Paper originally published in Electronic Communications of the EASST Volume 69 (2013).
DOI: http://dx.doi.org/10.14279/tuj.eceasst.69.962.943

Automated theorem proving for the systematic analysis of an infusion pump

M.D. Harrison[1,2], P. Masci[2], J. C. Campos[3], P. Curzon[2]

[1] School of Computing Science, Newcastle University, Newcastle-upon-Tyne, UK
michael.harrison@ncl.ac.uk

[2] Queen Mary University of London, School of Electronic Engineering & Computer Science, Mile End, London E1 4NS, UK
paolo.masci@eecs.qmul.ac.uk paul.curzon@eecs.qmul.ac.uk

[3] Dep. Informática / Universidade do Minho Braga and HASLab / INESC TEC Braga, Portugal
jose.campos@di.uminho.pt

Abstract: This paper describes the use of an automated theorem prover to analyse properties of interactive behaviour. It offers an alternative to model checking for the analysis of interactive systems. There are situations, for example when demonstrating safety, in which alternative complementary analyses provide assurance to the regulator. The rigour and detail offered by theorem proving makes it possible to explore features of the design of the interactive system, as modelled, beyond those that would be revealed using model checking. Theorem proving can also speed up proof in some circumstances. The paper illustrates how a theory generated as a basis for theorem proving (using PVS) was developed systematically from a MAL model used to model check the same properties. It also shows how the CTL properties used to check the original model can be translated into theorems.

Keywords: interactive systems, formal verification, medical devices, model checking, MAL, PVS

1 Introduction

The scaleable analysis of interactive devices using model checking techniques is now feasible [CH09, HCM13] and potentially fruitful. Formal methods have the capability to demonstrate that safety requirements, as prescribed by regulators, are true of a candidate design. However a number of barriers prevent formal methods from being the technology of choice during the development and certification of interactive systems. These barriers include difficulty of use of the techniques and the time taken to prove properties of realistic models. Work previously presented [HCM13] has demonstrated the use of the IVY tool to analyse a set of properties of interactive systems. The tool supports a relatively intuitive notation for specification and also supports template properties that can be easily instantiated to the specified model. There are limitations to the use of model checking however. One is performance. The state space generated

while model checking can become extremely large and therefore properties can take so long to check that an iterative approach to analysis becomes impossible.

Harrison and others [HCM13] describe how model checking can be used to reveal inconsistencies in the interactive behaviour of two infusion pump designs. The focus of that analysis was a systematic consideration of information that was displayed in response to action and how the mode structures of the two interfaces affect their use. The two analysed designs were medical infusion pumps commonly used in hospitals. The results provide support for human factors specialists by raising potential design issues that result from exhaustive analysis. They provide evidence that the design satisfies requirements that, if true, would mitigate interaction failures.

The IVY tool uses Modal Action Logic (MAL) to specify the effect of actions on state attributes, given preconditions. The MAL is translated into SMV to be analysed using NuSMV [CCG+02]. The models were checked against a set of property patterns. These patterns use templates that concern: the consistency of actions, the visibility of feedback, the effect of modes, and the existence of actions that enable reversal of the effect of previous actions. The templates are instantiated to the state and actions of a particular model. If a property fails to be true of the model then the checker generates a trace that indicates a sequence of actions where (according to the model) the device fails to be consistent. The trace can then be analysed from a human factors and domain perspective. While this approach is valuable, in order to make model checking tractable for systems of this size it is necessary to make radical state abstractions. In the case of [HCM13] the domains used in number entry were abstracted for the two medical infusion pumps so that it was possible to focus on interface mode structures. This paper explores the complementary role that interactive theorem proving can play. In some cases it can improve performance. It can also identify features in the model or in the design that would not have been noticed through a model checking approach. It provides a second approach to analysis that can be used to assure of the safety of a proposed design. Analysis can be used as part of an argument that risks are as low as reasonably practicable. Since there are no theorem provers for MAL or for SMV, the models were translated into PVS.

