MODE CONFUSION ANALYSIS OF A FLIGHT GUIDANCE SYSTEM USING FORMAL METHODS

Anjali Joshi, Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455 USA

Steven P. Miller, Advanced Technology Center, Rockwell Collins Inc.,
Cedar Rapids, IA 52498 USA

Mats P.E. Heimdahl, Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455 USA

Introduction

Advancements in digital avionics systems have accounted for much of the improvement in air safety seen over the last few decades. At the same time, the growing complexity of these systems places greater demands on the flight crew and increases the risk of mode confusion, a phenomenon in which pilots become confused about the status of the system and interact with it incorrectly. To fly commercial flights today, pilots must master several complex, dynamically interacting systems, often operating at different levels of automation. These systems typically have many *different modes of operation*, with different responses to crew actions and the other systems in each mode. Mode confusion occurs when the flight crew believes they are in a mode different than the one they are actually in and consequently make inappropriate requests or responses to the automation. Mode confusion can also occur when the flight crew does not fully understand the behavior of the automation in certain modes, i.e., when the crew have a poor “mental model” of the automation [4], [10], [9]. This same phenomenon is sometimes referred to by the more general name of *automation surprises*.

There is mounting evidence that mode confusion is an important safety concern. Several aircraft accidents and incidents involving mode confusion are listed in [7]. A study conducted by the Massachusetts Institute of Technology found 184 incidents attributed to mode awareness problems in NASA's Aviation Safety Reporting System (ASRS) [21]. In a survey of 1,268 pilots published in 1999 by the Australian Bureau of Air Safety Investigation (BASI), 73% of the respondents indicated that they had inadvertently selected a wrong mode. Moreover, 61% of the respondents agreed that there were still things about the automation that took them by surprise [2]. Of 536 interventions recommended by the Loss of Control Joint Safety Analysis Team (JSAT), they recommended improved training of automated flight systems as their 6th most important intervention and improved feedback from the automation as their 22nd most important intervention [6]. Advisory Circular AC/ACJ 25.1329 on Flight Guidance System Approval identifies “autoflight mode confusion as a significant safety concern” [1].

The basic premise behind detecting mode confusion through analysis of system requirements or design specifications is that *certain design features or patterns are more likely to cause mode confusion than others*. Studies by Sarter and Woods have found evidence for this hypothesis [17], [18], [19], [20], and Leveson, et. al. [9] used their work to identify several categories of problematic design features. In [10], we extended this work with additional examples from the literature and a checklist of specific design features to be searched for during manual reviews. This taxonomy and checklist was used as the basis for an informal review for potential sources of mode confusion in a representative specification of a Flight Guidance System mode logic [11], [12].

This paper describes the use of automated analysis tools, such as *model-checkers* [5] and *theorem provers* [13], to search for potential sources of mode confusion in a representative specification of the mode logic of a Flight Guidance System [11].

This work was supported in part by the NASA Aviation Safety program and the Langley Research Center under Contract NCC-01001.

The Problem Domain

In our studies we have used the Flight Guidance domain. The following sub-sections provide a brief overview of the Flight Guidance System with an emphasis on the mode logic.

The Flight Guidance System

A Flight Guidance System (FGS) is a component of the overall Flight Control System (FCS). It compares the measured state of an aircraft (position, speed, and altitude) to the desired state and generates pitch and roll guidance commands to minimize the difference between the measured and desired state. A simplified overview of an FCS that emphasizes the role of the FGS is shown in Figure 1.

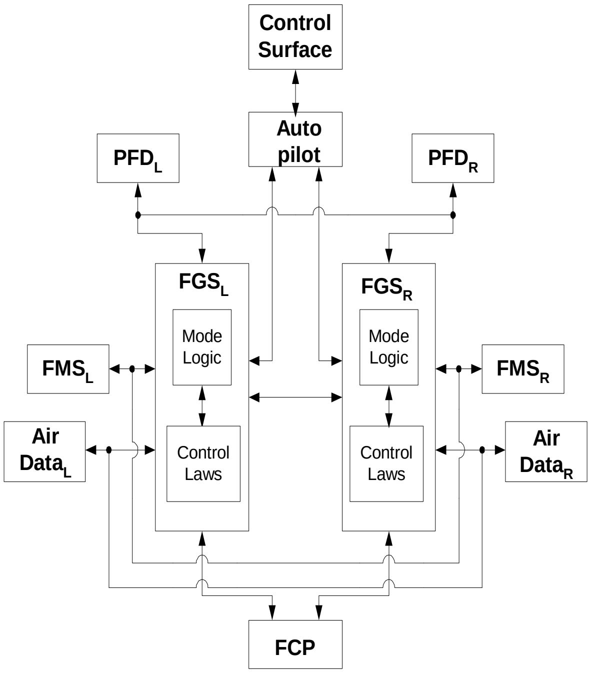

Figure 1: A section of the Avionics System

The flight crew interacts with the FGS primarily through the Flight Control Panel (FCP). The FCP includes switches for turning the Flight Director (FD) on and off, and switches for selecting the different flight modes. The FCP also supplies feedback to the crew, indicating selected modes by lighting lamps on either side of a selected mode's switch.

