forward and backward in these diagrams. In the worst case, it may be necessary to trace the control flow forward through multiple pages and then backward through these same pages to define a single path from entry to exit. Although the prose, tables, and flowcharts are generally consistent, all three have undergone five modifications – documented in-line – resulting in layers of annotation that can be difficult to unravel.

2.1 Background

The Space Shuttle mission profile distinguishes six main *flight phases*: ascent, orbit insertion, orbit, deorbit, entry, and abort, each of which is further divided into two to four *flight modes* and controlled by software systems referred to as "Digital AutoPilots" (DAPs). The Ascent and Transition DAPs provide guidance, navigation, and control

(GN&C) for the Space Shuttle during powered flight. Put simply, the navigation function determines where the Shuttle is, guidance determines where it should go next, and the control function determines how to get it there. Predictably, specific GN&C functions vary with respect to mission phase; for example, while guidance target parameters related to launch site are crucial during the ascent phase, they are irrelevant during the on-orbit phase. The Space Shuttle is propelled into orbit by two solid-fuel rocket boosters (SRBs) and three rocket engines (Space Shuttle Main Engines or SSMEs) that are fueled by an external tank (ET), with additional thrust for orbital insertion provided by the Orbital Maneuvering System (OMS) and the Reaction Control System (RCS). The SRBs burn out and are jettisoned approximately two minutes into flight. The SSMEs are shut off and the external tank jettisoned prior to stable-orbit insertion; the exact ascent profile varies depending on tradeoffs including where the ET is likely to land. For example, direct insertion using only the SSMEs means that the ET is carried to a higher apogee, increasing the possibility of an inopportune landing, whereas Main Engine Cutoff (MECO) followed by an OMS burn lessens the danger that the ET will land in a populated area. In any case, powered flight refers to the SSME-powered ascent and transition phases in a nominal ascent and to the SSME-powered ascent phase associated with a mission abort and Return to Launch Site (RTLS).[1]

RTLS is one of four types of intact ascent aborts and is used if one or more of the SSMEs fail during the first 4 minutes and 20 seconds of flight. As the name suggests, the Shuttle reverses flight direction and returns to the launch site. The other three abort options are: transoceanic abort landing (TAL), abort to orbit (ATO), and abort once around (AOA). TAL is an emergency landing at a European or North African Airport, chosen on the basis of launch azimuth and ascent profile. TAL is an option when a velocity greater than 4,000 mph has been reached and is generally preferred to an RTLS, if both options are available. The conditions under which an RTLS versus a TAL abort are required can be diagrammed as shown in Figure 2.1. The minimum and maximum velocities referred to in the figure are "I-loads," i.e., software constants that are (re)calculated for each mission. The maximum altitude referred to is calculated each guidance cycle as a function of apogee altitude, velocity, and I-load values. The arrows indicate values above(below) the thresholds defined by the given conditions.

AOA, which is more desirable than TAL, is used when one or two SSMEs fail after the SRBs burn out, but too early in the ascent for an ATO, which becomes an option if SSME failure occurs late in the ascent. The orbit achieved during ATO is typically lower than the nominal orbit, requires less performance, and provides time to evaluate the situation and decide on an alternative mission plan.

The four types of intact ascent aborts (summarized above) all use the Shuttle's OMS and RCS systems and, as the name suggests, assume that there is a reasonable chance for an intact abort. The 3E/O case is different because if all three SSMEs fail, the

[1] The terminology is potentially confusing. The RTLS abort clearly involves descent rather than ascent; the (mission) phase name reflects the fact that an RTLS abort is initiated under control of the Ascent DAP.

normal ascent

$\Uparrow$

maximum velocity for RTLS abort

$\Downarrow$

RTLS abort (too much energy for TAL abort)

$\Uparrow$

maximum altitude tolerable without exceeding 3.5g acceleration

$\Downarrow$

TAL abort

$\Uparrow$

minimum velocity for TAL abort

$\Downarrow$

RTLS abort

Figure 2.1: 3E/O Conditions for RTLS and TAL Aborts

chance of an intact abort is virtually nil. Nevertheless, a 3E/O abort invokes RTLS for two reasons. First, the Glide RTLS DAP is the only DAP designed to handle no-engine maneuvering of the combined orbiter and ET in the atmosphere; second, the RTLS DAP contains support software that provides the safest possible conditions for crew bailout.[2]

The Shuttle can move in six axes: three translational axes: `X`, `Y`, `Z`, and three rotational axes: *roll*, *pitch*, and *yaw*. Movement is achieved using two powerful OMS engines, 38 primary RCS jets (also called thrusters), and six vernier RCS jets. The OMS engines are used for major orbital maneuvers, and to provide control in the translational axes, with additional, finer control provided by the primary RCS jets. Smaller maneuvers use only the RCS jets. Translational maneuvers use the primary RCS jets and rotations normally use the vernier RCS jets. During contingency abort maneuvers, as well as normal ascent, RCS jets are used to safely maneuver the Shuttle away from the ET after separation. Since the Shuttle is considerably lower in the atmosphere, more RCS jets are needed for an RTLS ET separation than for a nominal or TAL ET separation. An RTLS abort typically requires dumping OMS and RCS fuel to improve maneuverability and reduce landing weight, which is accomplished by firing the OMS and RCS engines.

2.2 Contingency Three-Engines-Out

Contingency Three-Engines-Out is a low-level Ascent DAP and RTLS guidance function responsible for monitoring ascent parameters and, if three SSMEs fail sequentially or

[2]We are grateful to Ron Avery for clarifying the use of RTLS in the 3E/O case.

simultaneously, calculating and commanding the appropriate ET separation and entry maneuvers if an RTLS abort is necessary. In certain situations, 3E/O is also responsible for auto contingency maneuvers resulting from the failure of two SSMEs (2E/O). For example, 3E/O takes over when a main engine cut-off is commanded during a 2E/O contingency maneuver. 3E/O is executed repeatedly at specified intervals that range from 1.92 seconds between SRB separation and MECO confirmed, to 0.16 seconds after MECO confirmed or during a pre-SRB ("first-stage") ET separation. Each execution of 3E/O is part of a *guidance cycle* that remains active during powered flight until either an RTLS contingency abort is required or progress along the powered flight trajectory is sufficient to allow a normal ET separation even if three SSMEs fail.

The contingency 3E/O function consists of two main subfunctions: a region-select function that selects a contingency-maneuver mode based on the values of ascent parameters, and a contingency guidance function that is used strictly for display if the ascent is normal, but is responsible for calculating and commanding initial RTLS abort maneuvers determined by the selected region if an RTLS contingency arises. Note that the maneuvers calculated and commanded by the two 3E/O functions may differ from one guidance cycle to the next in response to changes in the external environment or in the Shuttle's internal state.

2.2.1 3E/O Region Selection

The ascent parameters monitored by 3E/O include dynamic pressure, altitude, altitude rate, velocity, range, and angle of attack. The value of these parameters, the flight phase, and a small number of commands passed down from higher-level control functions as flags (e.g., `high_rate_sep`) together determine the Shuttle's current contingency maneuver region. There are six possible regions, each associated with a particular type of abort maneuver. The maneuvers differ largely with respect to timing, angle of attack (α) and sideslip (β) necessary for safe separation of the external tank. Each of the six regions is associated with a unique color used to display the mode to the crew on the Shuttle's trajectory display. The color reflects either normal ascent (blank) or the relative severity of the contingency, although normal ascent is typically not displayed until roughly six minutes after SRB separation, i.e., approximately a minute before MECO. The six maneuver regions are usually denoted by an index, i, where $0 \leq i \leq 4$ or $i = 102$.[3] Table 2.1 summarizes each of the six regions, including its associated color and index, the conditions under which Main Engine Cut-Off (MECO) occurs, and the corresponding ET separation maneuver. The table is based on notes by Bill Kaufman (Loral), and anticipates discussion (in Section 2.2.2) of the 3E/O Contingency Guidance function. Only five of the six regions discussed here can actually be assigned during

[3]Region index values illustrate one of several apparent notational inconsistencies in the requirements document. The index value 102 reflects the mode corresponding to a 3E/O contingency prior to SRB separation, i.e., during flight mode 102. None of the other region index values correspond to flight modes.