This paper takes the MAL model of a version of the Alaris infusion pump that was presented in [HCM13] and demonstrates that theorem proving based on PVS can be used to complement the analysis. The paper indicates how MAL models can be translated into PVS, and CTL properties can be translated into PVS theorems. It is not the purpose of this paper to generate a formal mapping and to prove the equivalence. For this reason it should be seen as an initial exploration. The paper is structured as follows. In section 2 research on complementary approaches is briefly discussed. In section 3 the translation from MAL to PVS is described. In section 4 the CTL properties are translated into theorems over the PVS models.

2 Background

Motivation for this analysis has been a concern with the safety, particularly in relation to user interaction failure. The U.S. Food and Drug Administration (FDA) [US 10] is now encouraging the use of safety arguments based on formal justifications to provide evidence of the safety of medical devices. They have launched the Generic Infusion Pump project to investigate solutions to safety problems in infusion pump software. Their aim is to develop a set of safety reference

models that can be used to assess safety of infusion pump software. An important element in safety arguments is to provide alternative arguments that a system is acceptably safe. Multiple arguments increase confidence. In addition the need to abstract aspects of the model to assist tractability means that some feature of the design, for example the number entry, cannot be explored.

Recent developments in model checking have made the technique easier to use relative to other formal approaches as briefly discussed in the introduction. A range of property templates (see, for example, [DAC99]) have been developed, empirically based on typical practice, that can be instantiated to the particular requirements of a device model.

Recent formal modelling work, relevant to medical devices, has focused on a number of aspects of their programming. For example, Bolton and Bass [BB10] use SAL to analyse a model of the Baxter iPump which takes into account user goals, tasks and aspects of the environment. They explore the packaging of an automated reasoning tool so that human factors engineering practitioners can specify a realistic interactive system and verify a variety of tasks. They performed the verification on a simplified model of the pump, as the state space of the full model exceeded the capabilities of the model checker.

2.1 Complementary analysis approaches

The integration of model checking with automated theorem proving has been a topic of research for many years. Rajan and others [RSS95], for example, discussed how useful logic fragments can be proved using decision procedures and Graf and Saidi [GS97] discussed how PVS could be used to construct abstract graphs. The focus of their work was to simplify proof by model checking parts of it, or using counter-examples, generated by the failure to check properties, to change the assumptions in the theorem that is being attempted (see Kong and others [KOSF05] and the automated verification approaches based on counterexample-guided refinement of abstractions [CGJ+00], for example). Our approach takes a different view. Although the proofs are structurally complex, they can be proved with a fairly simple proof strategy based on case exploration and expansion of definitions for many cases. Because of this, checking a proof by theorem proving can be much quicker (given relevant skills to control "case explosion") than would be possible with a model checker.

2.2 The PVS language

The automated theorem prover used in this paper is *Prototype Verification System (PVS)* [SORS99]. It combines an expressive specification language based on higher-order logic with an interactive prover. PVS has been used extensively in several application domains. It is based on higher-order logic with the usual basic types such as `boolean`, `integer` and `real`. New types can be introduced either in a declarative form (these types are called *uninterpreted*), or through *type constructors*. Examples of type constructors that will be used in the paper are function and record types. Function types are denoted `[D -> R]`, where `D` is the domain type and `R` is the range type. Predicates are Boolean-valued functions. Record types are defined by listing the field names and their types between square brackets and hash symbols.

Predicate subtyping is a language mechanism used for restricting the domain of a type by

using a predicate. An example of a subtype is {x:A | P(x)}, which introduces a new type as the subset of those elements of type A that satisfy the predicate P. The notation (P) is an abbreviation of the subtype expression above. Predicate subtyping is useful for specifying partial functions. Dependent subtypes can be defined, e.g., the range of a function or the type of a field in a record may depend on the value of a function argument or the value of another field in the record, respectively.

Specifications in PVS are expressed as a collection of *theories*, which consist of declarations of names for types and constants, and expressions associated with those names. Theories can be parametrised with types and constants, and can use declarations of other theories by importing them. The prelude is a standard library automatically imported by PVS. It contains a large number of useful definitions and proved facts for types, including among others common base types such as Booleans (bool) and numbers (e.g., nat, integer and real), functions, sets, and lists.