The mode logic determines which lateral and vertical modes of operation are active and armed at any given time. These in turn determine which flight control laws are active and armed. These are annunciated, or displayed, on the Primary Flight Displays (PFD) along with a graphical depiction of the flight guidance commands generated by the FGS. The Primary Flight Display annunciates essential information about the aircraft, such as airspeed, vertical speed, altitude, the horizon, and heading. The active lateral and vertical modes are annunciated at the top of the display.

The Flight Guidance System Mode Logic

A *mode* is defined by Leveson et. al. as a *mutually exclusive set of system behaviors* [9]. The primary modes of interest in an FGS are the lateral and vertical modes. The lateral modes control the behavior of the aircraft about the longitudinal, or roll, axis, while the vertical modes control the behavior of the aircraft about the vertical, or pitch, axis. In addition, there are a number of auxiliary modes, such as half-bank mode, that control other aspects of the aircraft's behavior.

A mode is said to be *selected* if it has been manually requested by the flight crew or if it has been automatically requested by a subsystem such as the FMS. The simplest modes have only two states, *cleared* and *selecte*d. Some modes can be *armed* to become active when a criterion is met. In such modes, the two states *armed* and *active* are sub-states of the *selected* state. Some modes also distinguish between capturing and tracking of the target reference or navigation source. Once in the active state, such a mode's flight control law first *captures* the target by maneuvering the aircraft to align it with the navigation source or reference. Once correctly aligned, the mode transitions to the *tracking* state that holds the aircraft on the target. Both the *capture* and *track* states are sub-states of the *active* state and the mode's flight control law is active in both states.

The *mode logic* consists of all the available modes and the rules for transitioning between them. Figure 2 provides an overview of the Flight Guidance System modes. Traditionally, aircraft modes are associated with a flight control law that determines the guidance provided to the flight director or autopilot. For example, in Figure 2, there are lateral modes of Roll Hold, Heading Hold, Navigation, Lateral Approach, and Lateral Go Around. These control the guidance about the longitudinal, or roll, axis. Guidance about the vertical, or pitch, axis is controlled by the vertical modes of Pitch, Vertical Speed, Altitude Hold, Altitude Select, Vertical Approach, and Vertical Go Around. Each of these modes are associated with one or more control laws.

In order to provide effective guidance of the aircraft, these modes are tightly synchronized. Constraints enforce sequencing of modes that are dictated by the characteristics of the aircraft and the airspace. The mode logic is responsible for enforcing these constraints.