3E/O region selection; region 4 is assigned by 3E/O Contingency Guidance after trying and failing to execute a region 3 maneuver.

color	*index*	*MECO conditions*	*ET separation maneuver*
blank	0	Normal	No maneuver necessary.
blue	102	SRBs still attached.	Separate during SRB tailoff.
yellow	3	Dynamic pressure high, and not expected to drop.	Maneuver to $\alpha = -4$ deg, $\beta = 0$ and establish acceptable pitch, yaw, roll rates.
red	4	Region 3 maneuver started after MECO, but dynamic pressure too high for RCS jets to perform desired pitch-down maneuver.	Separate immediately if $\alpha < 0$ & pitch, yaw, roll rates acceptable. Pitch up to $\alpha \approx 125$ deg, establish small pitch, yaw, roll rates with pitch rate negative, then separate.
orange	2	Fairly high angle of attack and moderate dynamic pressure, or high-rate separation commanded by higher-level function.	Maneuver to $\beta = 0$. When β, yaw, and roll rates small, establish -4 deg/sec pitch rate, then separate.
green	1	No high-rate separation command and not yet past apogee, or either α or dynamic pressure is fairly low.	Attitude-independent separation; separate at earliest safe time after null pitch, yaw, and roll rates achieved.

Table 2.1: Summary of 3E/O ET Separation Maneuver Regions

When a region other than 0 is assigned and MECO has been confirmed, a contingency abort flag is set and 3E/O switches from display support to a guidance function responsible for calculating and commanding the necessary abort maneuvers. We turn now to the details of this guidance function.

2.2.2 3E/O Contingency Guidance

The primary tasks of 3E/O Contingency Guidance are to command the dissipation of excess fuel before and after ET separation, calculate the angle of attack and pitch rate for ET separation, monitor the `-Z` translation used to maneuver the Shuttle away from the ET after separation, calculate an angle of attack for entry, command the transition to the glide phase of the RTLS abort, and oversee the timing and sequencing of all of the above each guidance cycle during a contingency abort. The calculations mentioned here actually compute *quaternions*, four-element matrices that define coordinate transformations. For example, the quaternion calculated for the ET separation maneuver commands zero sideslip and the current bank angle, as well as the angle of attack. As Table 2.1 suggests, the characteristics of the maneuver commanded by the quaternion, as well as the sequence and timing of events, vary depending on the maneuver region dictated by prevailing conditions and ascent status.

2.2.2.1 Pre-ET Separation Maneuver for Region 102

The primary consideration for first-stage (i.e., flight mode 102) aborts is sequencing the pre-separation maneuvers with SRB tailoff, i.e., using up remaining SRB propellant. Once the SRB chamber pressures fall below an I-loaded-value indicating that enough of the propellant has been consumed, ET separation is commanded. Prior to separation, if the Shuttle's altitude is sufficient to permit a dump of OMS fuel, a one-time only firing of the OMS engines is commanded. No further attitude maneuvers are commanded prior to separation for region 102, because atmospheric density and dynamic pressure are too high to safely maneuver without risking the structural integrity of the shuttle.

2.2.2.2 Pre-ET Separation Maneuvers for Regions 3 and 4

The primary concern for region 3 aborts is high dynamic pressure, which can cause aerodynamic moments to exceed the ability of the RCS jets to perform the pitch-down to a near-zero angle of attack. If this occurs, the region is switched from 3 to 4, the pitch rate is zeroed, and a region 4 quaternion is calculated. Otherwise, a region 3 quaternion is calculated, commanding the near-zero angle of attack, along with a slight pitch rate (to compensate for flight-path angle rotation). When the appropriate pre-separation maneuver has been completed, ET separation is initiated. Since these calculations are repeated each guidance cycle, the switch from region 3 to region 4 can occur at any point after an I-loaded amount of time has elapsed prior to ET separation. If control limitations prevent the simultaneous satisfaction of all necessary parameters – in this case, angle of attack, sideslip, and body rates (i.e., pitch, yaw, and roll rates) – separation is commanded as soon as the angle of attack (the most crucial parameter for the separation maneuver) equals or exceeds an I-loaded threshold.

2.2.2.3 Pre-ET Separation Maneuvers for Regions 1 and 2

Region 2 aborts are referred to as "high-rate separation maneuvers" and are used when the Shuttle is coming down with a fairly high angle of attack and the dynamic pressure is moderate, or when a higher-level control function commands a region 2 maneuver to accommodate the transition from a 2E/O to a 3E/O guidance function. The quaternion calculated each guidance cycle when a region 2 abort is in progress establishes zero sideslip. When yaw and roll rates are small, a $-4\,\mathrm{deg/sec}$ pitch rate is commanded. Under certain conditions, the time required to zero the sideslip is sufficiently long that dynamic pressure builds to unsafe levels. Under these circumstances, if a region 2 abort has not been explicitly commanded, the region is switched from 2 to 1 and an attitude-independent separation maneuver is begun immediately. Otherwise, the pitch rate maneuver is begun as soon as sideslip is below an I-loaded threshold and yaw and roll rates are acceptably small. Separation is commanded as soon as the difference between the desired and established pitch rates is acceptably small, or when too much time has elapsed since sideslip was zeroed.

As noted above, region 1 separation is attitude independent, i.e., there is no attempt to modify the Shuttle's attitude prior to separation, which is commanded after body rates have become acceptably small and dynamic pressure has dropped to a safe value. If the rate of descent is too great or too little time remains for a maneuver to entry attitude, then separation is initiated immediately after body rates and dynamic pressure reach safe values. Otherwise, up to 14 seconds are allowed to permit the RTLS separation sequence to complete.

2.2.2.4 Post-ET Separation Maneuvers (All Regions)

The post-separation maneuver sequence is identical for all five abort regions. Immediately after ET separation, a `-Z` translation is commanded to maneuver the Shuttle away from the ET[4]. When this translation has been completed, excess OMS propellant, if any, is burned by firing the two OMS engines and selected aft RCS jets. RCS and OMS jets are interconnected for this (and other) purposes via a series of valves that can be configured either automatically or manually. The venting of excess OMS propellant occurs exactly once in the post-ET maneuver sequence for regions 1-4, but is not used in region 102; the time between the pre- and post-separation dumps in a first-stage abort may not be sufficient to satisfy OMS/RCS system constraints. When the RCS jets become available following the OMS/RCS interconnect (if any), an entry-attitude quaternion is commanded, establishing level wings, zero sideslip, and an angle of attack based on the Shuttle's current relative velocity. The final step in the post-separation sequence is to command an automatic transition to the glide phase of RTLS abort as soon as attitude and body rates reach acceptable ranges or dynamic pressure exceeds an I-loaded threshold.

2.3 The Existing Requirements Analysis Process

The process used to develop shuttle flight-software requirements documents typically yields a textual description, possibly accompanied by diagrams such as data-flow diagrams, state charts, or object diagrams. If diagrammatic material is provided, it is considered secondary; the primary requirements document is the informal English description. This phase is followed by the development of high-level test plans and product assurance activities, including Fagan-style inspections [Fag76,Fag86] of both the requirements and the test plan. Both the requirements and the test plan also undergo a baseline review before work on lower-level engineering products is allowed to proceed.

