An Introduction to Requirements Capture Using PVS: Specification of a Simple Autopilot

Ricky W. Butler
Langley Research Center, Hampton, Virginia

May 1996
Abstract

This paper presents an introduction to capturing software requirements in the PVS formal language. The object of study is a simplified digital autopilot that was motivated in part by the mode control panel of NASA Langley’s Boeing 737 research aircraft. The paper first presents the requirements for this autopilot in English and then steps the reader through a translation of these requirements into formal mathematics. Along the way deficiencies in the English specification are noted and repaired. Once completed, the formal PVS requirement is analyzed using the PVS theorem prover and shown to maintain an invariant over its state space.

KEYWORDS: Formal Methods, Software Requirements, Verification, Autopilot
1 Introduction

In this paper, the process of translating requirements into a formal language will be explored. The chosen target specification language is SRI International’s PVS Language[10], however, the modeling techniques used in this paper can be used with other formal specification languages that are based upon higher-order logic. The exposition is centered around the analysis of a simple autopilot that is somewhat related to an early Boeing-737 autopilot. The requirements for this autopilot are first given in English and then translated in a step-by-step manner into PVS. This “requirements capture” process is discussed in detail.

All of the PVS language features that are used are explained thoroughly to make the paper self-contained. More detailed information about PVS can be obtained from [9, 8, 14]. Also, several tutorial introductions to PVS are available [6, 3, 16, 4, 11, 15].

2 Example Application

The techniques of formal specification and verification of an avionics subsystem will be demonstrated on a very simplified example of a mode-control panel. An informal, English-language specification of the mode-control panel representative of what software developers typically encounter in practice will be presented. The process of clarifying and formalizing the English specification into a formal specification, often referred to as requirements capture, will then be illustrated.

2.1 English Specification of the Example System

This section presents the informal, English-language specification of the example system. The English specification is annotated with section numbers to facilitate references back to the specification in later sections.

1. The mode-control panel contains four buttons for selecting modes and three displays for dialing in or displaying values, as shown in Figure 1. The system supports the following four modes:

- attitude control wheel steering (att_cws)
- flight path angle selected (fpa_sel)
- altitude engage (alt_eng)
- calibrated air speed (cas_eng)

Only one of the first three modes can be engaged at any time. However, the cas_eng mode can be engaged at the same time as any of the other modes. The pilot engages a mode by pressing the corresponding button on the panel. One of the three modes, att_cws, fpa_sel, or alt_eng, should be engaged at all times. Engaging any of the first three modes will automatically cause the other two to be disengaged since only one of these three modes can be engaged at a time.

2. There are three displays on the panel: and altitude [ALT], flight path angle [FPA], and calibrated air speed [CAS]. The displays usually show the current values for the altitude, flight path angle, and air speed of the aircraft. However, the pilot can enter a new value into a display by dialing in the value using the knob next to the display. This is the target or “pre-selected” value that the pilot wishes the aircraft to attain. For example, if the pilot wishes to climb to 25,000 feet, he will dial 25,000 into the altitude display window and then press the alt_eng button to engage
the altitude mode. Once the target value is achieved or the mode is disengaged, the display reverts to showing the "current" value.

3. If the pilot dials in an altitude that is more than 1,200 feet above the current altitude and then presses the alt_eng button, the altitude mode will not directly engage. Instead, the altitude engage mode will change to "armed" and the flight-path angle select mode is engaged. The pilot must then dial in a flight-path angle for the flight-control system to follow until the aircraft attains the desired altitude. The flight-path angle select mode will remain engaged until the aircraft is within 1,200 feet of the desired altitude, then the altitude engage mode is automatically engaged.

4. The calibrated air speed and the flight-path angle values need not be pre-selected before the corresponding modes are engaged—the current values displayed will be used. The pilot can dial-in a different target value after the mode is engaged. However, the altitude must be pre-selected before the altitude engage button is pressed. Otherwise, the command is ignored.

5. The calibrated air speed and flight-path angle buttons toggle on and off every time they are pressed. For example, if the calibrated air speed button is pressed while the system is already in calibrated air speed mode, that mode will be disengaged. However, if the attitude control wheel steering button is pressed while the attitude control wheel steering mode is already engaged, the button is ignored. Likewise, pressing the altitude engage button while the system is already in altitude engage mode has no effect.

Because of space limitations, only the mode-control panel interface itself will be modeled in this example. The specification will only include a simple set of commands the pilot can enter plus the
functionality needed to support mode switching and displays. The actual commands that would
be transmitted to the flight-control computer to maintain modes, etc., are not modeled.

2.2 Formally Specifying the Example System

In this section, we will describe the process of formally specifying the mode-control panel described
in English in the previous section. Quotes cited from the English specification will be annotated
with the corresponding section number in parentheses. The goal is to completely describe the
system requirements in a mathematical notation, yet not overly constrain the implementation.

This system collects inputs from the pilot and maintains the set of modes that are currently
active. Thus, it is appropriate to model the system as a state machine. The state of the machine
will include the set of modes that are active, and the pilot inputs will be modeled as events that
transition the system from the current state ($S_c$) to a new state ($S_n$):

$$S_n = \text{nextstate}(S_c, \text{ev}).$$

Figure 2: State transition function

The arrow represents a transition function, $\text{nextstate}$, which is a function of the current state
and an event, say $\text{ev}$:

$$S_n = \text{nextstate}(S_c, \text{ev}).$$

The goal of the formal specification process is to provide an unambiguous elaboration of the
$\text{nextstate}$ function. This definition must be complete; i.e., it must provide a next state for all
possible events and all possible states. Thus, the first step is to elaborate all possible events and
the content of the state of the machine. The system will be specified using the PVS specification
language [12, 16, 14, 8].

2.3 Events

The pilot interacts with the mode-control panel by pressing the mode buttons and by dialing pre-
selected values into the display. The pilot actions of pressing one of the four buttons will be named
as follows: $\text{press\_att\_cws}$, $\text{press\_cas\_eng}$, $\text{press\_alt\_eng}$, and $\text{press\_fpa\_sel}$. The actions of
dialing a value into a display will be named as follows: $\text{input\_alt}$, $\text{input\_fpa}$, and $\text{input\_cas}$. The behavior of the mode-control panel also depends upon the following inputs it receives from
sensors: $\text{alt\_reached}$, $\text{fpa\_reached}$, and $\text{alt\_gets\_near}$. In PVS, the set of events are specified
as follows:

$$\text{events: TYPE} = \{ \text{press\_att\_cws, press\_cas\_eng, press\_alt\_eng, press\_fpa\_sel, input\_alt, input\_fpa, input\_cas, alt\_reached, alt\_gets\_near, fpa\_reached} \}$$
2.4 State Description

The state of a system is a collection of attributes that represents the system's operation. In the example system, the set of active modes would certainly be a part of the system state. Also, the values in the displays that show altitude, flight-path-angle and air-speed and the readings from the airplane sensors would be included in the state. It is important to find a set of attributes that enable a full description of the system behavior and an efficient method of representing these attributes.