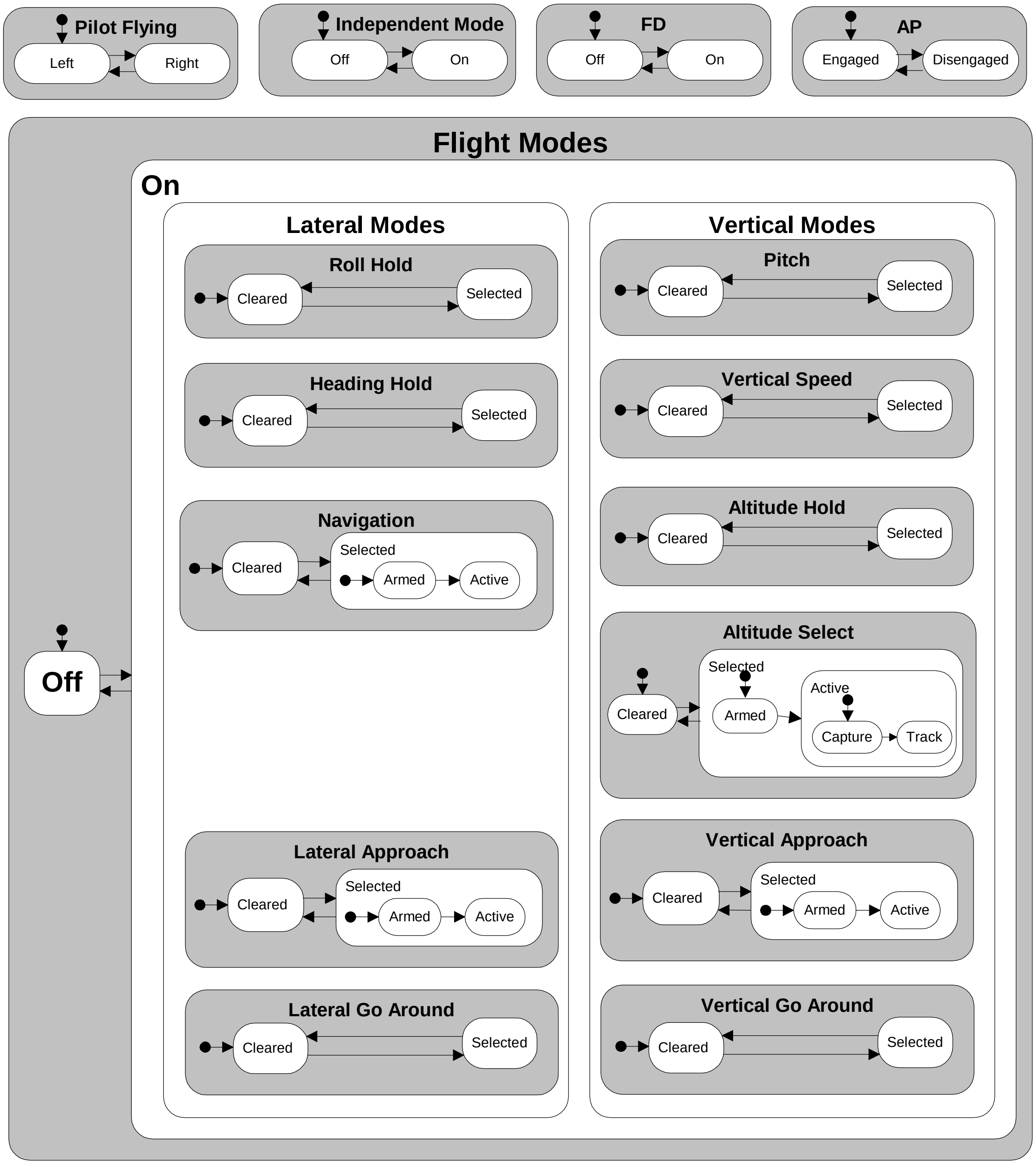

Figure 2: FGS Mode Logic

The FGS mode logic used in this study was a representative example created to explore different specification and verification tools [11] and does not describe an actual fielded product. The model is specified in the formal specification language, RSML^{-e} [22] (Requirements State Machine Language Without Events), developed by the Critical Systems Group at the University of Minnesota. RSML^{-e} is a synchronous language derived from RSML [8] (Requirements State Machine Language), which was developed by Nancy Leveson's group at University of California, Irvine.

Mode Confusion Analysis

The basic premise behind detecting mode confusion through analysis of system requirements or design specifications is that *certain design features or patterns are more likely to cause mode confusion than others*. Studies by Sarter and Woods have found evidence for this hypothesis [17], [18], [19], [20], and based on these studies, Nancy Leveson, et. al. [9] identified categories of design patterns that have historically been a source of mode confusion. In [4] and [10], we elaborated on a few of these categories and sketched out a direction for approaching mode confusion analysis using formal methods. The authors also noted the need for understanding these categories well enough to formalize them mathematically so as to perform formal analysis. In this section, we first discuss the general strategy of our formal approach to mode confusion analysis. We then introduce the formal methods tools that we used for the mode confusion analysis and discuss their applicability to this type of analysis. Finally, we provide a few examples of the formalization of mode confusion properties and performing the analysis with the help of these tools.

Formal Approach to Mode Confusion Analysis

A novel aspect of our approach to mode confusion analysis is that it emphasizes the use of formal verification tools for *exploration* rather than *verification*. In verification, one postulates a property that is believed to be true of a model and uses the verification tool to determine if the property holds or not. When used for exploration, one postulates a property that is probably false of the model, and uses the verification tool to generate insights into why the property is false.

For example, transitions to infrequently seen, or off-normal, modes are often cited as a potential source of mode confusion [9], [18]. Identifying all the scenarios that can cause an off-normal mode to be entered may not be obvious in a complex specification. The following paragraphs cast this example as both a verification and an exploration problem and describe our approach in each case.

When approached as a verification problem, we first enumerate all the *acceptable* ways of entering the mode and then verify that it is indeed an exhaustive list. If we are unable to verify this, i.e., there are other ways by which we can enter the mode, we consider it to be an error in the specification, change the specification to remove the error, and repeat the process until the original property is verified. The verification process can thus be viewed as one of *debugging* the specification to remove *errors* in it.

When approached as an exploration problem, our aim is to search for all the *unknown* ways by which the mode can be entered. To do this, we postulate that there are no ways by which the mode can be entered, then use the formal verification tool to identify the ways in which this is not true, then amend our property to state that there are no scenarios by which the mode can be entered except for these, and repeat the process until the property is true. This list of ways in which the mode can be entered can then be reviewed with pilots and engineers to determine if any of them pose an unacceptable risk of pilot confusion.

Most of the mode confusion analysis falls under the exploration category, as we are typically searching for new scenarios that may be a potential source of mode confusion.

Formal Methods Tools

To analyze the FGS model for Mode confusion, we made use of two analysis tools– the PVS theorem prover and the NuSMV model checker.

The NuSMV Model Checking System

Model checking is a formal verification technique that allows one to check for safety and liveness properties of a model through exhaustive exploration of the state space. This makes verification of properties highly automated and straightforward. However, state space explosion limits the size of the models that can be analyzed.

NuSMV is a re-implementation and extension of SMV [5], the first model checker based on BDDs. NuSMV has been designed to be an open architecture for model checking, which can be reliably used for the verification of industrial designs, as a core for custom verification tools, as a testbed for formal verification techniques, and applied to other research areas [13]. Properties to be verified in NuSMV are specified using either branching time logic (CTL) or linear time logic (LTL) [5].

The PVS Theorem Proving System

In contrast to model checkers, theorem provers apply rules of inference to a specification in order to derive new properties of interest. Theorem provers are generally harder to use than model checkers, requiring considerable technical expertise and understanding of the specification. However, theorem provers are not limited by the size of the state space. Also, some properties that cannot be easily specified using model checkers, such as comparing properties of two arbitrary states that are not temporally related, can be easily specified in the languages of most theorem provers.