Since the 3E/O document we are working from is a Change Request (CR) [Roc94], the similarly well-defined process for modifying an existing requirements document is of

[4] A `-Z` translation moves the Shuttle downward along the `Z` axis, which runs parallel to the Shuttle plane of symmetry, and perpendicular to the `X` axis, which runs parallel to the Shuttle body, with the `Y` axis completing the right-handed orthogonal system.

particular interest. The process begins when an engineer writes a requirements Change Request (CR), documenting modifications to a Shuttle flight software system. A CR typically goes through several drafts with the author interacting primarily with the Requirements Analyst (RA) responsible for the given software (sub)system. When the author and RA agree that the CR correctly captures the requirements, the CR is submitted to a review board. The CR is then prioritized along with other CRs for consideration during a formal review conducted by a group of RAs. The formal review process includes the following:

- Preparation of an engineering assesment including a summary, of the proposed change, a justification, and an analysis of its potential impact on the software system.
- Detailed analysis of the CR guided by a Requirements Inspection Checklist containing generic error categories. Errors found are documented on an Issue Form.
- One or more Formal Inspections of the CR, as needed (depending on factors such as CR size and complexity), conducted by a team consisting of the CR author, RA, developer, verifier, etc., to review issues found during analysis, to compile a list of items that must be investigated before implementation of the CR may procede, and to insure that all participants have a consistent understanding of the requirements.
- Tracking and resolution of all outstanding issues.
- Baselining; when all issues have been resolved ("closed"), the CR is baselined and scheduled for implementation.
- Collection and analysis of quality metrics, primarily the number of issues detected during inspection and the number of problems encountered after the requirements evaluation (RE) phase. The ratio of the former to the later yields a rough quality metric (the "process error rate") for RE.

Although the processes sketched here for both the initial development and the modification of requirement documents – including ongoing quality assurance activities – are considered effective, [NASA93, pp. 9,22] notes the limitations paraphrased below:

- Current techniques are largely manual and highly dependent on the skill and diligence of individual inspectors and review teams.
- There is no methodology to guide the analysis process, no structured way for RAs to document their analysis, and there are no completion criteria.
- Although these techniques catch a substantial number of defects, the density of defects found during requirements analysis suggests that many errors escape detection.

- NASA projects using currently-available techniques have reached a quality ceiling on critical software subsystems, suggesting that innovations are needed to reach new quality goals.

These limitations contribute a significant part of the rationale for exploring the use of formal methods – including the state exploration technique described below – as a strategy for complementing and enhancing the existing requirements analysis process.

Chapter 3

Formal Analysis of Contingency 3E/O Sequencing

The series of sequential maneuvers described in the informal description of 3E/O can be viewed as mode-sequencing, i.e., conditioned events or behaviors that occur in an order prescribed by the satisfaction of one or more constraints. Characterizing the constraint satisfaction that defines the permissible sequences of mode transitions is sometimes referred to as a *mode-sequencing problem*. Abstracting the 3E/O algorithm to its most basic precepts yields just such a series of sequential steps conditioned, in this case, by a context consisting of variables representing the Shuttle's external (physical) environment and internal state. As such, 3E/O is too procedural to be a good candidate for the type of formal specification and proof supported by systems like PVS [ORSvH95], but can be quite naturally modeled as a finite-state system. This observation led us to wonder whether the type of mode-sequencing exemplified by 3E/O could be verified most effectively using finite-state analysis techniques.[1]

3.1 Finite-State Verification Techniques

Finite-state verification techniques, described in [ZWR+80, CG87, BCM+92, Kur93, McM93], have been around in one guise or another since at least the late 1970's. These techniques were first used for verifying protocols (described as a collection of communicating finite-state machines) and have been applied almost exclusively to hardware or software implementation of control algorithms such as communication [HK90, Sha93] and cache coherence protocols [ID93]. Although to our knowledge, no previous attempts

[1] The quaternion calculations are the only computational component of 3E/O. Although these calculations constitute an important 3E/O output, the matrix manipulations involved are relatively simple, well understood operations. We model quaternion calculation with (only) enough granularity to ensure that a quaternion value is appropriately assigned, but otherwise focus exclusively on basic sequencing properties.

have been made to apply finite-state verification techniques to the type and scale of mode-sequencing problem described here, the approach seemed worth exploring. The rationale is as follows.

Like most fault-handling logic, 3E/O consists largely of mode switching and exception handling. The input and state spaces of these types of applications tend to lack a regular and easily characterizable structure and there is typically little or no algorithmic complexity. Typechecking and proof of invariants to establish correctness of functional requirements are therefore not well-suited to fault-handling applications. The simplest way to validate these systems is to enumerate the entire input and state spaces by brute force. While this is rarely practical—the state space of most applications of interest is far too great to make such brute force enumeration feasible—it is often possible to "downscale" the state space of an application to a fairly small finite size and still retain the essential behaviors of the original system.[2] The properties to be checked are generally specified as explicit assertions or as error-checks programmed into the component specifications. Since this approach checks that all reachable states satisfy the given properties, it is also referred to as "reachability analysis."

3.2 Modeling Issues and Strategy

As noted above, the 3E/O algorithm consists of a series of sequential maneuvers that are readily modeled as a finite-state machine. The only real difficulty derives from the number and characteristics of the input variables. The 3E/O requirements document contains six full, double-spaced pages of inputs, most of which represent I-loaded thresholds used to calculate the order and timing of the maneuver sequences. Nevertheless, the sheer number of inputs is less problematic than their inherent complexity. Even a simple model of the physics of the Shuttle's ascent is beyond the scope of this project.

The modeling strategy we developed is based on the following assumptions. First, there is no need to confront head-on the inherent complexities of the real physical world. For example, we care only whether the current altitude and altitude rate predict an apogee altitude greater or less than the calculated altitude-velocity curve; the exact values or physical laws involved are irrelevant for verifying sequencing properties. As a result, we don't have to explicitly model the largely continuous values documented in the requirements, but can instead use either qualitative ranges or booleans.

Second, we make the further simplifying assumption that there are no constraints on the simultaneous values assumed by variables representing physical parameters, i.e., we ignore physical constraints between physical parameters and assume that all such parameters are completely independent. For example, we make no attempt to capture the relation between velocity and altitude. Although this is clearly naive, it is also

[2]In fact, experience suggests that ennumerating *all* behaviors of a downscaled system is a considerably more effective debugging method than exploring *some* of the behaviors of the original system.

overly general, implying that while we may consider too many cases, we do not overlook any.[3]

Finally, we largely ignore time, except for the implicit notion of time inherent in an ordered sequence of events. These three assumptions provide a reasonably tractable and accurate model of the 3E/O input space.

In addition, we reduce the large number of inputs by exploiting the fact that a boolean-valued operation on two inputs is equivalent, as a sequencing constraint, to a simple boolean variable. For example, the boolean expression used to determine if the Shuttle has sufficient range to make it back to the runway checks whether the down-range horizontal earth-relative velocity is strictly less than 0, and the difference between the predicted and actual range capability is strictly greater than an I-loaded minimum acceptable range difference, i.e., `v_horiz_dnrng < 0 AND delta_r > del_r_usp`. The value of the operation in each conjunct is either true or false, and hence for our purposes, indistinguishable from that of a boolean-valued variable. In other words, as a sequencing constraint, this conjunction is equivalent to the expression: `v_horiz_dnrng_LT_0 AND delta_r_GTR_del_r_usp`, where each conjunct is reduced to a simple boolean variable. By universally quantifying over these variables, we effectively show that for all possible values of the two (original) expressions, certain properties hold. We use this strategy for all inputs that represent the physical environment. Inputs that represent the Shuttle's internal state, e.g., the flag that indicates whether contingency 3E/O has been activated or the variable that encodes whether MECO has occurred, are modeled as inputs. Another way to view the modeling approach taken here is that we define a state transition system operating within a two-level context: a global environment consisting of monitored variables for external physical aspects, i.e., variables representing sampled sensor values, and a local environment consisting of monitored variables for Shuttle-internal physical aspects, i.e., variables representing the Shuttle's internal status. This view is similar to the standard A-7 model [HKPS78, Hen80, vS90], except that the distinction made here between "external" and "internal" environment variables would probably not appear as such in a standard A-7 interpretation.[4]

3.3 Finite-State Specification of Three-Engines-Out

Murϕ, the finite-state verifier used in this study was developed by David Dill and his students at Stanford University [DDHY92, ID93][5] and consists of the Murϕ Compiler and the Murϕ description language for finite-state asynchronous concurrent systems,

[3]In the context of finite-state verification, a technique which prides itself on being able to handle very large (but finite) state spaces, it is far better to consider too many possibilities, than too few.