One possible approach to describing the system state for the example is to use a set to delineate which modes are active. For example, \{ att_cws, cas_eng \} would represent the state of the system where both att_cws and cas_eng are engaged but alt_eng and fpa_sel are not engaged. The alt_eng mode has the additional capability of being armed. Thus, a better approach to describing the example system state is to associate with each mode one of the following values: off, armed, or engaged. In PVS, a type or domain can be defined with these values:

\[
\text{mode_status: TYPE = \{off, armed, engaged\}}
\]

The state descriptor is as follows:

\[
[\# \% \text{RECORD} \\
\text{att_cws: mode_status,} \\
\text{cas_eng: mode_status,} \\
\text{fpa_sel: mode_status,} \\
\text{alt_eng: mode_status,} \\
\#] \% \text{END}
\]

For example, the record

\[
[\text{att_cws}=\text{engaged}, \text{cas_eng}=\text{engaged}, \text{fpa_sel}=\text{off}, \text{alt_eng}=\text{off}]
\]

would indicate a system where both att_cws and cas_eng are engaged while fpa_sel and alt_eng are off. However, there is still a problem with the state descriptor. In the example system only alt_eng can be armed. Thus, more restrictive domains are needed for the modes other than alt_eng. This can be accomplished by defining the following sub-type of mode_status:

\[
\text{off_eng: TYPE = \{mode: mode_status | mode = off OR mode = engaged \}}
\]

Thus, the type off_eng has two values: off and engaged. The state descriptor is thus modified to become:

\[
[\# \% \text{RECORD} \\
\text{att_cws: off_eng,} \\
\text{cas_eng: off_eng,} \\
\text{fpa_sel: off_eng,} \\
\text{alt_eng: mode_status} \\
\#] \% \text{END}
\]

The mode panel also maintains the state of the displays. To simplify the example, the actual values in the displays will not be represented. Instead, the state descriptor will only keep track of whether the value is a pre-selected value or the actual value read from a sensor. Thus, the following type is added to the formalism:
disp_status: TYPE = {pre_selected, current }

and three additional fields are added to the state descriptor:

alt_disp: disp_status
fpa_disp: disp_status
cas_disp: disp_status

The behavior of the mode-control panel does not depend upon the actual value of "altitude" but rather on the relationship between the actual value and the pre-selected value in the display. The following "values" of altitude are sufficient to model this behavior:

altitude_vals: TYPE = { away, near_pre_selected, at_pre_selected },

and the following is added to the state descriptor:

altitude: altitude_vals.

The final state descriptor is:

states: TYPE=[# % RECORD
  att_cws: off_eng,
cas_eng: off_eng,
fpa_sel: off_eng,
aleng: mode_status,
alt_eng: mode_status,
altdisp: disp_status,
fpa_disp: disp_status,
cas_disp: disp_status,
alitude: altitude_vals
#] END

2.5 Formal Specification of nextstate Function

Once the state descriptor is defined, the next step is to define a function to describe the system’s operation in terms of state transitions. The nextstate function can be defined in terms of ten sub-functions, one for each event, as follows:

event: VAR events
st: VAR states
nextstate(st,event): states =
  CASES event OF
    press_att_cws: tran_att_cws(st),
presalt eng: tran_alt_eng(st),
pres_fpa_sel: tran_fpa_sel(st),
pres_cas_eng: tran_cas_eng(st),
input_alt : tran_input_alt(st),
input_fpa : tran_input_fpa(st),
input_cas : tran_input_cas(st),
alt_reached : tran_alt_reached(st),
fpa_reached : tran_fpa_reached(st),
al gets_near: tran_alt_gets_near(st)
ENDCASES
The CASES statement is equivalent to an IF-THEN-ELSEIF-ELSE construct. For example, if the event is press_fpa_sel then 
\[
\text{nextstate}(\text{st}, \text{event}) = \text{tran_fpa_sel}(\text{st}).
\]
The next step is to define each of these subfunctions.

2.5.1 Specifying the att_cws Mode

The tran_att_cws function describes what happens to the system when the pilot presses the att_cws button. This must be specified in a manner that covers all possible states of the system. According to the English specification, the action of pressing this button attempts to engage this mode if it is off. Changing the att_cws field to engaged is specified as follows:

\[
\text{st WITH} \{ \text{att_cws := engaged} \}.
\]

The WITH statement is used to alter a record in PVS. This expression produces a new record that is identical to the original record st in every field except att_cws. Of course, this is not all that happens in the system. The English specification also states that, “Only one of the [att_cws, fpa_sel, or cas_eng] modes can be engaged at any time” (1). Thus, the other modes must become something other than engaged. It is assumed that this means they are turned off. This would be indicated as:

\[
\text{st WITH} \{ \text{att_cws := engaged, fpa_sel := off, alt_eng := off} \}.
\]

The English specification also states that when a mode is disengaged, “…the display reverts to showing the ‘current’ value” (2):

\[
\text{st WITH} \{ \text{att_cws := engaged, fpa_sel := off, alt_eng := off, alt_disp := current, fpa_disp := current} \}.
\]

Also, the English specification says that, “…if the attitude control wheel steering button is pressed while the attitude control wheel steering mode is already engaged, the button is ignored” (5). Thus, the full definition is:

\[
\text{tran_att_cws(st): states} = \\
\text{IF att_cws(st) = off THEN} \\
\text{st WITH} \{ \text{att_cws := engaged, fpa_sel := off, alt_eng := off, alt_disp := current, fpa_disp := current} \} \\
\text{ELSE st \% IGNORE: state is not altered at all} \\
\text{ENDIF}
\]

The formal specification has elaborated exactly what the new state will be. The English specification does not address what happens to the ALT display or the FPA display when they are pre-selected and the corresponding mode is not currently engaged. The English specification only covers the situation where the corresponding mode is engaged. However, the formal specification does explicitly indicate what will happen: the displays are returned to showing the current value. The process of developing a complete mathematical specification often leads to the discovery of
ambiguities in the English specification. In a real system design process, the developer would communicate with the system designers to clarify what behaviors were intended for these cases. The clarifications from the system designers would then be incorporated into the formal specification and also clarified in the accompanying English system description. If, for example, the system designers responded that the pre-selected displays should only remain the same if the corresponding mode is \textit{off}, the following specification would be appropriate:

\begin{verbatim}
tran_att_cws(st): states = 
  IF att_cws(st) = off THEN
    st WITH [a%%_cws
    alt_eng
    alt_disp
    fpa_disp
  ]
  ELSE st _ IGNORE: state is not altered at all
ENDIF
\end{verbatim}

We realize that this situation will arise in several other situations as well. In particular, whenever a mode (other than \textit{cas_eng}) becomes \textit{engaged}, should any other “pre-selected” displays be changed to \textit{current} or should they remain \textit{pre-selected}? We decide to consult the system designers. They agreed that the displays should be returned to \textit{current} and suggested that the following be added to the English specification

\begin{quote}
6. Whenever a mode other than \textit{cas_eng} is engaged, all other pre-selected displays should be returned to \textit{current}.
\end{quote}

We have labeled this as requirement 6.

2.5.2 Specifying the \textit{cas_eng} Mode

The \textit{tran_cas_eng} function describes what happens to the system when the pilot presses the \textit{cas_eng} button. This is the easiest mode to specify because its behavior is completely independent of the other modes. Pressing the \textit{cas_eng} button merely toggles this mode on and off. The complete function is:

\begin{verbatim}
tran_cas_eng(st): states =
  IF cas_eng(st) = off THEN
    st WITH [cas_eng := engaged]
  ELSE
    st WITH [cas_eng := off, disp_cas := current]
  ENDIF
\end{verbatim}

This specification states that if the \textit{cas_eng} mode is currently \textit{off}, pressing the button will engage the mode. If the mode is \textit{engaged}, pressing the button will turn the mode \textit{off}. Thus, the button acts like a switch. When \textit{cas_eng} is disengaged the corresponding display, \textit{disp_cas}, is returned to \textit{current}.
2.5.3 Specifying the fpa_sel Mode

The tran_fpa_sel function describes the behavior of the system when the fpa_sel button is pressed. The English specification states that this mode “need not be pre-selected (4)”. Thus, whether the FPA display is pre-selected or not the outcome is the same:

\[
\text{IF } \text{fpa}\_\text{sel}(st) = \text{off} \text{ THEN } \\
\text{st WITH } [\text{fpa}\_\text{sel} := \text{engaged}, \text{att}\_\text{cws} := \text{off}, \\
\text{alt}\_\text{eng} := \text{off}, \text{alt}\_\text{disp} := \text{current}] \\
\]

Note that the fpa_sel mode engaged and the att_cws and alt_eng are turned off as well. This was included because the English specification states that, “Engaging any of the first three modes will automatically cause the other two to be disengaged” (1). Also note that this specification indicates that alt Disp is set to current because the alt_eng mode has been disengaged. This was done because the English specification states that, “Once the target value is achieved or the mode is disengaged, the display reverts to showing the ‘current’ value” (2). If the alt_eng mode is not currently active, the WITH update does not actually change the value, but merely updates that attribute to the value it already holds.