PVS is an environment for specification and verification that has been developed at SRI International's Computer Science Laboratory. The system consists of a specification language, a parser, a type checker, and an interactive proof checker. The PVS specification language is based on higher order logic with a richly expressive type system so that a number of semantic errors in specification can be caught during type checking. The PVS prover consists of a powerful collection of inference steps that can be used to reduce a proof goal to simpler subgoals that can be discharged automatically by the primitive proof steps of the prover. The primitive proof steps involve, among other things, the use of arithmetic and equality decision procedure, automatic rewriting, and BDD-based Boolean simplification [13].

Applicability of the Formal Methods Tools to Mode Confusion Analysis

Earlier studies support the applicability of model checkers and theorem provers for mode confusion analysis [4], [15], [16]. To make these formal analysis tools applicable to our domain, the critical systems research group at the University of Minnesota built translators from the RSML^{-e} specification language to the input languages of NuSMV and PVS, allowing us to apply these formal analysis tools to the translated RSML^{-e} specification. We used these tools to analyze the translated FGS specification [11] originally defined in RSML^{-e} for some of the patterns identified in the mode confusion taxonomy [10]. This analysis using NuSMV and PVS is described below.

When given a *verification* type of mode confusion property, the NuSMV model checker can determine either that the property is true or that there is an error in the specification indicated by a counter-example. When checking a false property, NuSMV can find only one counter-example at a time. For performing exploratory analysis with the help of NuSMV, we need to proceed by dismissing each counter-example that we encounter by adding that condition as a known way of violating the property and repeating the process until the property is verified. Sometimes NuSMV will generate several counter-examples due to a single higher-level condition. In this case, the user may need to find a single stronger condition that encompasses all the specific counter-examples. Extracting relevant information from the counter-examples is a difficult job that often requires domain expertise.

Since the FGS mode logic consists almost entirely of enumerated and Boolean types, it is ideally suited to verification through model checking. Not surprisingly, proving a property of the mode logic with PVS usually required more effort than with NuSMV. If our specification had included more integer or real variables, this situation would probably have been quite different. One surprise was that once completed, the PVS proofs often ran faster than the corresponding proof in NuSMV.

One advantage of using PVS for the exploratory analysis is that it lists all the violations of the property as un-dismissed sub-goals of the proof tree. With adequate knowledge of the domain, the user can recognize an unprovable sub-goal at a higher level in the proof tree. This avoids the burden of extracting stronger conditions as in NuSMV. Conversely, the user can proceed down the proof tree by proving parts of a sub-goal to get a more refined condition. An advantage of using PVS is that the user has finer control of the level of detail exposed in the sub-goals.

We found both NuSMV and PVS useful in performing our analysis. Since the mode logic lends itself so well to model checking, the level of automation provided by NuSMV would often save us considerable time and effort. On the other hand, a skilled user of PVS could often gain insights while doing the proof that might be missed using NuSMV. We also encountered some properties that could not be specified in CTL or LTL but could be expressed in PVS. These are discussed later.

Examples of Mode Confusion Patterns and their Analysis

We analyzed our specification for several patterns that might indicate a potential source of mode confusion. In the following sub-sections, we will describe in detail the formalization and analysis of three major patterns. The analysis of the remaining patterns was similar to these and hence is described here only briefly.

Transitions between Normal and Off-normal Modes

Sarter and Woods [18] found that most difficulties related to mode confusion occur during off-normal, time critical situations such as aborted takeoff, disengagement from an automatic mode during approach, or loss of glide slope signal during final approach. In a complex specification, it may be difficult to determine all the scenarios under which an off-normal mode can be entered or exited. Formal analysis tools such as a model checker or theorem prover can be used to identify the conditions under which these modes can be entered or exited.

The general strategy to identify all the ways a system can enter a mode is to first prove that the system will indeed enter the mode for all the known ways, and then try to prove that if none of these conditions exist, the system will not enter the mode. This will identify any unforeseen ways in which the mode can be entered, which are then added to the list of known ways. This process is repeated until all the ways to enter the mode are identified. The process is similar for finding all the ways a mode can be exited.

A committee consisting of engineers, pilots, and specialists in human factors identified Roll, Pitch, and Overspeed modes as off-normal modes. The selection of Roll and Pitch as off-normal modes may be surprising, as these are the default modes of operation. However, in our example, they are never directly selected by the pilot. Instead, they are default modes that become active when the active mode is deselected, either by the pilot or by the system. For this reason, there are many ways to enter these modes.