3 The PVS model of the infusion pump

The MAL models of the infusion pumps, referred to as A and B, in [HCM13] have been translated into PVS. The focus here is not rigorous translation, rather it is concerned with an intuitive description of how the approach works. For brevity this paper focuses on infusion pump A.

3.1 Overview of the infusion pump

Most infusion pumps have three basic states: infusing, holding and off. In the infusing state the volume to be infused (vtbi) is pumped into the patient intravenously at a pre-determined infusion rate. While in the infusing state the vtbi can be exhausted, in which case the pump continues in KVO (Keep Vein Open) mode and sets off an alarm. When the pump is in holding state, values and settings can be changed using a combination of function keys and chevron buttons (for the device layout, see Figure 1). A subset of the features can also be changed when infusing. Number entry is achieved by means of chevron buttons. These buttons are used to increase or decrease entered numbers incrementally. Depending on current mode the chevron buttons can be used to change infusion rate, volume to be infused and time, or alternatively allow the user to move between options in a menu, for example in bag mode and in query mode. Bag mode allows the user to select from a set of infusion bag options, thereby setting vtbi to a predetermined value. Query mode, which is invoked by pressing the query button, generates a menu of set-up options. These options depend on how the device is configured by the manufacturer, and include the means of locking the infusion rate, or disabling the locking of it, or setting vtbi and time rather than vtbi and infusion rate. There is also the possibility of changing the units of volume and infusion rate. The device allows movement between display modes via three function keys (*key1*, *key2* and *key3*). Each function key has a display associated with it, indicating its present function.

The infusion process can be captured in MAL using an invariant over the state transition process.

$$infusionrate > 0 \rightarrow infusionrateaux = infusionrate \quad (1)$$

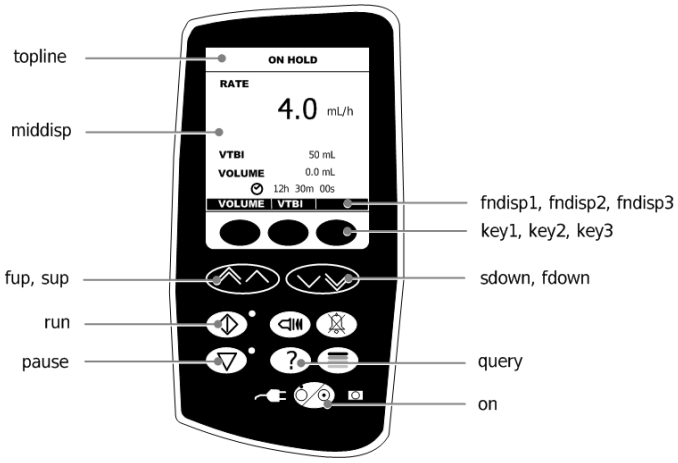

Figure 1: The pump user interface and actions

$$infusionrate > 0 \rightarrow time = (vtbi/infusionrateaux)$$
$$infusionrate = 0 \rightarrow time = 0$$

This invariant asserts a relationship between vtbi, infusion rate and time to completion of the process. *infusionrateaux*, which takes values in the range $1..maxrate$, is introduced to ensure division by zero cannot happen. To indicate the level of detail provided by the MAL model a couple of sample axioms are included and translated into PVS. The *tick* action describes the steps in the process, and the alarms that occur when the volume to be infused is exhausted, or when the device has been left in a hold state for too long. As illustration the normal conditions for *tick* are described.

$$(infusionstatus = infuse)\ \&\ (infusionrate < vtbi) \rightarrow [tick]\ vtbi' = vtbi - infusionrate\ \&\ elapsedtime' = elapsedtime + 1\ \&\ volumeinfused' = volumeinfused + infusionrate\ \&\ keep(kvorate, kvoflag, infusionrate, infusionstatus) \quad (2)$$