Since PVS requires that functions be completely defined, we must also cover the case where fpa_sel is already engaged. We consult the English specification and find The calibrated air speed and flight-path angle buttons toggle on and off every time they are pressed. (5). We interpret this to mean that if the fpa_sel button is pressed while it is currently engaged, the mode will be turned off. This is specified as follows:

\[
\text{st WITH } [\text{fpa}\_\text{sel} := \text{off}, \text{fpa}\_\text{disp} := \text{current}] \\
\]

Because the mode is disengaged, the corresponding display is returned to current. We realize that we also must cover the situation where the alt_eng mode is armed and the fpa_sel is engaged. In fact, section (3) of the English specification indicates that this will occur when one presses the alt_eng button and the airplane is far away from the pre-selected altitude. However, Section (3) does not tell is whether the disengagement of fpa_sel will also disengage the armed alt_eng mode. We decide to consult the system designers. They inform us that pressing the fpa_sel button should turn off both the fpa_sel and alt_eng mode in this situation. Thus, we modify the state update statement as follows:

\[
\text{st WITH } [\text{fpa}\_\text{sel} := \text{off}, \text{alt}\_\text{eng} := \text{off}, \\
\text{fpa}\_\text{disp} := \text{current}, \text{alt}\_\text{disp} := \text{current}] \\
\]

The complete specification is thus:

\[
\text{IF } \text{fpa}\_\text{sel}(st) = \text{off} \text{ THEN } \\
\text{st WITH } [\text{fpa}\_\text{sel} := \text{engaged}, \text{att}\_\text{cws} := \text{off}, \\
\text{alt}\_\text{eng} := \text{off}, \text{alt}\_\text{disp} := \text{current}] \\
\text{ELSE } \\
\text{st WITH } [\text{fpa}\_\text{sel} := \text{off}, \text{fpa}\_\text{disp} := \text{current}, \\
\text{alt}\_\text{eng} := \text{off}, \text{alt}\_\text{disp} := \text{current}] \\
\text{ENDIF} \\
\]

The perspicacious reader may have noticed that there is a mistake in this formal specification. The rest of us will discover it when a proof is attempted using a theorem prover in the later section entitled, “Formal Verification of the Example System.”
2.5.4 Specifying the alt_eng Mode

The alt_eng mode is used to capture a specified altitude and hold it. This is clearly the most difficult of the four to specify since it has a complicated interaction with the fpa_sel mode.

The English specification states that, "The altitude must be pre-selected before the altitude engage button is pressed" (4). This is interpreted to mean that the command is simply ignored if an altitude has not been pre-selected. Consequently, the specification of tran_alt_eng begins:

```
tran_alt_eng(st): states =
    IF alt_disp(st) = pre_selected THEN
    ...
ELSE % IGNORE command
    ENDIF
```

This specifies that the system state will change as a result of pressing the alt_eng button only if the alt_disp is pre_selected.

We must now proceed to specify the behavior when the IF expression is true. The English specification indicates that if the aircraft is more than 1,200 feet from the target altitude, this request will be put on hold (the mode is said to be armed) and the fpa_sel mode will be engaged instead. The English specification also says that, "The pilot must then dial in a flight-path angle at this point" (3). The question arises whether the fpa_sel engagement should be delayed until this is done. Another part of the English specification offers a clue, "The calibrated air speed and flight-path angle values need not be pre-selected before the corresponding modes are engaged" (4). Although this specifically addresses the case of pressing the fpa_sel button and not the situation where the alt_eng button indirectly turns this mode on, we suspect that the behavior is the same. Nevertheless, we decide to check with the system designers to make sure. The system designers explain that this is the correct interpretation and that this is the reason the mode is called "flight-path angle select" rather than "flight-path angle engage."

The behavior must be specified for the two situations: when the airplane is near the target and when it is not. There are several ways to specify this behavior. One way is for the state specification to contain the current altitude in addition to the target altitude. This could be included in the state vector as two numbers:

```
target_altitude: number
actual_altitude: number
```

The first number contains the value dialed in and the second the value last measured by a sensor. The specification would then contain:

```
IF abs(target_altitude - actual_altitude) > 1200 THEN
```

where abs is the absolute value function. If the behavior of the mode-control panel were dependent upon the target and actual altitudes in a multitude of ways, this would probably be the proper approach. However, in the example system the behavior is only dependent upon the relation of the two values to each other. Therefore, another way to specify this behavior is by abstracting away the details of the particular values and only storing information about their relative values in the state descriptor record. In particular, the altitude field of the state record can take on one of the following three values:

```
away the pre-selected value is > 1,200 feet away
near_pre_selected the pre-selected value is <= 1,200 feet away
at_pre_selected the pre-selected value = the actual altitude
```

9
The two different situations can then be distinguished as follows:

\[
\text{IF } \text{altitude}(\text{st}) \neq \text{away} \text{ THEN}
\]

When the value is not away, the \text{alt\_eng} mode is immediately engaged. This is specified as follows:

\[
\text{IF } \text{altitude}(\text{st}) \neq \text{away} \text{ THEN}
\]
\[
\text{st WITH } [\text{alt\_eng} := \text{engaged}, \text{att\_cws} := \text{off}, \text{fpa\_sel} := \text{off}, \text{fpa\_disp} := \text{current}]
\]

Note that not only is the \text{alt\_eng} mode engaged, but this specification indicates that several other modes are affected as well. This was done because the English specification states that, “Engaging any of the first three modes will automatically cause the other two to be disengaged” (1). Thus, both \text{att\_cws} and \text{fpa\_sel} are turned off when \text{alt\_eng} is engaged. This specification also returns the FPA display to current as required in the English specification “Once the target value is achieved or the mode is disengaged, the display reverts to showing the ‘current’ value.” (2).

Now the behavior of the system must be specified for the other situation, when the aircraft is away from the target altitude. In this case \text{fpa\_sel} is engaged and \text{alt\_eng} is armed:

\[
\text{ELSE}
\]
\[
\text{st WITH } [\text{fpa\_sel} := \text{engaged}, \text{att\_cws} := \text{off}, \text{alt\_eng} := \text{armed}]
\]

As before, the \text{att\_cws} mode is also turned off.

So far we have not considered whether the behavior of the system should be different if the \text{alt\_eng} mode is already \text{armed} or \text{engaged} The English specification states that, “Pressing the altitude engage button while the system is already in altitude engage mode has no effect” (5). However, there is no information about what will happen if the mode is \text{armed}. Once again, the system designers are consulted and we are told that the mode-control panel should ignore the request in this case as well. Thus, the first IF expression should be augmented to include a test of \text{alt\_eng}(\text{st}) = \text{off}:

\[
\text{IF } \text{alt\_eng}(\text{st}) = \text{off} \text{ AND } \text{alt\_disp}(\text{st}) = \text{pre\_selected} \text{ THEN}
\]

and the complete specification of \text{tran\_alt\_eng} becomes:

\[
\text{tran\_alt\_eng}(\text{st}): \text{states} =
\]
\[
\text{IF } \text{alt\_eng}(\text{st}) = \text{off} \text{ AND } \text{alt\_disp}(\text{st}) = \text{pre\_selected} \text{ THEN}
\]
\[
\text{IF } \text{altitude}(\text{st}) \neq \text{away} \text{ THEN } \% \text{ ENGAGED}
\]
\[
\text{st WITH } [\text{att\_cws} := \text{off}, \text{fpa\_sel} := \text{off}, \text{alt\_eng} := \text{engaged}, \text{fpa\_disp} := \text{current}]
\]
\[
\text{ELSE } \% \text{ ARMED}
\]
\[
\text{st WITH } [\text{att\_cws} := \text{off}, \text{fpa\_sel} := \text{engaged}, \text{alt\_eng} := \text{armed}]
\]
\[
\text{ENDIF}
\]
\[
\text{ELSE}
\]
\[
\text{st } \% \text{ IGNORE request}
\]
\[
\text{ENDIF}
\]

Note that the last ELSE takes care of both the \text{armed} and \text{engaged} cases.
2.5.5 Input To Displays

The next three events that can occur in the system are input_alt, input_fpa, and input_cas. These occur when the pilot dials a value into one of the displays. The input_alt event corresponds to the subfunction of nextstate named tran_input_alt. The obvious thing to do is to set the appropriate fields as follows:

\[ \text{st WITH } [\text{alt}\_\text{disp} := \text{pre}\_\text{selected}] \]

This is certainly appropriate when alt_eng is off. However, we must carefully consider the two cases: (1) when the alt_eng mode is armed and (2) when it is engaged. In this case, the pilot is changing the target value after the alt_eng button has been pressed. The English specification required that the alt_eng mode be pre_selected before it could become engaged, but did not rule out the possibility that the pilot could change the target value once it was armed or engaged. We consult the system designers once again. They inform us that entering a new target altitude value should return the alt_eng mode to off and the pilot must press the alt_eng button again to re-engage the mode. We add the following to the English specification:

7. If the pilot dials in a new altitude while the alt_eng button is already engaged or armed, then the alt_eng mode is disengaged and the att_cws mode is engaged. If the alt_eng mode was armed then the fpa_sel should be disengaged as well.