We performed an analysis of the ways ROLL and Overspeed mode could be entered using both PVS and NuSMV. When analyzing for ROLL mode, we started with nineteen known ways of entering ROLL mode. The first step was to verify that each of these nineteen conditions indeed caused entry into ROLL mode. This step frequently revealed exceptions, forcing the conditions to be strengthened. For example, in PVS, the condition, that ROLL mode is entered when the FLC switch is pressed while the mode annunciations are turned off was originally stated as:

```
ROLL_Selected_If_FLC_Switch_Pressed_While_Modes_Off: LEMMA =
verify((Is_This_Side_Active AND
     NOT PREV(Is_ROLL_Selected) AND
     NOT PREV(Mode_Annunciations_On) AND
     When_FLC_Switch_Pressed_Seen) IMPLIES
        ROLL == L(Selected))
```

The verification of this property lead to an unprovable sequent in PVS (and a counter-example in NuSMV) pointing out that HDG mode rather than ROLL mode would be selected if the pilot or copilot pressed the HDG switch at the same time the FLC switch was pressed. Consequently, the condition was modified to state that ROLL mode would be entered if the modes are off and the FLC switch is pressed *and no other lateral model is requested at the same time*. After all the known ways in which ROLL mode can be entered were proven, these were encoded in a condition,

***ROLL_Selected_Known_Ways*,** and the following theorem was defined to verify that this was indeed an exhaustive list of conditions causing entry into ROLL mode:

```
ROLL_Not_Selected: THEOREM
verify(((NOT (PREV(ROLL)) == L(Selected)) AND
          NOT ROLL_Selected_Known_Ways) IMPLIES
                   NOT (ROLL == L(Selected)))
```

Attempting to prove this theorem lead to unprovable sub-goals, which in turn lead to the discovery of two additional ways of entering ROLL mode that had been overlooked. We finally ended up with an exhaustive list of twenty-one different ways of entering ROLL mode. While none of these were surprising in retrospect, it is notable that individuals familiar with the specification were unable to come up with an exhaustive list without formal analysis.

Unintended Side Effects

Unintended side effects occur when an action intended to have one effect has an additional effect not anticipated by the operators [9], [18]. These can include unanticipated mode changes or changes in the system state. Examples include changing a mode when a new target value is entered, clearing related data when new data are entered, or changing a mode for one controller when a mode in a different controller is changed. Not all side effects are undesirable. Often, they provide convenient shortcuts for the operator, such as turning the system on if it is off when selecting a system function, or clearing out data that is no longer valid. In some cases, the absence of the side effect may itself be a source of mode confusion if the operator anticipates the side effect as a natural consequence of the original operator action.

An effective way of identifying unintended side effects, as well as several other potential sources of mode confusion, is to identify the response of the system to each operator input. Since we already have an RSML^{-e} specification of how the system responds to all its inputs, it is possible to prove that the new partial specification (i.e., the response of the system to a single operator input) is consistent with the full specification. Typically, this proof identifies several exceptions, which are the side effects that are not well understood. We consequently revise the specification of the expected system response until we have captured the actual system response to the operator input. These specification fragments provide valuable insight into how the system behaves in a format that is often obscured by the full model. Reviewing these specification fragments with pilots and experts in human factors can lead to the identification of several potential sources of mode confusion.

The first step in this process is to identify the possible operator inputs and to define the notion of the system state. The next step is to specify how the system state variables change in response to each operator input. For the purpose of this analysis, it is convenient to specify this change in state as a primary effect and several secondary effects.

For example, PVS can be used to detect unintended side effects by defining what the primary effect of each operator input should be, and then trying to prove that for every state, the effect of that input is only the primary effect. The exceptions to this proof are the side effects. The proof statement can then be strengthened to state the effect of the input is the primary effect *and* all of the side effects. In this way, the list of side effects can be enumerated.

To keep the proofs manageable, we chose to do this by defining how each component of the system state responds to each operator input. For example, based on our experience with the FGS system, we postulated that the effect of pressing the FD switch on the first component of the system state, the autopilot (AP), was to not affect the AP state at all. This is stated in PVS as the lemma:

```
FD_Switch_AP_Side_Effects : LEMMA
verify((Is_This_Side_Active AND
          When_FD_Switch_Pressed_Seen)
                    IMPLIES (AP == PREV(AP)))
```

which claims that if this side of the FGS is active and the FD switch is pressed and no higher priority event preempts the pressing of the FD switch, then the state of the AP is just its value in the previous state. This lemma is easily proven in PVS in a few seconds, confirming that pressing the FD switch has no side effects on the AP.

The situation becomes more interesting as we proceed to the display of the lateral and vertical mode annunciations. The mode annunciations are turned on when either the onside or the offside FD are turned on and are turned off when both the onside and offside FDs are turned off. However, we wish to define this not in terms of the onside and offside FD, but in terms of the FD switch. After a few iterations defining the behavior of the RSML^{-e} mode annunciations when the FD switch is pressed and trying to prove that behavior agrees with the model, we proved the following PVS lemma

```
FD_Switch_Modes_Effects : LEMMA
verify((Is_This_Side_Active AND
       When_FD_Switch_Pressed_Seen) IMPLIES
       IF PREV(Modes) == L(Off)
       THEN Modes == L(On)
       ELSE
            IF (PREV(Onside_FD) == L(Off) OR
                Overspeed_Condition OR
                Is_AP_Engaged OR
                Offside_FD == L(On))
           THEN Modes == L(On)
           ELSE Modes == L(Off)
           ENDIF
       ENDIF)
```

This lemma states that if this side of the FGS is active and the FD switch is pressed and no higher priority event preempts the FD switch, then if the mode annunciations were off in the previous state, they will be turned on. If the mode annunciations were on in the previous state, then if the onside FD was on in the previous state, or an Overspeed condition exists, or the AP is engaged, or the offside FD is on, then the mode annunciations will remain on. Otherwise they will be turned off. As shown by this simple proof, pressing the FD switch has rather complicated side effects on the mode annunciations. The remaining question is whether these side effects are appropriate, a question best answered by experts in human factors and pilots. Depending on the audience's comfort with mathematical logic, it may be necessary to present these results in a more intuitive format.