[4] [vS90] distinguishes three types of environmental state variables: *application*, *established* and *hardware-dependent*, but these distinctions are implementation-based, whereas ours reflect a somewhat different bias.

[5]Murϕ is named after the author of the irrefutable law which states that "The bug is always in the case you didn't test."

which is loosely-based on Chandy and Misra's Unity model [CM88] and includes user-defined datatypes, procedures, and parameterized descriptions. A Murϕ description consists of constant and type declarations, variable declarations, rule definitions, start states, and a collection of invariants. The Murϕ compiler takes a Murϕ description and generates a C++ program that is compiled into a special-purpose verifier that checks for invariant violations, error statements, assertion violations, deadlock, and (in certain versions) liveness. Murϕ can be used as a verifier or as a simulator. The verifier attempts to enumerate all possible states of the system and the simulator explores a single path through the state space. In both cases, efficient encodings, including symmetry-based techniques, and effective hash-table strategies are used to alleviate state explosion. To date, we have used Murϕ exclusively as a verifier.

The 3E/O specification is written in two parts, one for each of the two main 3E/O functions: region selection and contingency guidance. The region selection function is quite small and was specified first in order to refine the modeling approach and analysis strategy before tackling the considerably larger guidance function.

We illustrate the use of Murϕ to specify 3E/O with a somewhat abridged version of the region-selection algorithm. We actually implemented two versions of this algorithm, one in the so-called "protocol-style" which is highly nondeterministic and therefore rather difficult to follow, and a more readily understood functional version, part of which we present here.

To begin, we need a way of talking about variables that are not simple booleans. As noted earlier, we would like to do this as abstractly as possible, using types that range over qualitative rather than quantitative domains. Thus for the highly restricted purpose of this specification, we need only three types of values for velocity: those greater than a maximum threshold, those greater than a minimum threshold, and those less than or equal to the minimum threshold. We use a similarly abstract domain for dynamic pressure. The other two type definitions explicitly enumerate domains representing the relevant flight modes and the maneuver regions. Since region 4 is not assigned by the region selection function (for reasons discussed in Section 2.2.1), it does not appear as an element of the domain of type `region`. The value `regE` denotes errors and is introduced strictly to make it easier to track error states.

```
Type
   velocity: enum{GTR_vi_3eo_max, GTR_vi_3eo_min, LEQ_vi_3eo_min};

   dynamic_pressure: enum{GTR_qbar_reg3, GTR_qbar_reg1, LEQ_qbar_reg1};

   major_mode: enum{mm102, mm103, mm601};

   region: enum{regE, reg0, reg1, reg2, reg3, reg102};
```

We also need a handful of global variables. The first four variables are flags indicating whether a 3E/O contingency abort has been signaled, whether the SSMEs have been shut down (main engine cutoff is a necessary condition for ET separation), whether a high-rate ET separation (i.e., maneuver region 2) has been signaled, and whether a region has been assigned, respectively. Two other global variables are introduced: one of type `major_mode` and one of type `region`.

```
Var
  cont_3EO_start,
  meco_confirmed,
  high_rate_sep,
  region_selected: boolean;
  m_mode: major_mode;
  r: region;
```

We next define the basic region selection function. To conserve space in the presentation, we have not included declarations for the functions `Nominal_Ascent` and `RTLS_Abort`, referenced in `reg_sel`. `Nominal_Ascent` checks to see if the ascent is normal and, if not, calls a function that assigns an appropriate maneuver region. `RTLS_Abort` monitors the abort phase. `reg_sel` assigns a maneuver region based on the current flight mode (`m_mode`) and the set of variables representing the prevailing physical conditions. `m_mode` is a global variable nondeterministically set by rules (see below). The formal parameters provide the external context that conditions the selection of the maneuver region, several of which should be familiar from the previous discussion on modeling. In the course of state exploration, `reg_sel` is called with all possible combinations of values for these variables. The `switch` construct introduces a standard case expression.

```
Function reg_sel(vel:velocity; q_bar:dynamic_pressure;
                 delta_r_GTR_del_r_usp, v_horiz_dnrng_LT_0,
                 alpha_n_GTR_alpha_reg2,h_dot_LT_hdot_reg2,
                 apogee_alt_LT_alt_ref:boolean): region;
Begin
  Switch m_mode
  case mm102: r := reg102; cont_3EO_start := true;
  case mm103: r := Nominal_Ascent_ck(vel,q_bar,apogee_alt_LT_alt_ref,
                                   alpha_n_GTR_alpha_reg2,
                                   h_dot_LT_hdot_reg2);
  case mm601: r := RTLS_Abort(q_bar,v_horiz_dnrng_LT_0,
                             delta_r_GTR_del_r_usp, h_dot_LT_hdot_reg2,
                             alpha_n_GTR_alpha_reg2);
  Endswitch;
Return r;
Endfunction;
```

The rule below uses the previously defined `reg_sel` function to assign an appropriate region and, if conditions warrant and SSME cutoff has been confirmed, to initiate an RTLS abort. Murϕ rules define nondeterministic state transitions. The boolean expression that precedes the arrow (`==>`) defines a condition that characterizes the states under which the body, i.e., the statements following the arrow, may be executed. The *Ruleset* construct is syntactic sugar that generates a copy of the rules within its scope for every value of the bound variable. For example, the variable `vel` ranges over the enumerated type velocity, and therefore can take on the following three values: `GTR_vi_3eo_max`, `GTR_vi_3eo_min`, and `LEQ_vi_3eo_min`. There is a single rule, `Select Region`, within the scope of this ruleset, so the ruleset will generate three rules identical in all respects except for the value of the variable `vel`. The deeply nested rulesets are used to generate all possible combinations of values for the variables that model the Shuttle's ascent environment. As noted previously, this approach yields a complete, if overly simplistic, model of the input space.

```
Ruleset vel: velocity Do
Ruleset apogee_alt_LT_alt_ref: boolean Do
Ruleset q_bar: dynamic_pressure Do
Ruleset h_dot_LT_hdot_reg2: boolean Do
Ruleset alpha_n_GTR_alpha_reg2: boolean Do
Ruleset v_horiz_dnrng_LT_0: boolean Do
Ruleset delta_r_GTR_del_r_usp: boolean Do

Rule "Select Region"
  !cont_3EO_start
==>
Begin
  r:= reg_sel(vel,q_bar,delta_r_GTR_del_r_usp,v_horiz_dnrng_LT_0,
            alpha_n_GTR_alpha_reg2,h_dot_LT_hdot_reg2,
            apogee_alt_LT_alt_ref);
  region_selected := true;
  If meco_confirmed & r != reg0 Then cont_3EO_start := true Endif;
Endrule;

Endruleset;  -- delta_r
Endruleset;  -- v_horiz
Endruleset;  -- alpha
Endruleset;  -- h_dot
Endruleset;  -- q_bar
Endruleset;  -- apogee
Endruleset;  -- vel
```

