SOFTWARE SAFETY ANALYSIS OF A FLIGHT MANAGEMENT SYSTEM VERTICAL NAVIGATION FUNCTION – A STATUS REPORT

Alan C. Tribble and Steven P. Miller,

Rockwell Collins, Cedar Rapids, Iowa

Abstract

We have developed a formal, executable model of the requirements for portions of the Vertical Navigation (VNAV) function of a Flight Management System and have conducted a software safety analysis on the model. In particular, we have performed a Functional Hazard Assessment in order to identify the potentially hazardous conditions associated with the VNAV function. We then conducted a Fault Tree Analysis and a Failure Mode Effects Analysis in order to identify the general categories of errors that relate to safety. By comparing these general categories to the system architecture, we were able to develop a list of specific safety requirements for the VNAV function. We then used formal methods tools to verify that the VNAV model satisfied the safety requirements.

We provide an overview of the safety analysis performed to date on the VNAV model, and compare and contrast these results to a similar analysis performed on the mode logic of a Flight Guidance System. Because the Flight Guidance System model was constructed entirely from Boolean logic it was easily analyzable with model checkers. The VNAV model involves continuous logic, (altitude and position values), and requires the use of theorem provers. The results of this analysis provide insight into the feasibility of integrating formal methods tools into the safety analysis process in a Model Based Development environment.

Acknowledgements

This work was supported, in part, by the NASA Aviation Safety Program under contract NCC01-001 with the NASA Langley Research Center. Daniel O'Brien, of the University of Minnesota, and Lucas Wagner, of Iowa State University, are also commended for their assistance.

Introduction

The Problem Domain

As Figure 1 shows, the avionics architecture of a typical regional jet aircraft is comprised of many separate systems and subsystems. Two key functions are the Flight Management System (FMS) and the Flight Control System (FCS). The FCS is composed of a Flight Guidance System (FGS) that generates roll and pitch guidance commands, and an Auto-Pilot (AP) that executes them. In comparison, the FMS is responsible for a more diverse set of functions requiring complex logic, Table 1.

Table 1. The Flight Management System is Responsible for Providing Functionality in Eight Key Areas.

Flight Management System Functions
• Airplane Performance
• Flight Planning
• Lateral Navigation (LNAV)
• Pre-Flight Initialization
• Radio Tuning
• Route Planning
• Thrust Management
• Vertical Navigation (VNAV)

The FMS has knowledge of the flight plan, which specifies the desired navigational details of the flight from takeoff to landing. As shown in Figures 2 and 3, the flight plan contains both lateral and vertical data.

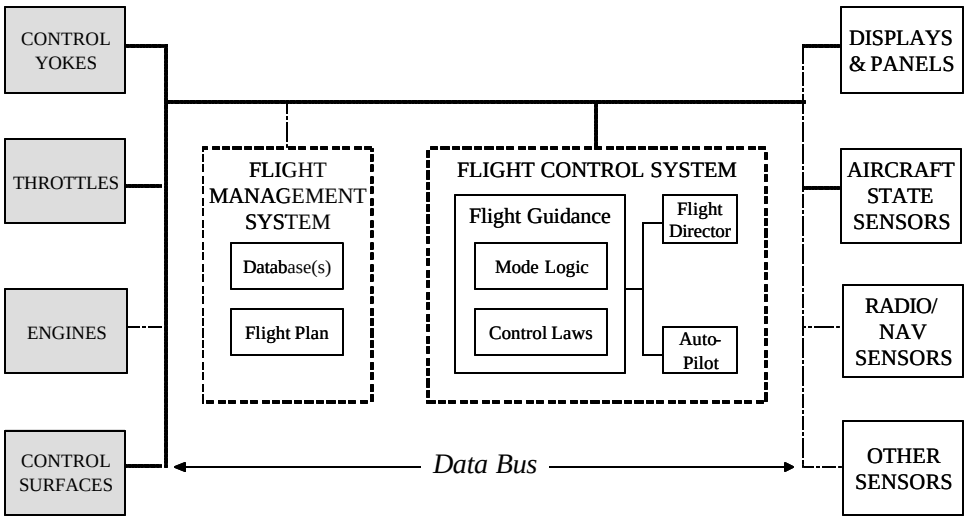

Figure 1. The High Level Architecture of an Avionics System.