This axiom specifies what happens when the pump is infusing (that is $infusionstatus = infuse$) and when *vtbi* exceeds the rate, that is it will not be exhausted in this step. The axiom describes the action (in square brackets); the conditions that must be satisfied for the action to have the stated effect (left side of the implication) and the result of the action under these conditions. The priming of attributes indicates the value that will be determined in the next state. *keep* specifies those attributes that keep their values in the next state, otherwise the value is randomly determined. The PVS function `tick`, which is the translation of this action under the specified conditions, has domain that is a sub-type of the pump state that satisfies the same conditions. The range of the function is the set of all states. The attributes `vtbi`, `time` and `volumeinfused` are updated in a way that is analogous to axiom 2. The following describes that part of the `tick` function that is analogous to axiom 2.

```
tick_case_infuse_and_infusionrateLvtbi
    (st: {st: pump | infusing?(st) & vtbi(st) - infusionrate(st) > 0}): pump =
 st WITH [ vtbi := vtbi(st) - infusionrate(st),
```

```
        time := COND infusionrate(st) = 0 -> 0
                ELSE -> floor((vtbi(st) - infusionrate(st))/infusionrate(st)) ENDCOND,
        volumeinfused := COND volumeinfused(st) + infusionrate(st) <= maxinfuse
                              -> volumeinfused(st) + infusionrate(st),
                         ELSE -> volumeinfused(st) ENDCOND,
        elapsedtime  := COND elapsedtime(st) < maxtime -> elapsedtime(st) + 1,
                        ELSE -> elapsedtime(st) ENDCOND ]
```

The invariant axiom 1 is replaced by an explicit specification: `floor((vtbi(st) - infusionrate(st)) / infusionrate(st))`. The `floor` function ensures that the result is the truncated integer value associated with the quotient. The `tick` function in PVS describes the behaviour that is captured by a number of MAL axioms defining [*tick*], given a variety of pre-conditions, including `tick_case_infuse_and_infusionrateLvtbi` defined above.

```
tick(st: {st: pump | per_tick(st)}): pump =
  COND infusing?(st) & infusionrate(st) < vtbi(st)
         -> tick_case_infuse_and_infusionrateLvtbi(st),
       infusing?(st) & infusionrate(st) >= vtbi(st) & NOT kvoflag(st)
         -> tick_case_infuse_and_infusionrateGEvtbi_NOTkvoflag(st),
       infusing?(st) & infusionrate(st) >= vtbi(st) & kvoflag(st)
         -> tick_case_infuse_and_infusionrateGEvtbi_kvoflag(st),
       NOT infusing?(st) & elapse(st) >= timeout
         -> st WITH [ elapse := 0 ],
       NOT infusing?(st)   & elapse(st) < timeout
         -> st WITH [ elapse := elapse(st) + 1 ] ENDCOND
```

3.2 Specifying the interface

As discussed in [HCM13], the display is specified in the model as having three parts as shown in Figure 1. *topline* describes the contents of the top line. This is represented in MAL by a type that describes an enumeration of possible top line displays.

$$\begin{aligned} iline \;=\; \{&holding, infusing, volume, dispvtbi, attention, vtbidone, dispkvo,\\ &setvtbi, locked, options, dispinfo, vtbitime, dispblank\}\end{aligned}$$

middisp is a Boolean array indicating which pump or other state attributes are visible (for example it indicates whether a menu is visible). *fndisp1*, *fndisp2* and *fndisp3* are state attributes that describe what is indicated by the three soft keys. Further fragments of the original specification are now described to indicate the complexity of the model. The MAL specification of the soft key 2, when the top line of the device shows "holding" (see Figure 1), has two components. The first is a permission that describes when action *key2* is permitted. If the condition is not true then the action cannot be invoked. The modal axiom describes what happens when *key2* is invoked and *topline* indicates either *holding* or *infusing*.

$$\begin{aligned} per(key2) \rightarrow (&fndisp2 \;!= \; fnull)\,\&\\ &topline \; in \; \{holding, infusing, volume, dispvtbi\} \;\&\; device.poweredon\end{aligned} \tag{3}$$