The next set of rules illustrates the use of nondeterministic assignment to model the input of state variables such as `major_mode`, `meco_confirmed`, and `high_rate_sep`. Note that these rules allow multiple assignments of the same mode, as well as a change of modes (from 103 to 601) to reflect the transition from a normal ascent to an abort contingency, but correctly preclude the possibility of transitioning from a contingent abort to a normal ascent state.

```
Rule "set mm103"
  !cont_3EO_start & m_mode != mm601
==>
Begin
  m_mode := mm103;
Endrule;

Rule "set MM601"
  !cont_3EO_start
==>
Begin
  m_mode := mm601;
Endrule;

Rule "set meco_confirmed"
  !cont_3EO_start & !m_mode = mm102 & !meco_confirmed
==>
Begin
  meco_confirmed := true;
Endrule;

Rule "set high_rate_sep"
  !cont_3EO_start & !high_rate_sep
==>
Begin
  high_rate_sep := !high_rate_sep;
Endrule;
```

`Startstate` is a special type of rule that is executed exactly once at the beginning of a Murϕ execution. A startstate that assigns a value to every global variable is required of all Murϕ programs. Here, we need only initialize the (internal) state variables, since the (external) environment variables are denoted by universally quantified variables in Murϕ rulesets. Note the use of `undefine`, an as yet undocumented Murϕ feature that at least partially relaxes the stricture that every global variable must be initialized. In this case, we prefer to leave the value of `region` undefined until an appropriate region is calculated, rather than introduce another element of the enumerated type (`region`).

```
Startstate "Init"
Begin
  cont_3EO_start := false;
  m_mode := mm102;
  meco_confirmed := false;
  high_rate_sep := false;
  region_selected := false;
  Undefine r;
Endstartstate;
```

We model the basic mode sequencing properties of the Region Selection and Contingency Guidance functions as state transition constraints and then show that if these constraints are satisfied, key properties of the algorithms also hold. The key properties are typically specified as invariants, i.e., expressions that must be true in all states, and are illustrated in the following section.

3.4 Finite-State Analysis of 3E/O Sequencing

The 3E/O requirements document [Roc94] provides a procedural description of 3E/O, but does not identify essential properties of the contingency abort functions. In order to validate the Murϕ specification, we derived a handful of properties that capture the fundamental characteristics of mode sequencing intrinsic to 3E/O region selection and contingency guidance. These derived properties fall into two categories: properties of the form function `X` shall satisfy constraint `Y`, e.g., property 2 in Section 3.4.1 below, and those of the form the outputs of function `X` shall obey constraint `Y`, e.g., property 1 below.

We have approached 3E/O mode-sequencing as a constraint satisfaction problem, but it is worth noting that the sequencing constraints we are exploring constitute a partial order on the modes, i.e., any two comparable modes are ordered by a precedence relation that is reflexive, antisymmetric, and transitive. For example, the constraint that a MECO confirmation check must precede a contingency abort command can be expressed as $\texttt{meco} \leq \texttt{cont\_ab}$, where $\leq$ denotes the relation "precedes or co-occurs," and transitivity ensures that

$$(\texttt{region\_select} \leq \texttt{meco} \wedge \texttt{meco} \leq \texttt{cont\_ab}) \rightarrow \texttt{region\_select} \leq \texttt{cont\_ab}.$$

3.4.1 General Properties of Contingency Abort Sequencing

The contingency abort algorithms are functional, i.e., each set of inputs maps to a unique output – either a maneuver region in the case of Region Select or an abort maneuver in the case of Contingency Guidance. Additionally, each contingency abort function is defined by a prescribed sequence of calculations, wholly determined by the Shuttle's external environment and internal state.

1. For each possible set of inputs, there is exactly one contingency abort output defined by exactly one sequence of calculations.
2. The sequence of calculations that defines a contingency abort output is always executed in the prescribed order.

We can instantiate these properties for each of the two major Contingency Guidance functions, as shown below.

3.4.2 Essential Properties of Region Selection Sequencing

1. On each invocation, Region Selection shall assign and output exactly one maneuver region.
2. Region Selection shall assign a maneuver region according to a fixed sequence of calculations.
3. Region Selection shall initiate a contingency abort for region 102, and, if main-engine cutoff has been confirmed, for regions 1, 2, and 3.

Property 1 states the basic premise of Region Selection, namely this function always calculates and outputs a region. Additionally, we require that the assigned region be one of: region 0, 102, 1, 2, 3, or E, and that these regions be distinct. In theory, these two properties should follow from the enumerated region type

```
region: enum{regE, reg0, reg1, reg2, reg3, reg102};
```

but to be certain, we explicitly state the following two invariants.

```
Invariant "regions are pairwise disjoint"
 (regE != reg0 & regE != reg1 & regE != reg2 & regE != reg3 &
  regE != reg102 & reg0 != reg1 & reg0 != reg2 & reg0 != reg3 &
  reg0 != reg102 & reg1 != reg2 & reg1 != reg3 & reg1 != reg102 &
  reg2 != reg3 & reg2 != reg102 & reg3 != reg102);
Invariant "regions are exhaustive"
 (forall r:region (r=regE | r=reg0 | r=reg1 | r=reg2 |
                   r=reg3 | r=reg102));
```

Property 2 states that the prescribed order of calculation is respected when assigning regions. This property follows by inspection of the Murϕ constraints encoded in the rules and in `If..then..else` statements that constitute the bodies of the Murϕ functions for region selection. Property 3 is a particular instance of Property 2 and states that a contingency abort is initiated only after an abort maneuver region has been selected and MECO has been confirmed. We typically establish specific properties of this kind through invariants. The two forms of the invariant shown below are equivalent, although the invariant form is usually more efficient because the Murϕ compiler can exploit restricted properties explicitly identified as invariants. The error statement, `Error`, generates a run-time error.

```
Invariant "cont_3EO_start"
cont_3EO_start ->
   ((m_mode=mm102 | meco_confirmed) & !Isundefined(r) & r!=reg0);

Rule "alternate invariant cont_3EO_start"
  !(cont_3EO_start ->
       ((m_mode=mm102 | meco_confirmed) & !Isundefined(r) & r!=reg0))
==>
  Error "Invariant violated: alternate invariant cont_3EO_start"
Endrule;
```

Additional invariants were used to debug the specification and to explore implications of the requirements, e.g., by checking for suspected anomalies. For example, we suspected that a region 0 assignment could persist from one cycle to the next, resulting in failure to detect changes that would ordinarily trigger an abort maneuver. To test this hypothesis, we output the error region, `regE`, whenever we detected a region 0 assignment that did not satisfy the necessary constraints and added a simple invariant stating that `regE` never occurs. This strategy is equivalent to stating the following invariant

```
 (r = regE) -> !((m_mode = mm103 & (vel = GTR_vi_3eo_max |
                 (vel = GTR_vi_3eo_min & apogee_alt_LT_alt_ref))) |
                 (m_mode = mm601 & v_horiz_dnrng_LT_0 &
                 delta_r_GTR_del_r_usp)))
```

The resulting error trace is reproduced below. The trace illustrates nicely the type of debugging information provided via counterexamples generated by finite state verification techniques. Note the final (anomalous) state in which `delta_r_GTR_del_r_usp` becomes false during a presumably normal RTLS abort (flight mode 601). This corresponds to a scenario in which an RTLS abort has been invoked, but the fact that the Shuttle lacks sufficient range for an intact RTLS abort is not detected or signaled.