The reason given by the system designers was that they didn't want the altitude dial to be able to automatically trigger a new active engagement altitude. They believed it was safer to force the pilot to press the alt_eng button again in order to change the target altitude.

Thus, the specification of tran_input_alt is:

\[
\text{tran_input_alt(st): states =}
\begin{align*}
\text{IF } & \text{alt_eng(st) = off THEN st WITH } [\text{alt}\_\text{disp} := \text{pre}\_\text{selected}] \\
\text{ELSIF } & \text{alt_eng(st) = armed OR alt_eng(st) = engaged THEN st WITH } \\
& [\text{alt}\_\text{eng} := \text{off}, \text{alt}\_\text{disp} := \text{pre}\_\text{selected}, \\
& \text{att}\_\text{cws} := \text{engaged}, \\
& \text{fpa}\_\text{sel} := \text{off}, \text{fpa}\_\text{disp} := \text{current}] \\
\text{ELSE st } & \text{no change needed already preselected ENDIF}
\end{align*}
\]

The other input event functions are similar:

\[
\text{tran_input_fpa(st): states =}
\begin{align*}
\text{IF } & \text{fpa}\_\text{sel(st) = off THEN st WITH } [\text{fpa}\_\text{disp} := \text{pre}\_\text{selected}] \\
\text{ELSE st ENDIF}
\end{align*}
\]

\[
\text{tran_input_cas(st): states =}
\begin{align*}
\text{IF } & \text{cas}\_\text{eng(st) = off THEN st WITH } [\text{cas}\_\text{disp} := \text{pre}\_\text{selected}] \\
\text{ELSE st ENDIF}
\end{align*}
\]
2.5.6 Other Actions

There are other events that are not initiated by the pilot but that still affect the mode-control panel: changes in the sensor input values. As described previously, rather than including the specific values of the altitude sensor, the state descriptor only records which of the following is true of the pre-selected altitude value:

- **away**
  - the pre_selected value is > 1,200 feet away
- **near_pre_selected**
  - the pre_selected value is <= 1,200 feet away
- **at_pre_selected**
  - the pre_selected value = the actual altitude

Events must be defined that correspond to significant changes in the altitude so as to affect the value of this field in the state. Three such events affect the behavior of the panel:

- **alt_gets_near**
  - the altitude is now near the pre_selected value but not equal
- **alt_reached**
  - the altitude reaches the pre_selected value
- **alt_gets_away**
  - the altitude is no longer near the pre_selected value

The transition subfunction associated with the first event must consider the case where the **alt_eng** mode is armed because the English spec states that "The flight-path angle select mode will remain engaged until the aircraft is within 1,200 feet of the desired altitude, then the altitude engage mode is automatically engaged." (3)

```
tran_alt_gets_near(st): states =
  IF alt_eng(st) = armed THEN
    st WITH [altitude := near_pre_selected, alt_eng := engaged, fpa_sel := off, fpa_disp := current]
  ELSE
    st WITH [altitude := near_pre_selected]
  ENDIF
```

The subfunction associated with second event is similar, because we can’t rule out the possibility that the event **alt_reached** may occur without **alt_gets_near** occurring first:

```
tran_alt_reached(st): states =
  IF alt_eng(st) = armed THEN
    st WITH [altitude := at_pre_selected, alt_disp := current, alt_eng := engaged, fpa_sel := off, fpa_disp := current]
  ELSE
    st WITH [altitude := at_pre_selected, alt_disp := current]
  ENDIF
```

Note that in this case, the **alt_disp** field is returned to current because the English specification states, “Once the target value is achieved or the mode is disengaged, the display reverts to showing the ‘current’ value.” (2).

The third event (i.e. **alt_gets_away**) is problematic in some situations. If the **alt_eng** mode is engaged, is it even possible for this event to occur? The flight-control system is actively holding the altitude of the airplane at the pre-selected value. Thus, unless there is some major external
event such as a wind-shear phenomenon, this should never occur. Of course, a real system should be able to accommodate such unexpected events. However, to shorten this example, it will be assumed that such an event is impossible, and thus is not included in the specification as a possible event\(^1\).

There is one other event corresponding to the flight-path angle being reached \texttt{fpa\_reached}. This event has no impact on the behavior of the panel other than changing the status of the FPA display.

\[
\text{tran\_fpa\_reached}(st): \text{states} = st \text{ WITH } [\text{fpa\_disp} := \text{current}]
\]

2.6 Initial State

The formal specification must include a description of the state of the system when the mode-control panel is first powered on. One way to do this would be to define a particular constant, say \texttt{st0}, that represents the initial state:

\[
\text{st0: states} = (\# \text{att\_cws} := \text{engaged}, \text{cas\_eng} := \text{off}, \\
\text{fpa\_sel} := \text{off}, \text{alt\_eng} := \text{off}, \\
\text{alt\_disp} := \text{current}, \text{fpa\_disp} := \text{current}, \\
\text{cas\_disp} := \text{current}, \text{altitude} := \text{away} \#)
\]

Alternatively, one could define a predicate (i.e., a function that returns true or false) that indicates when a state is equivalent to the initial state:

\[
\text{is\_initial}(st): \text{bool} = \\
\text{att\_cws}(st) = \text{engaged AND cas\_eng}(st) = \text{off AND fpa\_sel}(st) = \text{off AND alt\_eng}(st) = \text{off AND alt\_disp}(st) = \text{current AND fpa\_disp}(st) = \text{current AND cas\_disp}(st) = \text{current}
\]

Note that this predicate does not specify that the altitude field must have a particular value (e.g. \text{away}). Thus, this predicate defines an equivalence class of states, not all identical, in which the system could be initially. This is the more realistic way to specify the initial state since it does not designate any particular "altitude" value.

2.7 Formal Verification of the Example System

The formal specification of the mode-control panel is complete. But how does the system developer know that the specification is correct? Unlike the English specification, the formal specification is known to be detailed and precise. But it could be \textit{unambiguously wrong}. Since this is a requirements specification, there is no higher-level specification against which to prove this one. Therefore, ultimately the developer must rely on human inspection to insure that the formal specification is "what was intended." Nevertheless, the specification can be analyzed in a formal way. In particular, the developer can postulate properties that he believes should be true about the system and attempt to prove that the formal specification satisfies these properties. This process serves to reinforce the belief that the specification is what was intended. If the specification cannot be proved to meet the desired properties, the problem in the specification must be found or the property must be modified.

\(^1\)The situation where the \texttt{alt\_eng mode} is not \texttt{engaged} or \texttt{armed} is not difficult to specify, but also does not add anything pedagogically significant to the specification.
until the proof can be completed. In either case, the developer’s understanding of and confidence in the system is increased.