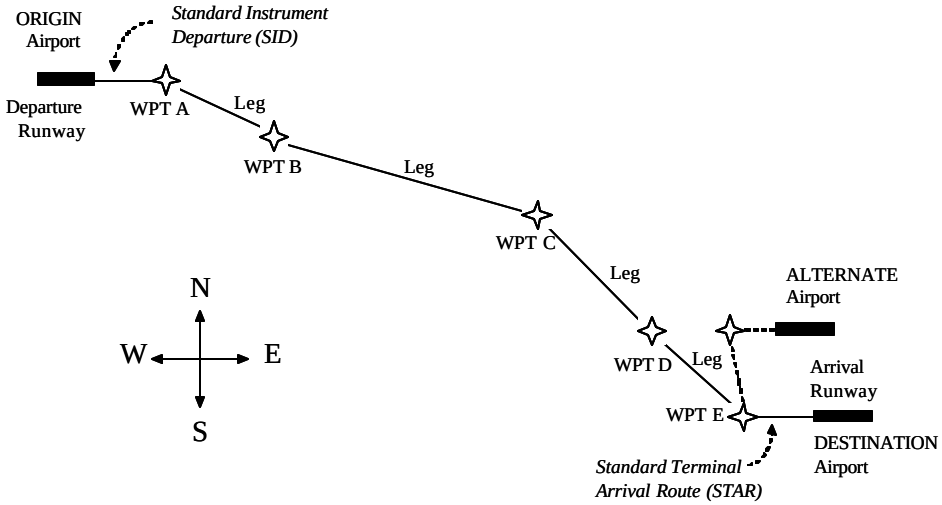

Figure 2. The Lateral Flight Plan.

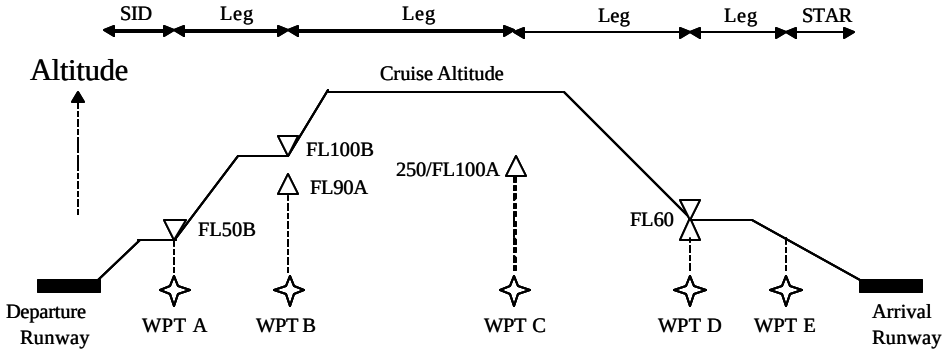

Figure 3. The Vertical Flight Plan.

Of particular interest is the interaction between the FMS and the FGS. With knowledge of the actual aircraft location the FMS can send commands to the FGS, which in turn commands the AP, in order to enable the aircraft to follow the flight plan. When engaged, the FMS Lateral Navigation (LNAV) and Vertical Navigation (VNAV) functions act like a silent crew member in the cockpit, ordering mode change requests and resetting target altitude values. Because of the safety critical nature of vertical navigation, and also the additional complexity that a change in altitude (and the associated change in energy) requires, the interaction between the FMS VNAV function and the FGS is of great concern.

The detailed behavior expected of a VNAV function is specified by ARINC 702A-1, [1]. The ARINC standard provides a listing of over one dozen separate areas of responsibility for VNAV. We have implemented in our FMS VNAV model those VNAV functions that interact, either directly or indirectly, with the FGS as shown in Table 2.

Table 2. Selected VNAV Requirements for a Typical Flight Management System.