Although there is no possibility of an intact RTLS with three engines out, failure to detect or signal accurate range information could potentially further jeopardize crew safety.

```
Invariant no pathological region 0 cycles failed.

Startstate Init fired.
cont_3EO_start : false
meco_confirmed : false
high_rate_sep : false
region_selected : false
m_mode:mm102
r:Undefined
----------

Rule set MM601 fired.
cont_3EO_start : false
meco_confirmed : false
high_rate_sep : false
region_selected : false
m_mode:mm601
r:Undefined
----------

Rule Select Region, vel:GTR_vi_3eo_max, apogee_alt_LT_alt_ref:true,
v_horiz_dnrng_LT_0:true, delta_r_GTR_del_r_usp:true,
q_bar:GTR_qbar_reg3, h_dot_LT_hdot_reg2:true,
alpha_n_GTR_alpha_reg2:true fired.
cont_3EO_start : false
meco_confirmed : false
high_rate_sep : false
region_selected : true
m_mode:mm601
r:reg0
----------

Rule Select Region, vel:GTR_vi_3eo_max, apogee_alt_LT_alt_ref:true,
v_horiz_dnrng_LT_0:true, delta_r_GTR_del_r_usp:false,
q_bar:GTR_qbar_reg3, h_dot_LT_hdot_reg2:true,
alpha_n_GTR_alpha_reg2:true fired.
cont_3EO_start : false
meco_confirmed : false
high_rate_sep : false
region_selected : true
m_mode:mm601
r:regE
----------

End of the error trace.
```

The error reflected in this trace initially appeared significant, given that the algorithm fails to detect and signal two anomalous RTLS scenarios: the situation reflected in the error trace, where the Shuttle lacks suffient range for an intact abort, and the situation where the vehicle is heading away, rather than toward, the landing site. Checking with the RA responsible for this function, we learned that the termination criteria for this task specify that the Contingency Abort function terminates if region 0 is assigned. Apparently the fact that the algorithm currently checks each cycle for a region 0 assignment from the previous cycle is misleading and indicates a questionable redundancy in the requirements, rather than a serious oversight.

3.4.3 Essential Properties of Contingency Guidance Sequencing

The properties enumerated for contingency guidance similarly reflect the general characteristics of 3E/O Contingency Abort sequencing. The basic order of calculation follows the two main guidance phases: pre-ET separation and post-ET separation. In the pre-ET phase, additional ordering constraints are imposed by the operative maneuver region. In the post-ET phase, the maneuver region is irrelevant and the ordered sequence includes `-Z` translation calculations, entry maneuver calculations, and commanded transition to flight mode 602. For clarity, we specify the basic functions of the guidance algorithm separately from the sequencing constraints. Since the analysis strategy for contingency guidance is identical to that for region selection, we list these properties without further discussion.

- On each invocation, contingency guidance shall calculate and output commands for exactly one contingency abort maneuver.
- Contingency Guidance shall calculate an abort maneuver according to a fixed sequence of calculations.
- Contingency Guidance shall command an ET separation.
- Contingency Guidance shall command at most one interconnected OMS dump.
- Contingency Guidance shall calculate and output an entry maneuver.
- Contingency Guidance shall command a transition to glide RTLS (flight mode 602).
- The transition to mode 602 shall not occur until the entry maneuver has been calculated.
- The entry maneuver calculations shall not commence until the OMS/RCS interconnect, if any, is complete.
- The OMS dump shall not be considered until the `-Z` translation is complete.

- Completion of `-Z` translation shall not be checked until ET separation has been commanded.
- ET separation shall be commanded if and only if an abort maneuver region is assigned.

3.5 Findings

The finite-state specification and analysis of 3E/O produced the set of issues enumerated below in Sections 3.5.1 and 3.5.2. These issues represent the usual collection of undocumented assumptions, inconsistent and imprecise terminology, redundant calculations, missing initializations, interface anomalies, and logical errors invariably exposed through the process of formalizing and analyzing requirements. Predictably, the significance of the issues varies. Of the approximately 20 issues listed below and reported to the 3E/O requirements analyst (RA), roughly one-third were noted and will appear in an upcoming Documentation (Errata) CR. Other issues that we felt impacted the clarity and precision of the requirements, including implementation bias and redundant calculations (cf. Section 3.5.3), were not considered important by the RA. The logical error listed in item 2 of Section 3.5.2 represents a significant error in the requirements that was also discovered by the existing requirements analysis process. To our knowledge, the other issues listed below had not been previously discovered.

3.5.1 3E/O Region Selection Task Findings

1. Anomaly in the algorithm on cycles in which no abort is signaled: the algorithm currently checks each cycle for a nominal ascent pattern that precludes the need for abort procedures (region 0). The termination criteria for this task imply that the algorithm will never be reentered if region 0 is assigned, in which case the region 0 check is unnecessary.[6]
2. Redundancy in the algorithm after a high rate separation is commanded: assuming that the command cannot be rescinded on subsequent cycles, the corresponding region doesn't need to be recalculated each cycle, as is currently the case.
3. Inconsistent initialization of output parameters: some are initialized, others are not. For example, `ALT_APOGEE` and `ALT_REF` are initialized, but `T_DEL`, `G`, and `RTLS_ABORT_DECLARED` are not. If these output parameters are initialized by other functions, that information is not currently documented in 3E/O.
4. Unstated Assumption: MECO must be confirmed for region 102 before a contingency abort is initiated.

[6] More precisely, the exit should be restored to step 5 and step 6 should be eliminated.

5. Inconsistencies between informal description and stated algorithm, e.g.:

    - The informal description says "If the apogee altitude is above this curve, a contingency abort capability is still required." The algorithm checks whether the apogee altitude is *strictly less than* the computed altitude-velocity curve.
    - The informal description says "If the vehicle is heading back towards the landing site, and the current range is greater than an I-loaded value, a 3E/O region index is calculated." The algorithm calculates a region index if the current range is *less than or equal to* the I-loaded value.

6. Ambiguities in the informal English description; e.g.,

    - "Otherwise, the 3E/O field is blanked out and no further contingency abort calculations will be performed." This is ambiguous; no further calculations are performed on this cycle or on this and (all) subsequent cycles?
    - "After SRB separation, on every pass that the 3E/O region index is calculated, a check is made to see if MECO confirmed has occurred. If so, a check is made to see if the major mode is 103. If so, an RTLS is automatically invoked to transition to major mode 601. A 3E/O contingency start flag is then set ..." The description is ambiguous and could be interpreted to mean the contingency flag is set only if the major mode is 103.

7. Omissions from the algorithm or informal English description; e.g.,

    - The algorithm indicates that a display is updated, but fails to mention which of the relevant displays should be changed.
    - The informal English description has not been updated to reflect modifications to parameters `DELTA_R` and `DEL_R_USP`.

3.5.2 3E/O Contingency Guidance Task Findings

1. Interface anomaly: while there is an explicit transition to an RTLS abort mode (that signals termination of 3E/O) in all other regions, there is no such transition for a first stage abort. The reason for this anomaly: existing requirements and code used functions external to 3E/O to command the RTLS transition for first-stage aborts and these existing requirements were not modified when they were "integrated" into the current requirements.

2. Logical error: If a Region 2 ET separation is downmoded to a Region 1 separation in Step 23, the pitch rate, i.e., output variable `WCB2`, will not be set to appropriately reflect the downmode. The problem is particularly acute if the downmode occurs after the first pass, because commanded region 2 pitch rates continue when they should in fact be zeroed for region 1.

3. Unnecessary entry maneuver calculations: Step 22 calculates a region 2 quaternion before Step 23 checks to see if conditions dictate a downmode to region 1. Since region 1 maneuvers are attitude-independent, the calculation is not used if a downmode is required.