In the English specification of the mode-control panel, there were several statements made that characterize the overall behavior of the system. For example, “One of the three modes [att_cws, fpa_sel, or alt_eng] should be engaged at all times” (1). This statement can be formalized, and it can be proved that no matter what sequence of events occurs, this will remain true of the system. Properties such as this are often called system invariants. This particular property is formalized as follows:

\[
\text{att_cws(st)} = \text{engaged} \\
\text{OR fpa_sel(st)} = \text{engaged} \\
\text{OR alt_eng(st)} = \text{engaged}
\]

Another system invariant can be derived from the English specification: “Only one of the first three modes [att_cws, fpa_sel, alt_eng] can be engaged at any time” (1). This can be specified in several ways. One possible way is as follows:

\[
(\text{alt_eng(st)} = \text{engaged OR fpa_sel(st)} = \text{engaged}) \\
\text{AND (att_cws(st)} = \text{engaged IMPLIES} \\
\text{alt_eng(st)} = \text{engaged AND fpa_sel(st)} = \text{engaged})
\]

Finally, it would be prudent to insure that whenever alt_eng is armed, that fpa_sel is engaged:

\[
(\text{alt_eng(st)} = \text{armed IMPLIES fpa_sel(st)} = \text{engaged}).
\]

All three of these properties can be captured in one predicate (i.e. a function that is true or false) as follows:

\[
\text{valid_state(st): bool} = \\
(\text{att_cws(st)} = \text{engaged} \\
\text{OR fpa_sel(st)} = \text{engaged} \\
\text{OR alt_eng(st)} = \text{engaged}) \\
\text{AND (alt_eng(st)} = \text{engaged OR fpa_sel(st)} = \text{engaged}) \\
\text{AND (att_cws(st)} = \text{engaged IMPLIES} \\
\text{alt_eng(st)} = \text{engaged AND fpa_sel(st)} = \text{engaged}) \\
\text{AND (alt_eng(st)} = \text{armed IMPLIES fpa_sel(st)} = \text{engaged})
\]

The next step is to prove that this is always true of the system. One way to do this is to prove that the initial state of the system is valid, and that if the system is in a valid state before an event, then it is in a valid state after an event, no matter what event occurs. In other words, we must prove the following two theorems:

\[
\text{initial_good: THEOREM is_initial(st) IMPLIES valid_state(st)}
\]
\[
\text{nextstate_good: THEOREM valid_state(st) IMPLIES valid_state(nextstate(st,event))}
\]

These two theorems effectively prove by induction that the system can never enter a state that is not valid. Both of these theorems are proved by the single PVS command GRIND. The PVS system replays the proofs in 192 secs. on a Sparc 20 with 32 Megabytes of memory. The following proof reduces the execution time to 57.0 secs:
As mentioned earlier, the specification of fpa_sel contains an error. On the attempt to prove the nextstate_good theorem on the erroneous version of fpa_sel described earlier, the prover stops with the following sequent:

nextstate_good :

-1 fpa_sel(st!1) = engaged
-2 press_fpa_sel?(event!1)

|--------
1 att_cws(st!1) = engaged
2 alt_eng(st!1) = engaged
3 press_att_cws?(event!1)
4 press_alt_eng?(event!1)

The basic idea of a sequent is that one must prove that one of the statements after the |-------- is provable from the statements before it. In other words, one must prove:

-1 AND -2 ===> 1 OR 2 OR 3 OR 4

Immediately we see that formulas {3} and {4} are impossible, because press_fpa_sel?(event!1) tells us that event!1 ≠ press_fpa_sel and not press_att_cws or press_alt_eng. Thus, we must establish {1} or {2}. However, this is impossible. But, there is nothing in this sequent to require that att_cws(st!1) = engaged or alt_eng(st!1) = engaged. Thus, it is obvious at this point that something is wrong with the specification or the proof. It is clear that the difficulty surrounds the case when the event press_fpa_sel occurs, so we examine tran_fpa_sel more closely. We realize that the specification should have set att_cws to engaged as well as turning off the fpa_sel mode and alt_eng mode:

tran_fpa_sel(st): states =
IF fpa_sel(st) = off THEN
  st WITH [fpa_sel := engaged, att_cws := off, alt_eng := off, 
          alt_disp := current]
ELSE
  st WITH [fpa_sel := off, fpa_disp := current, 
           att_cws := engaged, 
           alt_eng := off, alt_disp := current]
ENDIF
This modification is necessary because otherwise the system could end up in a state where no mode was currently active. After making the correction, the proof succeeds.

It should be pointed out that the predicate good must be sufficiently strong to make the induction go through. Alternatively one can prove theorems of the form:

\[ \text{reachable}(st) \implies \text{good}(st) \]

where \( \text{reachable}(st) \) is a predicate that delineates all states that are reachable from the initial state. The predicate \( \text{reachable} \) is typically recursively defined, so the theorem in this form must be proved by using induction explicitly.

3 A Different Decomposition

Many systems can be specified using the state-machine method illustrated in this chapter. However, as the state-machine becomes complex, the specification of the state transition functions can become exceedingly complex. Therefore, many different approaches have been developed to define the state machine in a manner that is convenient for different applications. Some of the more widely known are decision tables, decision trees, state-transition diagrams, state-transition matrices, Statecharts, Superstate and R-nets [7].

Although these methods effectively accomplish the same thing—the delineation of the state machine—they vary greatly in their input format. Some use graphical notation, some use tables, and others use language constructs. Some industries have used table-oriented methods because they believe that they are more readable for specifications that require a large number of pages. An interesting new approach is being developed by the Naval Research Lab called the Software Cost Reduction (SCR) method [5]. In the SCR method, a system is a collection of state machines (called mode classes) operating in parallel. The states of the machines are called modes. The transition of these state machines can be triggered by mode changes in other machines or by a change in an input variable.

Some of the above methods decompose the specification by mode rather than by event. This approach may be easier to comprehend and thus easier to validate by human inspection. To explore this possibility, the auto-pilot will be re-specified in a manner that treats each mode in isolation. Rather than decompose the state-transition function into subfunctions according to the triggering event, we will decompose the function into subfunctions that describe each of the state components (i.e., buttons, displays, etc.) in isolation from each other:

\[
\text{NEXTSTATE}(st, \text{event}): \text{states} = \\
st \text{ WITH } [\text{att}\_\text{cws} := \text{NEXT\_ATT\_CWS}(st, \text{event}), \\
\text{cas}\_\text{eng} := \text{NEXT\_CAS\_ENG}(st, \text{event}), \\
\text{fpa}\_\text{sel} := \text{NEXT\_FPA\_SEL}(st, \text{event}), \\
\text{alt}\_\text{eng} := \text{NEXT\_ALT\_ENG}(st, \text{event}), \\
\text{cas}\_\text{disp} := \text{NEXT\_CAS\_DISP}(st, \text{event}), \\
\text{fpa}\_\text{disp} := \text{NEXT\_FPA\_DISP}(st, \text{event}), \\
\text{alt}\_\text{disp} := \text{NEXT\_ALT\_DISP}(st, \text{event}), \\
\text{altitude} := \text{NEXT\_ALTITUDE}(st, \text{event})]
\]

Each of these subfunctions are then defined by a table. For example, the \text{att\_cws} mode is defined as follows:
The first column lists all possible states of att_cws prior to the occurrence of an event. The second column delineates all of the events which change the state of the att_cws mode. The event can be simple as in the first row (i.e. press_att_cws) or it may be restricted to trigger a transition only when the state satisfies the predicate following the WHEN keyword. The third column lists the new state of the att_cws mode after the occurrence of the event.

A still more compact table can be constructed using the PVS table syntax. Each of the triggering events are listed as column headers:

**The NEXT_ATT_CWS Mode**

<table>
<thead>
<tr>
<th>press_att_cws</th>
<th>press_fpa_sel</th>
<th>press_alt_eng</th>
<th>input_alt</th>
<th>ELSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>engaged</td>
<td>off</td>
<td>Ialt(st)</td>
<td>off</td>
</tr>
<tr>
<td>engaged</td>
<td>engaged</td>
<td>Pfpa(st)</td>
<td>Palt(st)</td>
<td>engaged</td>
</tr>
</tbody>
</table>

where

\[
Palt(st) = \text{IF } \text{alt_eng(st) = off AND alt_disp(st) = pre\_selected THEN off ELSE engaged ENDIF},
\]

\[
Pfpa(st) = \text{IF } \text{fpa\_sel(st) = engaged THEN engaged ELSE off ENDIF},
\]

\[
Ialt(st) = \text{IF } \text{alt_eng(st) = armed OR alt\_eng(st) = engaged THEN engaged ELSE att\_cws(st) ENDIF}
\]

The special function definitions are necessary when the behavior is dependent upon factors other than the current event and the current value of the mode being defined. For example, when the att_cws behavior is dependent upon the current value of the alt_eng mode. In fact wherever there is a WHEN clause, a special subfunction will have to be defined. Although this approach is not as readable as the previous table, it is formalizable within PVS.