Trajectory Prediction	Vertical Guidance
Provide Location of Top of Climb	Provide Altitude Targets That Enforce Altitude Compliance
Provide Location of Top of Descent	Provide Mode Commands to the FGS
	Provide Flight Phase Switching

Previous Work

Previously, we conducted a software safety analysis on a formal model of the FGS mode logic, [2]. This logic, which was entirely Boolean in nature, was representative of the complexity of an actual aircraft system. It was shown that industrial sized problems of this nature could be analyzed through the use of model checkers, a class of formal methods tools that explores the entire possible state space of all allowable solutions in order to identify any potential states that violate certain properties. We showed that using the NuSMV model checker, it was possible to verify over 300 properties of the FGS model in about an hour. This verified the conclusion that formal methods are ready for industrial use.

In contrast to the FGS, the FMS VNAV function is comprised of continuous equations involving integers and reals. As such, it is a very different kind of software model than the FGS. Because of the different nature of this model, and because of the inherent safety critical nature of the VNAV function, we have extended our previous study to investigate the feasibility of using formal methods tools on the FMS VNAV model. The sections that follow summarize the preliminary results of the study.

Software Safety Analysis

Background Information

Underlying our analysis is an assumption about the nature of accidents as illustrated in Figure 4. As shown, errors (in requirements, implementation, operation, …) can generate faults, which in turn can lead to failures, which lead to hazards, which can result in accidents. Our safety analysis therefore focuses on defining the hazards, failures, faults, and errors that could lead to accidents so that we can verify that no errors are present in the VNAV model. That is, we seek to verify that the VNAV model has been implemented correctly and does not contain any errors that could place the system in a hazardous condition. As is described in later sections, our analysis will use a combination of standard techniques, (e.g., Fault Tree Analysis and Failure Mode Effects Analysis), in combination with non-traditional, yet very powerful, formal methods techniques.

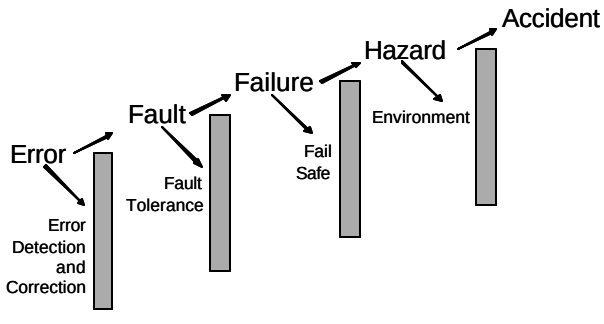

Figure 4. The Sequence of Events Leading to an Accident.

Specifying the Requirements

A specification of the FMS VNAV function has been generated in a formal language, the Requirements State Machine Language without Events (RSML^{-e}). RSML^{-e} is a synchronous language that was originally used to specify the Traffic Collision Avoidance System II (TCAS-II). It is important to note that the RSML^{-e} model was ultimately adopted by the FAA as the official specification for TCAS-II. As its name implies, RSML^{-e} eliminates the use of events. RSML^{-e} is similar to SpecTRM-RL, developed by the Safeware Engineering Corporation, and SCADE, developed by Esterelle Technologies, but has a slightly different syntax and underlying philosophy. RSML^{-e} runs in the "Nimbus" environment developed by the Critical Systems Research group at the University of Minnesota. An important advantage of RSML^{-e} is that it possesses a precise formal semantics so that the models can be formally analyzed, [3].

Defining the Safety Properties

The first step in the safety analysis process is to formally define those properties of the software associated with safety. Safety properties were generated via a Bi-Directional Analysis (BDA) technique, [4]. The starting point for the BDA is the list of hazards. Top-down analysis is then used to trace the hazards down to the related errors. To close the loop, an independent bottom-up analysis is then used to trace the errors back up to hazards.

Defining the Hazards

Safety is a system level problem and aviation safety standards ARP 4754 and ARP 4761 specify that safety analysis be performed both at the aircraft level and at the system level, [5, 6]. The aircraft level hazards are generally very few, such as loss of control. If the loss of control hazard is examined, it can be found that failures in a number of systems, (e.g., hydraulic lines, control yokes, flight control surfaces,), could give rise to it. However, as stated before, we are interested in defining the hazards for the FMS VNAV function. These hazards will derive from functional failures and are defined in a Functional Hazard Assessment (FHA).