4. Unstated Assumptions:

   - The variable `ET_SEP_MAN_INITIATE` is set both internal and external to 3E/O. If it is set externally, it is possible to exit 3E/O via a region 1 maneuver without setting output variable `FRZ_3EO` (cf. Step 28). Apparently only the crew can cause this and the assumption is that under these circumstances, the crew will also take responsibility for attitude control.
   - In Step 6, the algorithm checks for `REGION 102` and for `REGIONS 1-4`. and exits if it detects any other region assignment. In fact, any other region assignment is an error, but the assumption is that testing and inspection will ensure that only correct region values are used and that Step 5, which guarantees an emergency separation if all else fails, will catch any remaining region-assignment errors.
   - After ET separation, dynamic pressure permitting, a one-time only interconnected OMS dump is performed following completion of the `-Z` translation. However, due to timing constraints, this dump is not performed if ET separation occurs in first stage. The algorithm currently initiates the dump maneuver prior to checking for a first-stage abort. As a result, the delay in the main propulsion system LO2 dump is zeroed, regardless of whether a first-stage ET separation has occurred and the dump is precluded. The operative assumption is that all region 102 post-ET separation dumps will be inhibited, but this appears inconsistent with zeroing the LO2 dump delay.
   - Output parameters, including `ET_SEP_MAN_INITIATE` and `ETSEP_Y_DRIFT`, that are not initialized in Contingency 3E/O Guidance are apparently assumed to be initialized elsewhere.

5. Inconsistencies between informal description and stated algorithm, e.g.:

   - The informal English description in Steps 8 and 9 does not make it clear that these steps must be performed one time only *in lockstep*, i.e., on the same cycle.
   - In Step 14, the English reads in part: "[if the] MET has exceeded an I-loaded value..." while the pseudo-code test checks for `=>`. Actually, the pseudo-code no longer checks MET, but the difference between the current running GMT (Greenwich-Mean-Time) and the GMT associated with liftoff, so the English needs to be updated in both respects.
   - The definition in the Table 4.10.1-1 entry for `P_MAX_REG1` states "Maximum pitch rate ...," but it should read "Maximum roll rate ..."

- Similarly, The definition in the Table 4.10.1-1 entry for `Q_MINUS_Z_MAX` states "Maximum pitch rate for MM102-to-MM602 transition," but it should read "Maximum pitch rate for immediate MM602 transition."[7]

6. Ambiguities in the informal English description; e.g.,

   - The conjugate of the VR-to-M50 quaternion is not `Q_M50_VR` (as the description and parentheses suggest), but rather `Q_VR_M50`.

3.5.3 Desirable Characteristics of 3E/O Algorithms

The findings enumerated above suggest certain characteristics that a "good" set of 3E/O requirements should exhibit. First, the requirements should specify that the contingency guidance algorithm calculate all and only necessary maneuvers. Assuming the current requirements are updated to correct the logical error responsible for failure to recalculate the ET separation maneuver following a downmode from region 2 to region 1 (cf. Section 3.5.2, item 2), the requirements appear to correctly specify calculation of all necessary maneuvers. However, the requirements also specify several unnecessary calculations, including 2 (Section 3.5.1) and 3 (Section 3.5.2). According to the 3E/O requirements analyst, these redundant calculations are not considered important because the shuttle currently uses only around 50% of the available compute cycles. However, the fact that the requirements assume the availability of compute cycles suggests an underlying implementation bias that distracts attention from the more fundamental nature of these algorithms. Thus a second desirable characteristic of a good set of 3E/O requirements is that the algorithm should focus on essential properties and behaviors and avoid implementation considerations.

A third and final desideratum is that the requirements should consistently maintain and explicitly state all underlying assumptions. For example, the current requirements for the region selection algorithm specify that before a contingency abort can be initiated, MECO must be confirmed. This confirmation is explicitly checked for all abort regions except region 102. Why isn't MECO confirmed for region 102 and what is the underlying requirement? In fact, the underlying requirement is exactly what you would expect: MECO must be confirmed in all abort regions before a contingency abort can be signaled. The confirmation is implicitly assumed rather than explicitly checked for major mode 102 ("first-stage") aborts because in first-stage aborts, region selection is executed (only) after MECO has been confirmed.[8] This is a nice example because it not only illustrates the type of implicit assumption frequently underlying these requirements, it also provides a clear, simple example of how implementation-level detail and

[7] Ron Avery pointed out the definition for `R_MINUS_Z_MAX`, as well as for `Q_MINUS_Z_MAX`, should read "maximum ... rate for resuming active attitude control in region 102 -Z maneuver."

[8] We suspect that this is another instance where existing requirements for region 102 were retrofitted into the current requirements without documentation or modification. As a result, implicit assumptions, special-case interfaces, and implementation considerations tend to mask the basic functionality that region 102 shares with the other abort-regions.

inconsistent functional interfaces can obscure the *real* requirements. To capture the requirements as given, the invariant would have to reflect the special case for region 102, i.e.,

$$\texttt{cont\_3EO\_start} \rightarrow (\texttt{m\_mode} = \texttt{mm102} \vee \texttt{meco\_confirmed}),$$

obscuring the essential property of the contingency guidance algorithm, namely that

$$\texttt{cont\_3EO\_start} \rightarrow \texttt{meco\_confirmed}.$$

Chapter 4

Discussion

In this final chapter we briefly explore how the current study relates to other work in this area, review the limitations of the method, and anticipate directions for future work.

4.1 Relation to Other Work

Although this work draws on several different formal-methods techniques, there is virtually no published work that is directly comparable. As noted in Section 3.1, finite-state verification techniques have been around for approximately twenty years, but have been used almost exclusively to verify hardware and software implementation of control algorithms such as communication and cache coherence protocols. As far as we've been able to determine, only one other application to the domain of software requirements has been documented. Atlee and Gannon [AG93] describe a largely automated technique for transforming tabular, SCR-style requirements [HKPS78] such as those used in [vS90] into a finite-state machine analyzable by the CTL model checker [CES86,McM93]. The technique is used to analyze safety properties of two examples, each with a small set of environmental conditions: an automobile cruise control system and a water-level monitoring system. The major difference between their approach and ours, aside from the nature and scale of the respective application domains, is the fact that their approach has been refined into an automated process and focuses on one style of requirements specification (SCR) entailing an explicit modeling bias, whereas our approach is manual and agnostic with respect to style of requirements specification and modeling strategy.

There is also little precedent for modeling continuous physical domains, such as the Shuttle ascent environment, since these domains are notoriously complex and not considered good candidates for current formal-methods techniques. Boyer, Green, and Moore discuss an initial investigation into applying formal methods to programs that interact with environments using a very simple example: "steering a vehicle down a straightline course in a crosswind that varies with time" [BGM82, 3], but apparently have not pursued this or similar examples on a more realistic scale.

Our use of qualitative values for physical entities is reminiscent of techniques used in certain subfields of Artificial Intelligence such as Qualitative Physics [WdK90] for modeling and reasoning about the physical world, but the analogy ends there. Our approach also differs from the simulation and scenario generation used extensively in industrial applications, including Shuttle requirements analysis (cf. Section 2.3), because state-exploration is exhaustive, checking all possible paths through the state space, whereas simulation and scenario generation test extensively, but are inherently non-exhaustive.

4.2 Limitations of the Method

There are two distinct, but related limitations to the approach taken in this study. First, developing a Murϕ specification differs very little from writing a program in a conventional programming language. As a result, it is difficult to regard a Murϕ specification as an intellectually compelling verification of the desired properties. This limitation is compounded by a second factor: the utility of a nondeterministic, rule-based notation is inherently more analytic than descriptive. Unlike a PVS requirements-level specification that provides a clear, unambiguous description that is potentially useful for formal calculation, informal analysis, reviews, inspections, and documentation, a Murϕ specification is not an effective descriptive document. In other words, although the rule-based Murϕ notation is an excellent vehicle for exhaustively analyzing the behaviors of finite-state systems and for checking their properties, it is a decidedly poor vehicle for communicating and documenting these systems and their properties.