To make the tables more compact and usable, the following name changes will be made:
3.1 Re-specifying the Autopilot in a Mode Decomposition Style

Using this approach, all of the modes can be defined:

The **NEXT_ATT_CWS Mode**

<table>
<thead>
<tr>
<th>P_cws</th>
<th>P_fpa</th>
<th>P_alt</th>
<th>L_alt</th>
<th>ELSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>eng</td>
<td>Pfpa(st)</td>
<td>off</td>
<td>Ialt(st)</td>
</tr>
<tr>
<td>eng</td>
<td>eng</td>
<td>Pfpa(st)</td>
<td>Palt(st)</td>
<td>eng</td>
</tr>
</tbody>
</table>

where

\[
Palt(st) = \text{IF alt}_\text{eng}(st) = \text{off AND alt}_\text{disp}(st) = \text{pre}_\text{sel} \text{ THEN off ENDIF},
\]

\[
Pfpa(st) = \text{IF fpa}_\text{sel}(st) = \text{eng THEN eng ELSE off ENDIF},
\]

\[
Ialt(st) = \text{IF alt}_\text{eng}(st) = \text{arm OR alt}_\text{eng}(st) = \text{eng THEN eng ELSE att}_\text{cws}(st) \text{ ENDIF}
\]

The **NEXT_CAS_ENG Mode**

<table>
<thead>
<tr>
<th>P_cas</th>
<th>I_cas</th>
<th>ELSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>eng</td>
<td>off</td>
</tr>
<tr>
<td>eng</td>
<td>off</td>
<td>eng</td>
</tr>
</tbody>
</table>

The **NEXT_FPA_SEL Mode**

<table>
<thead>
<tr>
<th>P_cws</th>
<th>P_fpa</th>
<th>P_alt</th>
<th>L_alt</th>
<th>L_fpa</th>
<th>alt_re</th>
<th>alt_ne</th>
<th>ELSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>off</td>
<td>eng</td>
<td>Palt(st)</td>
<td>off</td>
<td>off</td>
<td>off</td>
<td>off</td>
</tr>
<tr>
<td>eng</td>
<td>Pcws(st)</td>
<td>off</td>
<td>Palt(st)</td>
<td>Ialt(st)</td>
<td>eng</td>
<td>altr(st)</td>
<td>eng</td>
</tr>
</tbody>
</table>

where
\[ \text{Pcws}(\text{st}) = \begin{cases} \text{off} & \text{IF att_cws}(\text{st}) = \text{off} \text{ THEN off ELSE fpa_sel}(\text{st}) \text{ ENDIF,} \\
\end{cases} \]
\[ \text{Palt}(\text{st}) = \begin{cases} \text{off} & \text{IF alt_eng}(\text{st}) = \text{off AND alt_disp}(\text{st}) = \text{pre_sel THEN} \\
\text{IF altitude}(\text{st}) (\text{st}) /= \text{away} \text{ THEN off ELSE eng ENDIF,} \\
\text{ELSE fpa_sel}(\text{st}) \text{ ENDIF,} \\
\end{cases} \]
\[ \text{Ialt}(\text{st}) = \begin{cases} \text{off} & \text{IF alt_eng}(\text{st}) = \text{arm OR alt_eng}(\text{st}) = \text{eng THEN off} \\
\text{ELSE fpa_sel}(\text{st}) \text{ ENDIF,} \\
\end{cases} \]
\[ \text{altr}(\text{st}) = \begin{cases} \text{off} & \text{IF alt_eng}(\text{st}) = \text{arm THEN off} \\
\text{ELSE fpa_sel}(\text{st}) \text{ ENDIF} \\
\end{cases} \]

The NEXT_ALT.ENG Mode

<table>
<thead>
<tr>
<th>P_cws</th>
<th>P_fpa</th>
<th>P_alt</th>
<th>I_alt</th>
<th>ELSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>off</td>
<td>off</td>
<td>off</td>
<td>Palt(st)</td>
<td>off</td>
</tr>
<tr>
<td>arm</td>
<td>Pcws(st)</td>
<td>off</td>
<td>arm</td>
<td>off</td>
</tr>
<tr>
<td>eng</td>
<td>Pcws(st)</td>
<td>off</td>
<td>eng</td>
<td>off</td>
</tr>
</tbody>
</table>

where

\[ \text{Palt}(\text{st}) = \begin{cases} \text{off} & \text{IF alt_disp}(\text{st}) = \text{pre_sel THEN} \\
\text{IF altitude}(\text{st}) (\text{st}) /= \text{away} \text{ THEN eng ELSE arm ENDIF,} \\
\text{ELSE fpa_sel}(\text{st}) \text{ ENDIF,} \\
\end{cases} \]
\[ \text{els}(\text{st}) = \begin{cases} \text{off} & \text{IF event = alt_ne OR event = alt_re THEN eng} \\
\text{ELSE arm ENDIF,} \\
\end{cases} \]
\[ \text{Pcws}(\text{st}) = \begin{cases} \text{off} & \text{IF att_cws}(\text{st}) = \text{off THEN off ELSE alt_eng}(\text{st}) \text{ ENDIF} \end{cases} \]

The NEXT.CAS.DISP Mode

<table>
<thead>
<tr>
<th>P_cas</th>
<th>I_cas</th>
<th>ELSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>current</td>
<td>current</td>
<td>Icas(st)</td>
</tr>
<tr>
<td>pre_sel</td>
<td>Pcas(st)</td>
<td>Icas(st)</td>
</tr>
</tbody>
</table>

where

\[ \text{Pcas}(\text{st}) = \begin{cases} \text{current} & \text{IF cas_eng}(\text{st}) = \text{eng THEN current} \\
\text{ELSE cas_disp}(\text{st}) \text{ ENDIF,} \\
\end{cases} \]
\[ \text{Icas}(\text{st}) = \begin{cases} \text{current} & \text{IF cas_eng}(\text{st}) = \text{off THEN pre_sel} \\
\text{ELSE cas_disp}(\text{st}) \text{ ENDIF} \end{cases} \]

The NEXT.FPA.DISPMode

<table>
<thead>
<tr>
<th>P_cws</th>
<th>P_fpa</th>
<th>P_alt</th>
<th>I_fpa</th>
<th>fpa_re</th>
<th>I_alt</th>
<th>alt_re</th>
<th>alt_ne</th>
<th>ELSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>current</td>
<td>current</td>
<td>current</td>
<td>current</td>
<td>Ifpa(st)</td>
<td>current</td>
<td>current</td>
<td>current</td>
<td>current</td>
</tr>
<tr>
<td>pre_sel</td>
<td>Pcws(st)</td>
<td>Pfpa(st)</td>
<td>Palt(st)</td>
<td>Ifpa(st)</td>
<td>current</td>
<td>Ialt(st)</td>
<td>alre(st)</td>
<td>alre(st)</td>
</tr>
</tbody>
</table>
where

\[
P_{cw}(st) = \text{IF } \text{att}_{cw}(st) = \text{off } \text{THEN } \text{current} \text{ ELSE } \text{alt}_{disp}(st) \text{ ENDIF},
\]

\[
P_{fpa}(st) = \text{IF } \text{fpa}_{sel}(st) = \text{eng } \text{THEN } \text{current} \text{ ELSE } \text{fpa}_{disp}(st) \text{ ENDIF},
\]

\[
P_{alt}(st) = \text{IF } \text{alt}_{eng}(st) = \text{off AND alt}_{disp}(st) = \text{pre}_{sel} \text{ AND }
\]
\[
\text{altitude}(st) /= \text{away } \text{THEN } \text{current} \text{ ELSE } \text{fpa}_{disp}(st) \text{ ENDIF},
\]

\[
P_{fpa}(st) = \text{IF } \text{fpa}_{sel}(st) = \text{off } \text{THEN } \text{pre}_{sel} \text{ ELSE } \text{fpa}_{disp}(st) \text{ ENDIF},
\]

\[
P_{alt}(st) = \text{IF } \text{alt}_{eng}(st) = \text{arm OR alt}_{eng}(st) = \text{eng } \text{THEN }
\]
\[
\text{current } \text{ELSE } \text{fpa}_{disp}(st) \text{ ENDIF},
\]

\[
\text{alre}(st) = \text{IF } \text{alt}_{eng}(st) = \text{arm } \text{THEN } \text{current} \text{ ELSE } \text{fpa}_{disp}(st) \text{ ENDIF}
\]