We started with the functional requirements for the FMS VNAV, which were defined in Table 2. Examining the consequences of the FMS VNAV failing to provide this functionality identified the associated hazards. Each of these hazards was then assigned a level of criticality in accordance with DO-178B and MIL STD 882, [7, 8]. We identified 5 Level D (Minor) hazards, 8 Level C (Major) hazards and 1 Level B (Catastrophic) hazard in our model. (It should be noted that there are likely other hazards associated with the VNAV function on actual systems, but our analysis will address only those hazards present in our model.) Because Level B is the most critical hazard, the FMS VNAV is considered a Level B system. The FHA for the Level B hazard is shown in Table 3. This hazard forms the starting point for the next stage of the analysis.

Top-Down Analysis

Fault Tree Analysis (FTA) is a top-down analysis technique that is used to identify the contributing elements (errors / faults / failures) that could precipitate the system level hazards identified [9, 10]. FTA is a feed-back technique in that one starts with the system level hazards and attempts to work backward by identifying all possible causes of the hazards. Although the name implies that the technique is limited only to “faults”, it should be emphasized that FTA is a general, visual technique that is used to trace higher level events (such as hazards) down to their contributing events. These contributing events could be failures, or errors, in addition to faults.

Table 3. The Functional Hazard Assessment Identified One Level B Hazard.

Functional Failure (Hazard)	Critical Operational Phase	Aircraft Manifestation	Comment
Allows Aircraft to Descend Through PSA.	Descent	Aircraft Drops Below the Pre-Select Altitude (PSA), Violating Pilot Defined (and possibly Air Traffic Control Mandated) Altitude Limit.	During Approach the Aircraft is Allowed to Fly Through the PSA.

In an actual aircraft program, the FTA would start with the system level hazards, for example, Loss of Control, and include all aircraft systems that could potentially contribute to such a hazard. For our purposes, the FTA will start with the hazards identified in the FHA. Note that because safety is a system level property the FTA must include elements that input information to the FMS VNAV, such as the FGS, and elements that the FMS VNAV outputs information to, such as the PFD. By performing a FTA on each of the hazards it is possible to a listing of the individual base events that could contribute to hazardous conditions. For example, the FTA associated with the hazard "Allows Aircraft to Descend Through PSA." is shown in Figure 5. As shown at least 13 separate events could precipitate this hazard.

Bottom-Up Analysis

As a check of the results from the top-down FTA is a bottom-up Failure Modes Effects Analysis (FMEA). FMEA is a feed-forward technique in that the starting points for the analysis are possible errors, which are then traced forward to see if they have any impact on system safety (i.e., if they lead to potential hazards), [9, 10]. As with the FTA, the term FMEA should not imply that the results are limited to "failures". FMEA is a general analysis method that flows errors (or faults or failures) forward to hazardous conditions.

The output of a FMEA is a tabular presentation that lists: a) failure mode (error); b) effects (hazard); and c) analysis (interpretation). The starting point for the FMEA is the list of errors identified in the FHA. For example, a portion of the FMEA for the level B mode logic hazard is shown in Table 5.

Safety Properties

Once the categories of errors that can result in a hazard have been identified, they are in further examined in order to identify the specific properties of the model that could produce the higher level events. For example, the properties associated with "Error in Flight Phase Logic" are shown in Table 6. Note that many of these "safety" properties look like functional requirements. In this sense, we have identified those requirements that are directly related to safety. Verifying that these design requirements are indeed artifacts of the model we have constructed is an important step – one that is often omitted because it is so difficult.

Our analysis of the full model is still underway. However, we anticipate that the final results will identify over 100 separate requirements / safety properties that we will wish to verify in the model.