This permission asserts that *key2* can be invoked when the soft key has a value other than null, and the top line is one of *holding*, *infusing*, *volume*, *dispvtbi*, and the device is powered on. The effect of *key2*, when top line shows holding or infusing and vtbi has not been exhausted (as indicated by the fact that *kvoflag* is false), is as follows:

$$(topline\ in\ \{holding, infusing\})\ \&\ !kvoflag \rightarrow [key2] \quad (4)$$
$$topline' = dispvtbi\ \&\ oldvtbi' = vtbi\ \&\ middisp[dvtbi]'\ \&\ !middisp[dvol]'\ \&$$
$$!middisp[dtime]'\ \&\ !middisp[dbags]'\ \&\ !middisp[dkvorate]'\ \&\ !middisp[dquery]'\ \&$$
$$fndisp1' = fok\ \&\ fndisp2' = fbags\ \&\ fndisp3' = fquit\ \&\ entrymode' = vtmode\ \&$$
$$effect(device.resetElapsed)\ \&\ keep(onlight, runlight, pauselight, rdisabled, rlock)$$

The translated PVS description includes the type definition for `iline`, the definition of the permission `per_key2`, and the description of the effect in the particular situation described in the MAL axiom. Finally, to indicate the context of this particular condition, the top level cases, including the one that has been specified, are described in the function `key2`. The translations of the axioms for *key2* from MAL into PVS can be achieved systematically.

```
iline: TYPE = { holding, infusing, volume, dispvtbi, attention, vtbidone, dispkvo,
                setvtbi, locked, options, dispinfo, vtbitime, dispblank, clearsetup }

per_key2(st: alaris): bool =
   NOT(fndisp2(st) = fnull) & (topline(st) = holding OR topline(st) = infusing
      OR  topline(st) = volume OR topline(st) = dispvtbi) & (device(st)`powered_on?)

key2_case_holding_infusing(st: (per_key2)): alaris =
      st WITH [ topline := dispvtbi, oldvtbi := device(st)`vtbi,
                middisp := LAMBDA(x: imid_type)
                 COND x = dvtbi -> TRUE,   x = dvol     -> FALSE, x = dtime -> FALSE,
                      x = dbags -> FALSE, x = dkvorate -> FALSE, x = dquery -> FALSE,
                      x = drate -> FALSE ENDCOND,
                fndisp1 := fok, fndisp2 := fbags, fndisp3 := fquit, entrymode := vtmode,
                device  := resetElapsed(device(st)) ]
```

The *key2* function combines all the individual MAL axioms.The domain of this function is that set of states that are permitted by `per_key2`.

```
key2(st: (per_key2)): alaris =
  COND (topline(st) = holding OR topline(st) = infusing)
                -> key2_case_holding_infusing(st),
       (topline(st) = volume)
                -> key2_case_volume(st),
       (topline(st) = dispvtbi OR topline(st) = vtbitime)
                -> key2_case_dispvtbiORvtbitime(st),
       (topline(st) = setvtbi)
                -> key2_case_setvtbi(st) ENDCOND
```

4 Proving the property templates as theorems

The model checking analysis was performed using an Intel Core rated at 3.2 GHz per processor with 24GB of RAM. The PVS analysis used an Intel Core i5 rated at 2.4 GHz with 8 GB of RAM. Given the PVS version of the interactive device, sketched in the previous section, it is possible to prove the same properties that have been proved already using the IVY tool. These were described as being concerned with a number of characteristics of the device:

- Checking that the process represented in the innermost pump layer is visible through the device interface (mirroring the process in the interface).

- Checking that modes can be determined unambiguously from the interface (mode clarity).
- Checking that actions provide appropriate feedback, for example when they change mode or change the values of pump attributes.
- Ensuring consistency of use of the display, or of action (consistency of the interface).
- Checking ease of recovery from an action.
- Ensuring that activities described in the outer layer are supported (supporting activities).

These properties have all been translated into PVS and proved of the translated specification using structural induction. Proving that a property is true for all reachable states `AG p` is achieved by: showing first that it is true of the initial state of the device; and second that if the property is true of a state then it is true of any state that can be reached from that state by a single action. For space reasons only a sample of these CTL properties are considered to illustrate the approach. All the properties shown in [HCM13] have been proved, although in some cases it has involved a tightening of the MAL model (which reflected a weakness of the MAL model rather than a lack of tractability of the PVS approach).