The NEXT_ALT_DISP Mode

<table>
<thead>
<tr>
<th>P_cws</th>
<th>P_fpa</th>
<th>P_alt</th>
<th>P_alre</th>
<th>ELSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>current</td>
<td>current</td>
<td>current</td>
<td>Ialt(st)</td>
<td>current │ current</td>
</tr>
<tr>
<td>pre_sel</td>
<td>P_cws(st)</td>
<td>current</td>
<td>pre_sel</td>
<td>current</td>
</tr>
</tbody>
</table>

where

\[
P_{cw}(st) = \text{IF } \text{att}_{cw}(st) = \text{off } \text{THEN } \text{current } \text{ELSE } \text{alt}_{disp}(st) \text{ ENDIF},
\]

\[
P_{alt}(st) = \text{IF } \text{alt}_{eng}(st) = \text{off } \text{THEN } \text{pre}_{sel} \text{ ELSE IF } \text{alt}_{eng}(st)(st) = \text{pre } \text{THEN } \text{alt}_{disp}(st) \text{ ELSE } \text{pre}_{sel} \text{ ENDIF}
\]

The NEXT_ALTITUDE Mode

<table>
<thead>
<tr>
<th>alt_ne</th>
<th>alt_re</th>
<th>ELSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>away</td>
<td>near</td>
<td>at</td>
</tr>
<tr>
<td>near</td>
<td>near</td>
<td>at</td>
</tr>
<tr>
<td>at</td>
<td>near</td>
<td>at</td>
</tr>
</tbody>
</table>

This method of specifying the autopilot can be shown to be equivalent to the former method, as follows:

\[
\text{test: LEMMA } \text{nextstate}(st, event) = \text{NEXTSTATE}(st, event)
\]
3.2 Some Observations About Mode Decomposition

The second method of specifying the autopilot has the advantage that each of the input buttons and displays are specified separately. Although conceptually this is a reasonable thing to do, it resulted in a more complex formulation for this example problem. This approach would work nicely where the interactions between the different modes (buttons and displays) were minimal. However, in this example, where the interactions were extensive, the resulting tables were complex. The presence of the extra functions such as $P_{fpa}$ and $I_{alt}$ clearly reveal these interactions. One could argue that it is a good thing for interactions such as these to clutter up the specification because it will serve as deterrent. But where such interactions are inevitable, this approach may lead to a more complex presentation.

Another reason that this system may not lend itself well to the mode decomposition style is that it is largely driven by external commands, i.e. from a pilot, rather than changes in variables of other state-machines. This type of system is naturally described by enumerating the effect of a pilot input (such as pressing the $att_{cws}$ button) on all of the different parts of the system state in one place:

```plaintext
tran_att_cws(st): states =
   IF att_cws(st) = off THEN
       st WITH [att_cws := engaged, fpa_sel := off,
                   alt_eng := off, alt_disp := current,
                   fpa_disp := current]
   ELSE st IGNORE: state is not altered at all
ENDIF
```

The mode-decomposition style approach specifies the result of a pilot input over many tables. For example, to understand the effect of the pilot pressing the $P_{cws}$ button, every table containing a column labeled $P_{cws}$ must be consulted. This includes the tables for $NEXT_{ATT.CWS}$, $NEXT_{FPA.SEL}$, $NEXT_{ALT.ENG}$, $NEXT_{FPA.DISP}$ and $NEXT_{ALT.DISP}$. Also note that for each of these tables except $NEXT_{ATT.CWS}$, a special function $P_{cws}$ had to be defined, because the behavior is dependent upon the current value of $att_{cws}(st)$. The mode-decomposition style specification is also larger—761 words rather than 373 words.

There is another factor that should be considered when making this comparison. Tables such as these can be used by system engineers and human factors engineers to develop the original requirements. We performed this exercise assuming that the requirements development phase had already been completed, and the formalists were merely trying to “capture” these pre-existing requirements. It may well turn out that the mode-decomposition style tables are more useful in this type of activity because the expose the complexity in a more explicit manner. Only the experience of applying these techniques to real systems will reveal the best approach.

It should also be noted that the conclusions of this section are based solely on the experience from this one example. It is that quite different conclusions will be reached for other problems.

4 Some More Analysis

The previous verification section made the observation that it is necessary that the predicate $good$ be sufficiently strong in order for the proof to go through. To see this, we will weaken the predicate as follows:
good(st): bool = (att_cws(st) = engaged OR fpa_sel(st) = engaged
  OR alt_eng(st) = engaged) AND
  (alt_eng(st) /= engaged OR fpa_sel(st) /= engaged) AND
  (att_cws(st) = engaged IMPLIES
  alt_eng(st) /= engaged AND fpa_sel(st) /= engaged)

This causes the proof of nextstate_good

nextstate_good: THEOREM good(st) IMPLIES good(nextstate(st,event))

to terminate as follows:

-1      armed?(alt_eng(st!l))
-2      engaged?(att_cws(st!l))
  |-------
  1      engaged?(fpa_sel(st!l))

This sequent is concerned about an "unreachable" state, namely, a state where att_cws and fpa_sel are engaged and alt_eng is armed. The problem is that the predicate good is no longer filtering out the non-reachable states. This illustrates why this simple verification approach will only work when the predicate good is only true for states that are actually reachable from the initial state via a sequence of nextstate executions.

The following approach is more general:

n: VAR nat
ev: VAR events
pst: VAR states
reachable_in(n,st): RECURSIVE bool =
  IF n = 0 THEN is_initial(st)
  ELSE
    (EXISTS pst,ev: st = nextstate(pst,ev) AND
      reachable_in(n-1,pst))
  ENDIF
MEASURE n

is_reachable(st): bool = (EXISTS n: reachable_in(n,st))

reachable_good: THEOREM is_reachable(st) IMPLIES good(st)

Using this approach, the predicate good can be any desired property and does not have to evaluate to false for non-reachable states. Sometimes a non-recursive expression (call it reachable?) can developed that logically follows from is_reachable.

reachable?(st): bool = (att_cws(st) = engaged OR fpa_sel(st) = engaged
  OR alt_eng(st) = engaged) AND
  (alt_eng(st) /= engaged OR fpa_sel(st) /= engaged) AND
  (att_cws(st) = engaged IMPLIES
  alt_eng(st) /= engaged AND fpa_sel(st) /= engaged) AND
  (alt_eng(st) = armed IMPLIES fpa_sel(st) = engaged)

is_reach_implies: LEMMA is_reachable(st) IMPLIES reachable?(st)
Theorems of the form \textit{is_reachable}(st) \texttt{IMPLIES} \textit{good}(st) follow trivially from lemmas of the form \textit{reachable}? \texttt{IMPLIES} \textit{good}(st).

Using our new approach, we can establish some additional properties about our specification in a straightforward manner. Accordingly we scrutinize the English specification for some additional global properties to test the formal specification against. For example, the English specification "Once the target value is achieved or the mode is disengaged, the display reverts to showing the "current" value. (2)\), leads to the following theorems:

\begin{verbatim}
safety1: THEOREM reachable?(st) AND fpa_sel(st) = engaged AND fpa_sel(nextstate(st,event)) = off IMPLIES fpa_disp(nextstate(st,event)) = current
safety2: THEOREM reachable?(st) AND alt_eng(st) = engaged AND event /= input_alt AND alt_eng(nextstate(st,event)) = off IMPLIES alt_disp(nextstate(st,event)) = current
\end{verbatim}

Note that the \textit{safety2} theorem needs the additional premise that \texttt{event /= input_alt}, because \texttt{input_alt} disengages the \texttt{alt_eng} mode but immediately preselects it again.

\section{Other Methods}

It would be interesting to compare the performance of model-checking techniques against the PVS command GRIND for problems such as these where there is only a single finite state machine. The GRIND command appears to be complete for such problems though I imagine that it will quickly be overcome by computational complexity as the problem size increases. Some of the more widely known methods are SMV \cite{1}, Murč \cite{2} and COSPAN \cite{13}. These require that the state-space of the machine be finite. Our example specification has a finite state space. However, if the values of pre-selected and measured altitude had not been abstracted away, the state-space would have been infinite.