Translating the Requirements Model and Safety Properties into Analysis Tools

Once the safety properties are defined, the next step in the safety analysis process is to express the requirements model and safety properties in the same formal language. It is certainly possible to define the requirements and safety properties in the same language initially, but for real projects this will rarely be the case. The requirements model will probably be developed in tools like RSML^{-e}, SCADE, or Simulink. Safety properties will probably be defined in English prose. Even if the properties were defined in the requirements modeling tool, the modeling tool itself would probably lack the analysis capability and would require translation to a theorem prover or model checker. As part of this project, the University of Minnesota has automated the translation of the model from RSML^{-e} to the NuSMV model checker and the PVS theorem prover. Building the translation capability is possible since both the origin and destination languages have a well defined semantics.

The translation of the safety properties into the model checker or theorem prover language was done manually. As a check on the accuracy of the translation, two experienced investigators translated each property independently and then compared results. The translation is straightforward, but requires some knowledge of logic and the syntax of the tool being used. If not already transparent, doing the manual translation also serves as a check on how well the properties (functional or safety) are defined. Ambiguous requirements result in ambiguous properties that cannot be stated in the precise manner demanded by a formal language.

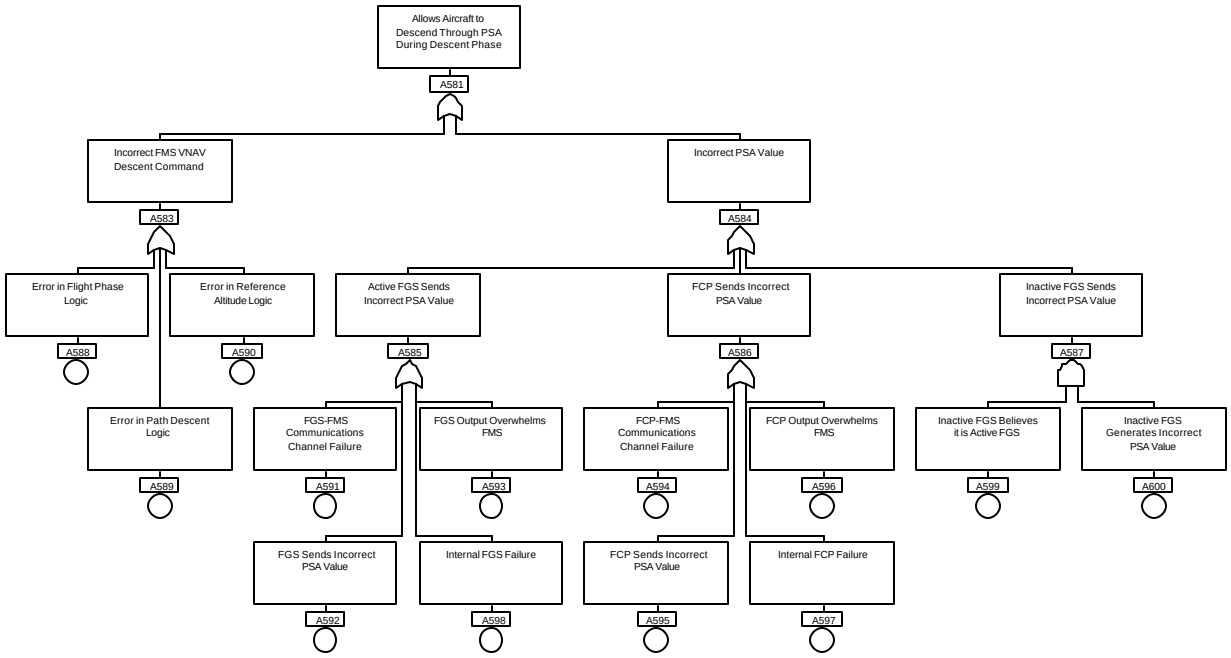

Figure 5. A Top Level View of the Fault Tree Analysis for the Hazard: Allows Aircraft to Descend Through PSA.

Table 5. The Failure Mode Effects and Criticality Analysis for the Hazard: Allows Aircraft to Descend Through PSA.

Failure Mode	Effects	Analysis
Error in Flight Phase Logic	Allows Aircraft to Descend Through PSA	Aircraft Drops Below the Pre-Select Altitude (PSA), Until Detected by Flight Crew During Monitoring of Flight Critical Data.
Error in Path Descent Logic		
Error in Reference Altitude Logic		

Table 6. Example Safety Properties are Associated With "Error in Flight Phase Logic."