4.1 Mirroring the process in the interface

The first set of properties to be considered determine how the underlying modes and variables of the pump process are reflected in the interface. For example, a question considered was whether the top line of the display adequately determined whether the mode of the device was infusing or holding. Two properties were used to explore this.

$$AG(device.poweredon \;\&\; (topline\ in\ \{infusing, dispkvo, vtbidone\} \rightarrow device.infusingstate)) \tag{5}$$

This proof generated a system diameter of 53. It reached $2^{39.4686}$ states out of $2^{92.1771}$. Verification was completed in 1 hour 52 minutes. The costs of proof using model checking for the other properties are similar to this example. The run time associated with theorem proving will be indicated with each theorem. It must be emphasised that in some cases the real time, including human time, associated with theorem proving involved guiding the proof strategy and making sense of the cases where a proof failed. Dealing with counter-examples when model checking is a much simpler task than the comparable activity when theorem proving. It has not been possible in this case to quantify the time involved, including the learning curve involved, in a first serious use of PVS by the first author.

Property 5 shows that when the top line displays “infusing”, “vtbi done” or “KVO” the pump is infusing. Other top lines can appear in both infusing and holding states. For this reason property 6 that is concerned with hold excludes top lines of locked, volume, options, dispinfo and dispvtbi.

$$AG(device.poweredon \;\& \tag{6}$$

$$!(topline\ in\ \{locked, volume, options, dispinfo, dispvtbi\})$$
$$\rightarrow\ (topline\ in\ \{holding, setvtbi, attention, vtbitime, clearsetup\}$$
$$\leftrightarrow\ device.infusionstatus = hold))$$

The structural induction is achieved using `alaris_transitions`, a function that relates `pre: alaris` to all states that can be reached through an action.

```
alaris_transitions(pre, post: alaris): boolean =
  (per_sup(pre)   & post = sup(pre))   OR (per_fup(pre) & post = fup(pre))     OR
  (per_sdown(pre) & post = sdown(pre)) OR (per_fdown(pre) & post = fdown(pre)) OR
  (per_tick(pre)  & post = tick(pre))  OR (per_key1(pre) & post = key1(pre))   OR
  (per_key2(pre)  & post = key2(pre))  OR (per_key3(pre) & post = key3(pre))   OR
  (per_query(pre) & post = query(pre)) OR post = on(pre)                       OR
  (per_run(pre)   & post = run(pre))   OR (per_pause(pre) & post = pause(pre))
```

Note that the conjunction `per_action(pre) & post = action(pre)` is required because the actions are defined in PVS with a domain that is a subtype of the alaris state. Therefore any `pre: alaris` that is not in the subtype that is the domain of `action` produces an undefined value. Properties 5 and 6 can be proved using the following predicates that transform the CTL properties.

```
tlinfusionstatusinfuse(st: alaris): bool =
  (device(st)`powered_on? AND topline(st) = infusing
    AND topline(st)=dispkvo AND topline(st) = vtbidone) => device(st)`infusing?

tlinfusionstatushold(st: alaris): bool =
  (device(st)`powered_on? AND NOT(topline(st) = locked OR topline(st) = volume OR
    topline(st) = options OR topline(st) = dispinfo OR topline(st) = dispvtbi)) =>
      ((topline(st) = holding OR topline(st) = setvtbi OR topline(st) = attention
        OR topline(st) = vtbitime OR topline(st) = clearsetup) <=> NOT device(st)`infusing?)
```

The theorem combines the two properties. The PVS proof is much quicker than the equivalent property checked using model checking. The standard tactic is to skolemise the property, split it so that the initial condition can be proved separately, expand `alaris_transitions` and then split this into a case for each possible transition. These cases can then be proved relatively simply, or if they fail, the particular decomposition makes diagnosis of the problem relatively straightforward.

```
% QED Run time  = 44.38 secs. 12/3/2013
tlinfusionstatus: THEOREM
  FORALL (pre, post: alaris):
   ((init?(pre) => tlinfusionstatusinfuse(pre) AND tlinfusionstatushold(pre)) AND
      ((alaris_transitions(pre, post) AND tlinfusionstatusinfuse(pre)
         AND tlinfusionstatushold(pre))
          => tlinfusionstatusinfuse(post) AND tlinfusionstatushold(post)))
```

In proofs of this kind it was occasionally necessary to add a number of details to the theory. These were mainly in the form of additional permissions on actions to limit the states that could be in the domain of the action. This could be achieved while continuing to capture the properties of the device. There is not sufficient space in this preliminary paper to discuss the details of these additions.