\section{Conclusions}

The preceding discussion illustrates the process that one goes through in translating an English specification into a formal one. Although the example system was contrived to demonstrate this feature, the process demonstrated is typical for realistic systems, and the English specification for the example is actually more complete than most because the example system is small and simple.

The formal specification process forces one to clearly elaborate the behavior of a system in detail. Whereas the English specification must be examined in multiple places and interpreted in order to make a judgment about the desired system's behavior, the formal specification completely defines the behavior. Thus, the requirements capture process includes making choices about how to interpret the informal specification. Traditional software development practices force the developer to make these interpretation choices (consciously or unconsciously) during the process of creating the design or implementation. Many of the choices are hidden implicitly in the implementation without being carefully thought out or verified by the system designers, and the interpretations and clarifications are seldom faithfully recorded in the requirements document. On the other hand, the formal methods process exposes these ambiguities early in the design process and forces early and clear decisions, which are fully documented in the formal specification.
References


Appendix A: A Graphical View Of State Space

Figure 3 shows a picture of the state space of the auto-pilot with the cas_eng mode excluded. Note

\[ [\text{cws, fpa, alt}] \]

Figure 3: State Space Diagram
that the state vector is \([\text{att}_cws, \text{fpa}_sel, \text{alt}_eng]\) and

\[
\begin{align*}
E &= \text{engaged} \\
P &= \text{pre-selected} \\
A &= \text{armed} \\
O &= \text{off}
\end{align*}
\]

For example, \([E, 0, 0]\) is the state where \(\text{att}_cws = \text{engaged}, \ \text{fpa}_sel = \text{off}\) and \(\text{alt}_eng = \text{off}\). An asterisk is placed next to the letter to indicate that the corresponding display has been pre-selected. For example, \([E, 0*, 0]\) indicates that the \(\text{fpa}_sel\) mode is \(\text{off}\) and the FPA display has been pre-selected.

**Appendix B: Complete Listing of Specification**

In these section a complete listing of the specification is given for the Example System. This example can also be retrieved via the internet at


The listings are included here for easy reference.

```plaintext
defs: THEORY
BEGIN

md_status: TYPE = off, armed, engaged

off_eng: TYPE = md: md_status | md = off OR md =engaged

disp_status: TYPE = pre_selected, current

altitude_vals: TYPE = away, near_pre_selected, at_pre_selected

states: TYPE = [
    [# % RECORD
        att_cws: off_eng,
        cas_eng: off_eng,
        fpa_sel: off_eng,
        alt_eng: md_status,
        alt_disp: disp_status,
        fpa_disp: disp_status,
        cas_disp: disp_status,
        altitude: altitude_vals
    ]
]

events: TYPE = press_att_cws, press_cas_eng, press_fpa_sel, press_alt_eng,
        input_alt, input_fpa, input_cas, alt_reached,
        alt_gets_near, fpa_reached

event: VAR events
st: VAR states
nextstate(event, st): states

END defs
```

26
TRAN: THEORY
BEGIN

IMPORTING defs

event: VAR events
st: VAR states

tran_att_cws(st): states =
  IF att_cws(st) = off THEN
    st WITH [att_cws := engaged, fpa_sel := off, alt_eng := off,
    alt_disp := current, fpa_disp := current]
  ELSE st %% IGNORE
  ENDIF

tran_cas_eng(st): states =
  IF cas_eng(st) = off THEN
    st WITH [cas_eng := engaged]
  ELSE
    st WITH [cas_eng := off, cas_disp := current]
  ENDIF

tran_fpa_sel(st): states =
  IF fpa_sel(st) = off THEN
    st WITH [fpa_sel := engaged, att_cws := off, alt_eng := off,
    alt_disp := current]
  ELSE
    st WITH [fpa_sel := off, fpa_disp := current,
    att_cws := engaged,
    alt_eng := off, alt_disp := current]
  ENDIF

tran_alt_eng(st): states =
  IF alt_eng(st) = off AND alt_disp(st) = pre_selected THEN
    IF altitude(st) /= away THEN %% ENG
      st WITH [att_cws := off, fpa_sel := off, alt_eng := engaged,
      fpa_disp := current]
    ELSE %% ARM
      st WITH [att_cws := off, fpa_sel := engaged, alt_eng := armed]
    ENDIF
  ELSE %% IGNORE request
  ENDIF

tran_input_alt(st): states =
  IF alt_eng(st) = off THEN
    st WITH [alt_disp := pre_selected]
  ELSIF alt_eng(st) = armed OR alt_eng(st) = engaged THEN
    st WITH [alt_eng := off, alt_disp := pre_selected,
    att_cws := engaged,
    fpa_sel := off, fpa_disp := current]
  ELSE st %% no change needed already preselected
  ENDIF

tran_input_fpa(st): states =
IF fpa_sel(st) = off THEN st WITH [fpa_disp := pre_selected] ELSE st ENDIF

tran_input_cas(st): states =
    IF cas_eng(st) = off THEN st WITH [cas_disp := pre_selected] ELSE st ENDIF

tran_alt_gets_near(st): states =
    IF alt_eng(st) = armed THEN
        st WITH [altitude := near_pre_selected,
            alt_eng := engaged,
            fpa_sel := off, fpa_disp := current]
    ELSE
        st WITH [altitude:= near_pre_selected]
    ENDIF

tran_alt_reached(st): states =
    IF alt_eng(st) = armed THEN
        st WITH [altitude := at_pre_selected, alt_disp := current,
            alt_eng := engaged, fpa_sel := off, fpa_disp := current]
    ELSE
        st WITH [altitude:= at_pre_selected, alt_disp := current]
    ENDIF

tran_fpaReached(st): states = st WITH [fpa_disp := current]

nextstate(st,event): states =
CARES event OF
    press_att_cws: tran_att_cws(st),
    press_alt_eng: tran_alt_eng(st),
    press_fpa_sel: tran_fpa_sel(st),
    press_cas_eng: tran_cas_eng(st),
    input_alt := tran_input_alt(st),
    input_fpa := tran_input_fpa(st),
    input_cas := tran_input_cas(st),
    alt_reached := tran_alt_reached(st),
    fpa_reached := tran_fpa_reached(st),
    alt_gets_near := tran_alt_gets_near(st)
ENDCASES

END tran

panel: THEORY
BEGIN
IMPORTING tran

event: VAR events
st: VAR states
stO: states = (#
    att_cws := engaged,
cas_eng := off,
fpa_sel := off,
alt_eng := off,
alt_disp := current,
fpa_disp := current,
cas_disp := current,
altitude := away
#

good(st): bool = (att_cws(st) = engaged OR fpa_sel(st) = engaged
 OR alt_eng(st) = engaged) AND
 (alt_eng(st) /= engaged OR fpa_sel(st) /= engaged) AND
 (att_cws(st) = engaged IMPLIES
 alt_eng(st) /= engaged AND fpa_sel(st) /= engaged) AND
 (alt_eng(st) = armed IMPLIES fpa_sel(st) = engaged)

is_initial(st): bool = att_cws(st) = engaged
 AND cas_eng(st) = off
 AND fpa_sel(st) = off
 AND alt_eng(st) = off
 AND alt_disp(st) = current
 AND fpa_disp(st) = current
 AND cas_disp(st) = current

st0_good: LEMMA good(st0)
initial_good: THEOREM is_initial(st) IMPLIES good(st)
nextstate_good: THEOREM good(st) IMPLIES good(nextstate(st,event))
nextstate_good2: THEOREM good(st) IMPLIES good(nextstate(st,event))

END panel
This paper presents an introduction to capturing software requirements in the PVS formal language. The object of study is a simplified digital autopilot that was motivated in part by the mode control panel of NASA Langley's Boeing 737 research aircraft. The paper first presents the requirements for this autopilot in English and then steps the reader through a translation of these requirements into formal mathematics. Along the way deficiencies in the English specification are noted and repaired. Once completed, the formal PVS requirement is analyzed using the PVS theorem prover and shown to maintain an invariant over its state space.