Descent Phase	Climb Phase	Cruise Phase
• Flight Phase Shall be Descent After the First Top of Descent Has Been Reached	• Flight Phase Shall be Climb if There is a "Climb" Waypoint Ahead of the Aircraft	• Flight Phase Shall be Cruise if the Aircraft Has Reached the Cruise Altitude
	• Flight Phase Shall be Climb if the Last Top of Climb Has Not Been Reached	

Formal Methods Analysis

As pointed out earlier, the computation of the flight phase directly impacts many safety properties of the FMS VNAV function. The property

> "The aircraft must be in either climb, cruise, or descent phase."

would be translated into NuSMV syntax as:

```
SPEC AG(Is_FMS_VNAV_Valid ->
(Is_Cruise_Phase_Criteria_Met |
Is_Climb_Phase_Criteria_Met |
Is_Descent_Phase_Criteria_Met)
```

"SPEC" refers to the fact that this is a property being specified, "AG" means to do the proof for "All Global" states, "Is_FMS_VNAV_Valid" is the macro used to determine if FMS VNAV is valid and active, and "Is_Climb_Phase_Criteria_Met" is the macro used to determine if the conditions necessary for the aircraft to be in climb phase are valid. This NuSMV statement would be literally translated as:

> "For all global states, if FMS VNAV is valid then either Cruise Phase Criteria is met, or Climb Phase Criteria is met, or Descent Phase Criteria is met."

This highlights one of the advantages of using formal specification. English statements are notoriously ambiguous, but logical statements must be clear and precise. While attempting to convert many of the original English properties into logic we find that we often go back and modify the English statement to remove ambiguity. In this example, a clearer English statement – a statement that more clearly reflects exactly the property being investigated is

> "If FMS VNAV is valid, the aircraft must be in either climb, cruise, or descent phase."

The same property would be stated in PVS as

```
Flight_Phase : LEMMA
   verify(Is_FMS_VNAV_Valid IMPLIES
   Is_Cruise_Phase_Criteria_Met   OR
   Is_Climb_Phase_Criteria_Met   OR
   Is_Descent_Phase_Criteria_Met)
```

Intermediate Results Analysis

As previously mentioned, the safety analysis of the FMS VNAV model is still underway. Nevertheless, some interesting results can be reported. First and foremost is the fact that, without further work, the FMS VNAV model is not suitable for analysis using a model checker such as NuSMV. The reasons for this are not surprising. The FMS VNAV model contains real numbers and trigonometric functions used to calculate the aircraft's trajectory, neither of which can be handled by NuSMV. In contrast, the FGS model, while complex, consisted almost entirely of enumerated and Boolean values, making it ideally suited for verification through model checking.

Of course, the reals can be converted to integers, using scaling to preserve the accuracy already included in the specification. For example, most altitude values are specified in feet, with possible values ranging from -1,300.00 to +65,000.00. Even so, the state space would be quite large. For example, truncating the accuracy of the altitude at two decimal points results in 6,630,000 possible values for all altitude variables. In theory, the combined state space for two altitude variables, (such as the Pre-Select Altitude and the Flight Plan Target Altitude), would be the product of the state space for either variable, or more than 10^{12}. This is the equivalent to the state space of over 40 independent Boolean variables.

The University of Minnesota is augmenting the RSML^{-e} to NuSMV translator with additional abstraction techniques that will hopefully allow us to analyze the FMS VNAV model using NuSMV. Another approach would be to manually replace the computations using reals and trigonometric functions with Boolean inputs. For example, when comparing the aircraft altitude to the PSA the actual desired property is the predicate *"Is the Barometric Altitude less than the Pre-Select Altitude"* rather than the actual numerical difference in the altitudes. Many of the interesting properties of the FMS VNAV model could still be verified using such a simplified model.