This process is continued for each operator input and each component of system state in order to generate a complete description of how the system state responds to each operator input. As shown by these examples, the response of the system to each input can be surprisingly complex. Even for something as simple as pressing the FD switch, the authors of the full FGS specification found it difficult to predict how each component of the system state would change without using formal analysis.

Indirect Mode Changes / Hidden Modes

Indirect mode changes occur when the automation changes mode without an explicit instruction by the operator. Indirect mode changes are a natural consequence of delegating tasks to the automation and cannot be eliminated without losing the benefits of automation. At the same time, indirect mode changes are obvious sources of mode confusion. Most indirect mode changes can be easily identified through inspection simply by reviewing how the *non-operator* inputs are used.

A mode is operationally relevant if the system responds differently to operator or sensor inputs while in that mode. Hidden modes are operationally relevant modes that are *not annunciated*. The entry of a hidden mode differs from an indirect mode change in that the operator receives no indication of the change in behavior. While this is technically the same thing as an *un-annunciated indirect mode change*, hidden modes seem to be so frequently involved in incidents of mode confusion that we have chosen to identify them as a separate category.

Finding hidden modes in a large, complex system appears to be an intrinsically hard problem. These modes are hidden precisely because they are buried in a sea of complex logic. What is needed is a systematic way to search a large, complex specification for hidden modes of behavior. Fortunately, the same approach that was used to search for unintended side effects can be used to detect hidden modes. The basic idea is to determine the change in the system state for each operator input, i.e., to differentiate the specification with respect to each operator input. This breaks a large, complex specification down into many small fragments that can be easily reviewed. Hidden modes are identified by asking whether the pilot has sufficient information to predict how each new state value will be changed in response to the operator input. Situations where the pilot does not have sufficient information indicate the presence of a hidden mode.

For example, consider the FD_Switch_Mode_ Effects lemma described in the discussion of unintended side effects. This lemma states that the new value of the mode annunciations are determined by values of the previous system state (i.e., PREV(Modes) and PREV(Onside_FD)), and the current inputs (i.e., Overspeed_Condition, Is_AP_Engaged, and Offside_FD). The key question is whether these values are available to the pilot. The status of the mode annunciations and the onside FD cues in the previous step are available on the Primary Flight Display (PFD).[1] The current status of the autopilot (AP) is also displayed on the PFD. An overspeed condition is annunciated by both aural and visual alarms. However, the status of the offside FD is visible to the pilot only by looking at the copilot's display (and vice-versa). If the offside FD is not adequately visible to the pilot, there is no way for the pilot to predict whether the mode annunciations will turn off when pressing the FD switch. This is a hidden mode.

While this is probably not a significant source of mode confusion, it does illustrate a systematic process by which hidden modes can be detected. In effect, the exact same analysis done to identify unintended side effects can also be used to detect hidden modes.

Operator Authority Limits / Ignored Operator Commands

Operator authority limits restrict the control of the operators in order to prevent the system from entering a hazardous state. An obvious danger of operator authority limits is that they may prohibit maneuvers that are required in extreme situations. Another danger occurs when pilots are not aware that these limits are in effect. Mode confusion can also arise if the crew expects operator authority limits to be in effect when they are not. This is especially true if the operator authority limits are present in most but not all modes.

The operator authority limits of the most interest are those related to continuous operator inputs, such as moving the throttles or the yoke. Unfortunately, the operator inputs in the FGS model are all discreet inputs, such as pressing a mode switch or engaging the autopilot. *Ignored Operator commands* are a special case of operator authority limits directly relevant to such discreet inputs.

Ignored operator commands commonly occur when a system ignores an operator command, not to prevent entry of a hazardous state, but because the command is meaningless in the current mode. Simply ignoring an operator command may or may not be appropriate. In some situations, the operator should receive an indication of why the command has been rejected so that they can correct their mental model. Also, without an indication that a command has been rejected, the operator may think the command has been accepted, leading to further confusion. In other situations, it may be clear both that the request has been ignored and why and providing the operator with more feedback is unnecessary.

To detect all ignored operator commands so that each can be reviewed, one attempts to prove that every operator input causes some visible change in the system state. The exceptions to this proof are ignored operator commands. This analysis closely follows the *unintended side effects* and *hidden modes* and is not described here further.

Interface Interpretation Errors

Interface interpretation errors are those in which the system interprets user-entered values differently than intended or maps multiple conditions onto the same output depending on the controller's current mode [9], [18]. If the users misunderstand what mode the system is in, or have a poor mental model and do not understand how the current mode affects the inputs or the outputs, they are likely to enter the wrong values or become confused about what the system is telling them.