4.3 Future Work

There is currently a great deal of interest in integrating various Formal Methods techniques to provide effective and efficient design and verification environments for complex systems. Combining general, but labor-intensive theorem proving techniques with the domain-specific, but highly automatic finite-state verification paradigm appears to be a promising approach. As discussed in [RSS95], several formal methods systems integrate model-checking and proof-checking, although to date there have been few, if any, truly effective integrations of these complementary technologies. Despite the shortcomings noted above, finite-state verification is a particularly useful tool for debugging formal specifications because of its speed, high-degree of automation, and ability to generate counterexamples. As noted at the end of Section 3.4.2, these failure traces, i.e., paths through the state space that terminate due to run-time errors, can be extremely useful for correcting and refining a specification. Used in conjunction with a richer, more flexible specification language and an automated theorem prover, the very real benefits of this technique can be exploited, while avoiding its weaknesses.

Exploring effective ways of integrating Murϕ-style state exploration with PVS-style specification and theorem checking is an example of this hybrid approach. There is already a BDD-based model-checker for the propositional mu-calculus available as a decision procedure within the PVS proof checker [RSS95]. There is also an experimental, automated translation from Murϕ to PVS and the reverse translation is under study. These translations will ultimately further automate integrated approaches in which properties that can be efficiently checked with state exploration and properties that require more general theorem-proving capabilities are both accommodated within a single system. We are also exploring largely manual points of integration, including the use of the PVS table construct to complement state-exploration, thereby providing perspicuous documentation, as well as the ability to explore additional properties of the 3E/O contingency abort function.

4.4 Concluding Remarks

The discipline of formal specification and analysis is invariably productive and the 3E/O study is no exception. The process of formally specifying the 3E/O requirements has exposed the usual collection of undocumented assumptions, logical errors, and inconsistent and imprecise terminology. Most of the errors were found in the process of scrutinizing and trying to understand the requirements document, but the finite-state verification provided additional assurance as well as documented counterexamples. Murϕ was particularly useful for exploring the large input space and verifying that assumptions about the cyclic behavior of the 3E/O Guidance Task were well-founded.

The 1993 assessment of Shuttle flight software development practices cited at the beginning of this report recommends that "...software safety programs must take advantage of state-of-the-art techniques and leading edge methodologies to build safety into the software and the system while enhancing software development capabilities." [NAS93, p. 91] Finite-state verification merits further study as we explore techniques, methodologies, and the integration of complementary paradigms for improving Space Shuttle flight software development.

References

[AG93] Joanne M. Atlee and John Gannon. State-Based Model Checking of Event-Driven System Requirements. *IEEE Transactions on Software Engineering*, 19(1):24–40, January 1993.

[BCC+95] R. Butler, J. Caldwell, V. Carreno, M. Holloway, P. Miner, and B. Di Vito. NASA Langley's Research and Technology Transfer Program in Formal Methods. In *Tenth Annual Conference on Computer Assurance COMPASS 95*, pages 26–30, Gaithersburg, MD, June 1995.

[BCM+92] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. *Information and Computation*, 98(2):142–170, June 1992.

[BGM82] Robert S. Boyer, Milton W. Green, and J. Strother Moore. The Use of a Formal Simulator to Verify a Simple Real Time Control Program. Technical Report ICSCA-CMP-29, Institute for Computing Science and Computer Applications, July 1982.

[CES86] E. M. Clarke, E. Emerson, and A. Sistla. Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.

[CG87] E. M. Clarke and O. Grümberg. Research on Automatic Verification of Finite-State Concurrent Systems. In Joseph F. Traub, Barbara J. Grosz, Butler W. Lampson, and Nils J. Nilsson, editors, *Annual Review of Computer Science, Volume 2*, pages 269–290. Annual Reviews, Inc., Palo Alto, CA, 1987.

[CM88] K. Mani Chandy and Jayadev Misra. *Parallel Program Design – A Foundation.* Addison-Wesley, Reading, MA, 1988.

[DDHY92] David L. Dill, Andreas J. Drexler, Alan J. Hu, and C. Han Yang. Protocol Verification as a Hardware Design Aid. In *1992 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 522–525. IEEE Computer Society, 1992. Cambridge, MA, October 11-14.

[Fag76] M. E. Fagan. Design and Code Inspections to Reduce Errors in Program Development. *IBM Systems Journal*, 15(3):182–211, March 1976.

[Fag86] M. E. Fagan. Advances in Software Inspection. *IEEE Transactions on Software Engineering*, SE-12(7):744–751, July 1986.

[Hen80] K. L. Heninger. Specifying Software Requirements for Complex Systems: New Techniques and Their Application. *IEEE Transactions on Software Engineering*, SE-6(1):2–13, January 1980.

[HK90] Zvi Har'El and Robert P. Kurshan. Software for Analytical Development of Communications Protocols. *AT&T Technical Journal*, 69(1):45–59, January/February 1990.

[HKPS78] K. L. Heninger, J. W. Kallander, D. L. Parnas, and J. E. Shore. Software Requirements for the A-7E Aircraft. NRL Report 3876, Naval Research Laboratory, November 1978.

[ID93] C. Norris Ip and David L. Dill. Better Verification through Symmetry. In *CHDL '93: 11th Conference on Computer Hardware Description Languages and their Applications*, pages 87–100. IFIP, 1993. Ottawa, Canada.

[Kur93] R.P. Kurshan. *Automata-Theoretic Verification of Coordinating Processes.* Princeton University Press, Princeton, NJ, 1993.

[McM93] K.L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1993.

[NAS93] *An Assessment of Space Shuttle Flight Software Development Practices.* National Research Council Committee for Review of Oversight Mechanisms for Space Shuttle Flight Software Processes, National Academy Press, Washington, DC, 1993.

[NASA93] *Formal Methods Demonstration Project for Space Applications – Phase I Case Study: Space Shuttle Orbit DAP Jet Select.* Multi-Center NASA Team from Jet Propulsion Laboratory, Johnson Space Center, and Langley Research Center, December 1993. NASA Code Q Final Report (Unnumbered).

[ORSvH95] Sam Owre, John Rushby, Natarajan Shankar, and Friedrich von Henke. Formal Verification for Fault-Tolerant Architectures: Prolegomena to the Design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, February 1995.

[Roc94] *Space Shuttle Orbiter Operational Level C Functional Subsystem Software Requirements: Guidance Navigation and Control – Part A Guidance Ascent and RTLS.* Rockwell International, Space Systems Division, OI-25, CR 90705h edition, June 1994.

[RSS95] S. Rajan, N. Shankar, and M.K. Srivas. An Integration of Model-Checking with Automated Proof Checking. In Pierre Wolper, editor, *Computer-Aided Verification, CAV '95*, pages 84–97, Liege, Belgium, June 1995. Volume 939 of *Lecture Notes in Computer Science*, Springer-Verlag.

[Sha93] W. David Shambroom. Use of Protocol Validation and Verification Techniques in the Design of a Fault-Tolerant Computer Architecture. In *Fault Tolerant Computing Symposium 23*, pages 636–640, Toulouse, France, June 1993. IEEE Computer Society.

[vS90] A. John van Schouwen. The A-7 Requirements Model: Re-Examination for Real-Time Systems and an Application to Monitoring Systems. Technical Report 90-276, Department of Computing and Information Science, Queen's University, Kingston, Ontario, Canada, May 1990.

[WdK90] Daniel S. Weld and Johan de Kleer, editors. *Readings in Qualitative Reasoning about Physical Systems.* Morgan Kaufmann, San Mateo, CA, 1990.

[ZWR+80] Pitro Zafiropulo, Colin H. West, Harry Rudin, D.D. Cowan, and Daniel Brand. Toward Analyzing and Synthesizing Protocols. *IEEE Transactions on Communications*, COM-28(4), April 1980.