In the interim, our work has focused on the use of the PVS theorem prover. Unlike a model checker, a theorem prover is not constrained by the size of the state space. Rather than conducting an exhaustive search of all possible states, a theorem

prover applies the rules of logic to reason about the system, in much the same way a human would. But where human reasoning is flawed and subject to error, a theorem prover ensures that each step of the proof is justified. For real systems, this level of rigor cannot be achieved without automated support..

Using the processing speed and memory of modern computing systems, a prover such as PVS can automate the bookkeeping and simpler proof steps, but much of the reasoning process is still guided by the human user. For example, the user may instruct the theorem prover to try to prove a lemma by induction, only to generate different subgoals that must be proven individually before the proof can be completed. The user must provide guidance on what rules of inference should be used, and in what order they should be applied, for the theorem prover to complete the proof.

Much of our analysis so far has been devoted to discharging the *type correctness conditions* (TCCs) generated by the PVS theorem prover. Since PVS is a strongly typed system based on total functions, it routinely generates type correctness conditions that must be proven in addition to the main properties of interest. Often, proving the type correctness conditions expose assumptions about the system that are unwarranted. For example, since a common source of safety errors is dependence on a system input that is invalid, input variables in RSML^{-e} can be set to UNDEFINED until a valid value is input. The PVS translation of an RSML^{-e} model generates a TCC to ensure that the guard on each transition is completely defined, ensuring that the system is deterministic ,

In the case of the FMS VNAV model the theorem prover generated about 150 TCCs, most of which ensure that values are defined before use. Resolving these has forced us to go back and think much more carefully about the behavior of the model when inputs might be invalid.

Other TCCs were more substantial. For example, one of the TCCs generated required us to prove our earlier safety property that the aircraft must be in either climb, cruise, or descent phase. We anticipate that proofs of the TCCs will be completed in a few weeks so that we can begin reasoning about the other safety properties of the system.

Summary and Conclusions

We have constructed a formal, executable model of a complex, embedded software system – the Vertical Navigation Function of a Flight Management System. In order to identify the properties of the model that are related to safety, we then conducted a software safety analysis using both standard techniques such as Functional Hazard Assessment (FHA), Fault Tree Analysis (FTA) and Failure Modes Effects Analysis (FMEA), and formal methods techniques such as model checkers and theorem provers. In particular, our analysis has focused on using the standard techniques to articulate the properties of the model that relate to safety. We then plan to use the formal methods tools to show that all of the safety properties are mathematically verifiable properties of the model.

To date, we have completed the FHA, and have made a first cut at the FTA and FMEA in order to identify some system level properties for further analysis. Additional abstraction techniques need to be developed to analyze the VNAV model using the NuSMV model checker. Our work with the PVS theorem prover is progressing, but has focused on discharging the type correctness conditions generated by the theorem prover.

We anticipate that over the course of the next several months we will be able to define a translation to the NuSMV model checker that will enable us to use it to analyze the VNAV model. At the same time, we should shortly complete the analysis of the TCC's and will begin to prove various safety properties of the VNAV model using the PVS theorem prover.

References

1. ARINC 702A-1, Advanced Flight Management System Computer, 31 January 2000.
2. Tribble, A. C., D. L. Lempia, and S. P. Miller, "Software Safety Analysis of a Flight Guidance System," *Proc. of 21st Digital Avionics Systems Conference (DASC)*, 2002.
3. Whalen, M. W., *A Formal Semantics for RSML*, Masters Thesis, University of Minnesota, April 2000.
4. Lutz, R. and R. Woodhouse, "Bi-Directional Analysis for Certification of Safety-Critical Software," *Proc. of 1st Int'l Software Assurance Certification Conf*," 1999.
5. ARP 4754, "Certification Considerations for Highly-Integrated or Complex Aircraft Systems," SAE International, November 1996.
6. ARP 4761, "Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment," SAE International, December 1996.
7. RTCA DO-178B, Software Considerations in Airborne Systems and Equipment Certification, 01 DEC 1992.
8. MIL STD 882C, System Safety Program Requirements, 19 January 1993.
9. Herrmann, D. S., *Software Safety and Reliability*, (Los Alamitos, CA: IEEE Computer Society, 1999).
10. *System Safety Analysis Handbook*, 2nd Ed., System Safety Society, July 1997.