Potential input interpretation errors can be detected by looking at each operator input and determining if its effect on the system state is dependent on the current state of the system or other inputs. The analysis for this is very similar to that described to detect *unintended side effects* and *hidden modes*.

Lack of Appropriate Feedback

An important source of mode confusion is lack of appropriate feedback about the system [9], [18],

[1] We assume the pilot will recall the previous displayed values given the standard 1/20th of a second refresh rate.

referred to by Billings as opacity [3]. Feedback about the *current* system state is obviously essential, but operators also need feedback about *pending* changes to allow them to anticipate system behavior [9], [18].

An obvious source of mode confusion occurs when the operator is not provided with sufficient information to distinguish between two different configurations of system modes. To detect this, we need to define what it means for two arbitrary states to have the same modes and what it means for two states to have the same mode annunciations. As it is not possible to express properties about two *arbitrary* states in CTL, we found PVS better suited for searching for this potential source of mode confusion than NuSMV.

Reviewing the list of FGS state variables, we determined that two states s1 and s2 have the same lateral modes if the following PVS predicate holds:

```
same_lateral_modes(s1,s2): boolean =
        ROLL(s1) = ROLL(s2) AND
        HDG(s1) = HDG(s2) AND
        NAV(s1) = NAV(s2) AND
        NAV_Selected(s1) = NAV_Selected(s2) AND
        LAPPR(s1) = LAPPR(s2) AND
        LAPPR_Selected(s1) =
        LAPPR_Selected(s2) AND
        LGA(s1) = LGA(s2)
```

Similarly, states s1 and s2 have the same vertical modes if the following PVS predicate holds:

```
same_vertical_modes(s1,s2): boolean =
        PITCH(s1) = PITCH(s2) AND
        VS(s1) = VS(s2) AND
        FLC(s1) = FLC(s2) AND
        ALT(s1) = ALT(s2) AND
        ALTSEL(s1) = ALTSEL(s2) AND
        ALTSEL_Selected(s1) =
        ALTSEL_Selected(s2) AND
        ALTSEL_Active(s1) =
        ALTSEL_Active(s2) AND
        VAPPR(s1) = VAPPR(s2) AND
        VAPPR_Selected(s1) =
        VAPPR_Selected(s2) AND
        VGA(s1) = VGA(s2)
```

Finally, we define two states s1 and s2 to have the same modes if the following PVS predicate holds:

```
same_modes(s1,s2): boolean =
        AP(s1) = AP(s2) AND
        Pilot_Flying(s1) = Pilot_Flying(s2) AND
        Onside_FD(s1) = Onside_FD(s2) AND
        Overspeed(s1) = Overspeed(s2) AND
        Modes(s1) = Modes(s2) AND
        same_lateral_modes(s1,s2) AND
        same_vertical_modes(s1,s2)
```

In a similar manner, we define what it means for two states to have the same annunciations. To detect a situation where two different mode configurations have the same annunciations, we attempt to prove the following theorem:

```
Distinct_Mode_Annunciations : THEOREM
     (NOT same_modes(s1,s2)) AND
     (Is_This_Side_Active(s1) = &(TRUE)) AND
     (Is_This_Side_Active(s2) = &(TRUE)) IMPLIES
        NOT same_annunciations(s1,s2)
```

This theorem claims that if states s1 and s2 do not have the same modes, their mode annunciations must also be different. To simplify the proof, we consider only the case where this side of the FGS is active for each state. Since an inactive FGS sets its modes to those of the active side, this is sufficient to show that distinct mode configurations will have distinct mode annunciations.

While this theorem can be proven directly in PVS for the FGS model, it is simpler and more efficient to first prove a number of smaller lemmas related to each mode and then use these lemmas in the proof of the theorem. For example, to ensure that two states s1 and s2 with different Flight Director modes have different pilot displays, we prove the following lemma:

```
Distinct_Onside_FD_Annunciations : LEMMA
     NOT (Onside_FD(s1) = Onside_FD(s2)) IMPLIES
     NOT(Onside_FD_On(s1) = Onside_FD_On(s2)) OR
     NOT (Is_This_Side_Active(s1)=
          Is_This_Side_Active(s2))
```

stating that if the modes of the onside Flight Director (Onside_FD) are different in the two states, than either the indicators sent to the PFD (Onside_FD_On) must be different or the FGS cannot be active in both states.

Similarly, to show that two states with different ROLL modes have different displays, we prove the lemma:

```
Distinct_ROLL_Annunciations : LEMMA
    ((NOT (ROLL(s1) = ROLL(s2))) AND
    (Is_This_Side_Active(s1) = &(TRUE)) AND
    (Is_This_Side_Active(s2) = &(TRUE)))
    IMPLIES
       NOT (Is_ROLL_Active(s1) =
            Is_ROLL_Active(s2)) OR
       NOT (Mode_Annunciations_On(s1) =
            Mode_Annunciations_On(s2))
```

Here, it is necessary to verify that either the value sent to the PFD if ROLL mode is active (Is_ROLL_Active) or the value sent to the PFD to turn the mode annunciations on (Mode_Annunciations_On) are different in the two states. This is because Is_ROLL_Active is false if ROLL mode is cleared (i.e., not selected) or if the mode annunciations are turned off.