4.2 Checking consistency of action

The second illustration is concerned with consistency in the use of the soft function keys. The IVY analysis explores two types of consistency: whether the same key is always associated with the same function and whether a particular soft display only appears associated with the same key. The first property is only true in some circumstances as can be specified by the CTL property.

$$AG((((topline = dispvtbi)\&(entrymode = vtmode)) \mid ((topline = vtbitime)\&(entrymode = vttmode)) \mid topline\ in\ \{options, volume, dispinfo\}) \leftrightarrow fndisp3 = fquit) \tag{7}$$

Property 7 can also be translated into a property in PVS and proved using structural induction.

```
conditions_for_quitequiv(st:alaris) : bool =
    ((((topline(st) = dispvtbi) AND (entrymode(st) = vtmode)) OR
      ((topline(st) = vtbitime) AND (entrymode(st) = vttmode)) OR
       (topline(st) = options) OR (topline(st) = dispinfo) OR (topline(st) = volume))
      <=> (fndisp3(st) = fquit))
```

with corresponding theorem:

```
%QED Run time  = 77.97 secs. 15/5/2013
alwaysquitequiv: THEOREM FORALL (pre, post: alaris):
 (init?(pre) =>  conditions_for_quitequiv(pre)) AND (alaris_transitions(pre, post)
 AND conditions_for_quitequiv(pre) => conditions_for_quitequiv(post))
```

The second type, illustrated by Property 8, ensures that quit can only appear when it is a soft key for *key3*:

$$AG(fndisp1! = fquit\ \&\ fndisp2! = fquit) \tag{8}$$

The translation of this property is:

```
never_key1_key2_quit?(st:alaris): bool =
    fndisp1(st) /= fquit AND fndisp2(st) /= fquit

%QED 18.85 secs 27/2/13
onlykey3quit1x: THEOREM FORALL (pre, post: alaris):
 (init?(pre) =>  never_key1_key2_quit?(pre)) AND (alaris_transitions(pre, post) AND
  never_key1_key2_quit?(pre) => never_key1_key2_quit?(post))
```

The final consistency illustration, property 9, requires that if the top line is volume then the same soft function keys always appear.

$$AG(topline = volume \leftrightarrow (fndisp1 = fnull\ \&\ fndisp2 = fclear\&\ fndisp3 = fquit)) \tag{9}$$

This is translated into:

```
topline_volume_fndisp?(st:alaris): bool =
 topline(st)=volume <=> (fndisp1(st)=fnull AND fndisp2(st)=fclear AND fndisp3(st)=fquit)

% Q.E.D. Run time  = 70.79 secs. 12/3/13
toplinevolumedisplaysx: THEOREM FORALL (pre, post: alaris):
 (init?(pre) =>  topline_volume_fndisp?(pre)) AND (alaris_transitions(pre, post) AND
  topline_volume_fndisp?(pre) => topline_volume_fndisp?(post))
```

4.3 Checking ease of recovery

The last illustrated property is concerned with ease of recovery. The MAL model uses a much simplified domain of numbers $(0\ldots7)$ to make analysis tractable. In the case of the PVS model, the actual number space of the infusion device is modelled and the chevron keys have behaviour as implemented in the device. The standard form of the property is illustrated in CTL as property 10.

$$AG(attribute = value \rightarrow AX(\texttt{action1} \rightarrow EX(\texttt{action2}) \,\&\, AX(\texttt{action2} \rightarrow (\texttt{attribute} = \texttt{value}))))) \tag{10}$$

Two theorems illustrate instantiations of the general form. It was possible to prove the properties over all states. Two illustrations are translated into PVS as follows:

```
% Q.E.D. Run time  = 5.91 secs. 27/2/13
undoinfusionratesupsdown: THEOREM
 (NOT rlock(st) AND entrymode(st) = rmode AND (topline(st) = holding OR
  topline(st) = infusing) AND (device(st)`infusionrate > 0) AND (per_sdown(st) AND
  per_sup(sdown(st)))) => device(sup(sdown(st)))`infusionrate = device(st)`infusionrate

% Q.E.D. Run time  = 7.03 secs.  27/2/13
undoinfusionratesdownsup: THEOREM
 (NOT rlock(st) AND entrymode(st) = rmode AND (topline(st) = holding OR
  topline(st) = infusing) AND (device(st)`infusionrate < maxrate) AND (per_sup(st) AND
  per_sdown(sup(st)))) => device(sdown(sup(st)))`infusionrate = device(st)`infusionrate
```