The lemmas proven for each mode are then used to prove the overall theorem and show that any two states with different mode configurations must have different displays to the pilot in our model.

Conclusion

Mode awareness has been identified by several researchers, the FAA, and the aviation industry as an important safety concern. Despite this level of interest, finding ways to detect and mitigate potential sources of mode confusion remains as much an art as a science. The basic premise behind our approach is that certain design features or patterns are more likely to cause mode confusion than others. This paper describes an approach in which requirements and design documents can be analyzed for potential sources of mode confusion through the use of automated analysis tools, such as model-checkers and theorem provers.

In particular, we have shown how formal analysis methods can be used to identify several potential sources of mode confusion in a system specification. Even though our analysis was only partial, we were able to find hidden modes, ignored operator inputs, unintended side effects, lack of feedback regarding current modes, and surprises in how off-normal modes can be entered and exited in our example specification. If this analysis was applied to an actual system, these potential sources of mode confusion could be taken back to the engineers, pilots, and experts in human factors for closer review.

References

[1] Anonymous, February 1, 2002, Flight Guidance System Approval, Joint Advisory Circular AC/ACJ 25.1329.

[2] Austrailian Bureau of Air Safety Investigation (BASI), June-August 1999, Advanced-technology Aircraft Safety Survey Report, Flight Safety Digest, pg. 137-216,.

[3] Billings, Charles E., 1997, Aviation Automation: the Search for a Human Centered Approach, Lawrence Erlbaum Associates, Inc., Mahwah, NJ.

[4] Butler, Ricky W., Steven P. Miller, James N. Potts, Victor A. Carreno, October 1998, A Formal Methods Approach to the Analysis of Mode Confusion, in Proceedings of the 17th AIAA/IEEE Digital Avionics Systems Conference, Bellevue, WA.

[5] Clarke, Edmund M., Orna Grumberg, and Doron A. Peled, 2001, Model Checking, The MIT Press, Cambridge, Massachusetts.

[6] Commercial Aviation Safety Team, December 15, 2000, Final Report of the Loss of Control JSAT: Results and Analysis, Paul Russell and Jay Pardee, Co-Chairs,.

[7] Hughes, Dan, Michael Dornheim, January 30-February 6, 1995, Automated Cockpits Special Report, Parts I & II, Aviation Week & Space Technology.

[8] Leveson, Nancy, Mats Heimdahl, Holly Hildreth, Jon Reese, September 1994, Requirements Specifications for Process-Control Systems, IEEE Transactions on Software Engineering, 20(9):684-707.

[9] Leveson, Nancy, L. Denise Pinnel, Sean David Sandys, Shuichi Koga, Jon Damon Reese, March 1997, Analyzing Software Specifications for Mode Confusion Potential, in Proceedings of a Workshop on Human Error and System Development, C.W. Johnson, Editor, pg. 132-146, Glasgow, Scotland.

[10] Miller, Steven P., February 2001, Taxonomy of Mode Confusion Sources Final Report, NASA Contractor Report.

[11] Miller, Steven P., Alan C. Tribble, Timothy M. Carlson, Eric J. Danielson, November 2001, Flight Guidance System Requirements Specification Final Report, NASA Contractor Report.

[12] Miller, Steven P., February 2002, FGS Model Visualization: Final Report, NASA Contractor Report.

[13] NuSMV: A New Symbolic Model Checking, available at http://nusmv.irst.itc.it/.

[14] PVS: Prototype Verification System, available at http://www.csl.sri.com/projects/pvs/.

[15] Rushby, John, June 1999, Using Model Checking to Help Discover Mode Confusions and Other Automation Surprises, in the Proceedings of the 3 rd Workshop on Human Error, Safety, and System Development (HESSD'99), Liege, Belgium.

[16] Rushby, John, Judith Crow, Everett Palmer, October 1999, An Automated Method to Detect Potential Mode Confusion, in the Proceedings of the 18 th AIAA/IEEE Digital Avionics Systems Conference (DASC), St. Louis, MO.

[17] Sarter, Nadine D., David D. Woods, April 1994, Decomposing Automation: Autonomy, Authority, Observability and Perceived Animacy, First Automation Technology and Human Performance Conference.

[18] Sarter, Nadine D., David D. Woods, 1995, How in the World Did I Ever Get Into That Mode?, Mode Error and Awareness in Supervisory Control, Human Factors, 37(1), pg. 5-19.

[19] Sarter, Nadine D., David D. Woods, February 1995, Strong, Silent, and Out-of-the-Loop, CSEL Report 95-TR-01, Ohio State University.

[20] Sarter, Nadine D., and David D. Woods, C. E. Billings, 1997, Automaton Surprises, in Handbook of Human Factors/Ergonomics, 2 nd Edition, G. Salvendy (editor), Wiley, New York.

[21] Vakil, Sanjay S. and John R. Hansman,.Jr., 2002, "Approaches to Mitigating Complexity-Driven Issues in Commercial Autoflight Systems," *Reliability Engineering and System Safety*, Vol. 75, pp. 133-145,.

[22] Whalen, Michael W., May 2000, A Formal Semantics of RSML^{-e}, Master's Thesis, University of Minnesota.