5 Conclusion

This paper has illustrated exploration of a currently informal process, transforming MAL models into PVS [HCM13]. Proof using interactive theorem proving for models of this kind is straightforward. However, when a proof fails, diagnosis requires expertise. The advantage of model checking is that properties can be explored by considering an ideal property and then restricting it by exploring a counter-example as discussed in [KOSF05]. Another mode in which the model checker can be used, that is difficult to achieve using the theorem prover, is to explore paths that achieve specific goals, considering counter-examples of properties such as $AG(device.volumeinfused\ != n)$.

Future work will be particularly concerned with demonstrating how models satisfy regulatory requirements, demonstrating how an approach that combines MAL with PVS can be used to prove systematically that these requirements can be proved. It is also concerned with adding tools to the IVY toolkit to enable the automatic development of PVS specifications based on MAL models and assistance with the proofs of these properties.

Acknowledgements. CHI+MED, EPSRC research grant EP/G059063/1

Bibliography

[BB10] M. L. Bolton, E. J. Bass. Formally verifying human-automation interaction as part of a system model: limitations and tradeoffs. *Innovations in System and Software Engineering* 6(3):219–231, 2010.

[CCG+02] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella. NuSMV 2: An Open Source Tool for Symbolic Model Checking. In Larsen and Brinksma (eds.), *Computer-Aided Verification (CAV '02)*. Lecture Notes in Computer Science 2404. Springer-Verlag, 2002.

[CGJ+00] E. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith. Counterexample-guided abstraction refinement. In *Computer aided verification*. Pp. 154–169. Springer Berlin Heidelberg, 2000.

[CH09] J. C. Campos, M. D. Harrison. Interaction engineering using the IVY tool. In Calvary et al. (eds.), *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. Pp. 35–44. ACM Press, 2009.

[DAC99] M. Dwyer, G. Avrunin, J. Corbett. Patterns in Property Specifications for Finite-State Verification. In *21st International Conference on Software Engineering, Los Angeles, California*. Pp. 411–420. May 1999.

[GS97] S. Graf, H. Saidi. Construction of Abstract State Graphs with PVS. In *Computer Aided Verification*. Springer Lecture Notes in Computer Science 1254, pp. 72–83. Springer-Verlag, 1997.

[HCM13] M. Harrison, J. Campos, P. Masci. Reusing models and properties in the analysis of similar interactive devices. *Innovations in Systems and Software Engineering*, 2013. doi:10.1007/s11334-013-0201-3

[KOSF05] W. Kong, K. Ogata, T. Seino, K. Futatsugi. A Lightweight Integration of Theorem Proving and Model Checking for System Verification. In *Proceedings of the 12th Asia-Pacific Software Engineering Conferende (APSEC'05)*. Pp. 8 pp.–. 2005.

[RSS95] S. Rajan, N. Shankar, M. K. Srivas. An Integration of Model Checking with Automated Proof Checking. In *Computer Aided Verification*. Springer Lecture Notes in Computer Science 939, pp. 84–97. Springer-Verlag, 1995.

[SORS99] N. Shankar, S. Owre, J. M. Rushby, D. Stringer-Calvert. PVS System Guide, PVS Language Reference, PVS Prover Guide, PVS Prelude Library, Abstract Datatypes in PVS, and Theory Interpretations in PVS. Computer Science Laboratory, SRI International, Menlo Park, CA, 1999. Available at http://pvs.csl.sri.com/documentation.shtml.

[US 10] US Food and Drug Administration. Infusion Pump Improvement Initiative. Technical report, Center for Devices and Radiological Health, April 2010. http://www.fda.gov/MedicalDevices/ProductsandMedicalProcedures/GeneralHospitalDevicesandSupplies/InfusionPumps/ucm205424.htm