Automated theorem proving for the systematic analysis of interactive systems

M.D. Harrison, P. Masci, José Creissac Campos, P. Curzon

12 pages
Automated theorem proving for the systematic analysis of interactive systems

M.D. Harrison¹, P. Masci², José Creissac Campos³, P. Curzon²

¹ michael.harrison@ncl.ac.uk Queen Mary University of London, School of Electronic Engineering & Computer Science, Mile End, London E1 4NS, and School of Computing Science, Newcastle University, Newcastle-upon-Tyne, UK

² Queen Mary University of London, School of Electronic Engineering & Computer Science, Mile End, London E1 4NS, UK

³ Dep. Informática / Universidade do Minho, Braga and HASLab / INESC TEC, Braga, Portugal

Abstract: This paper discusses the role of automated theorem proving in analysing properties of interactive behaviour. It builds on previous work that used model checking techniques to analyse a set of template properties systematically. It recognises that while model checking can be used for real-world examples, the performance of existing tools on everyday computers can be unacceptable and as a result the time taken to explore alternative design solutions, or misunderstanding in existing designs, is unacceptable within a development timescale. The paper explores how model checking can be complemented with a theorem proving approach for a large subset of the properties. The paper briefly addresses the translation of MAL models and CTL properties into PVS and replicates the results of the analysis of a particular infusion pump product.

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 though the performance of analysis tools continues to be relatively slow, as discussed in [CH09, HCM13]. Two important barriers prevent these techniques from being used by developers. The first is concerned with comprehensibility and ease of learning. It is important that the models and the analysis process are readily understood, capturing the developer’s intuition about how the device works. The IVY tool [CH09], upon which this work is based, makes progress in this direction. It uses a notation that aims to make specification easier. It also supports the analysis of properties by providing a set of property templates. The second barrier is the time taken to perform the analysis which seriously inhibits an iterative approach to analysis.

The analysis described in [HCM13] is designed to reveal inconsistencies in the interactive behaviour of two infusion pump designs. The focus of the analysis is a systematic consideration of display information in response to action and how the mode structure of the two interfaces
affects their use. The analysis provides support for human factors specialists by raising potential design issues. It can also be effective in providing evidence that interface behaviours have been explored exhaustively.

The scaleability of this technique has been demonstrated by providing evaluation of two medical infusion pumps in common use in hospitals. The IVY tool uses Modal Action Logic (MAL) which specifies the effect of actions on state attributes given preconditions. Model checking provides an algorithmic approach to analysing interactive systems systematically. The model, once produced, is analysed using a set of property patterns. These patterns include templates that relate to the consistency of actions, the visibility of feedback, the effect of modes, the existence of actions that will enable reversal of the effect of previous actions for example. The IVY tool allows these templates to be instantiated to the state and actions of the particular device. These instantiated properties can be checked of the model. If a particular property fails then the model checker generates a trace that indicates a particular sequence of actions that indicates where (according to the model) the device fails to be consistent. The analysis does not end there. The trace can then be analysed in the context of user understanding. While this approach is valuable, 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 was abstracted for the two medical infusion pumps so that it was possible to focus on interface mode structures. This paper addresses the performance problems associated with model checking by exploring complementary techniques based on model checking and automated theorem proving. It seems clear that for some properties a theorem proving approach would bring benefits in terms of performance. However there are problems. First, there are no theorem provers for MAL. Second, the translation of CTL properties into theorems is not always straightforward. Thirdly, conducting proofs using an automated theorem prover requires significant expertise.

This paper takes the MAL model of a version of the Alaris infusion pump (hereafter called A) 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. The paper is structured as follows. In section 2 research on complementary approaches is 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, particular in relation to human error. The U.S. Food and Drug Administration (FDA) [FA10] 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.

Recent developments in model checking have made model checking easier to use relative to other formal approaches. These developments have included the use of generic models that can be instantiated to a particular system and the use of property templates. A generic infusion device
underlies the specification of the two specific device models described in [HCM13] and the analysis made use of the IVY tool that simplifies property formulation by offering general property templates (see, for example, [DAC99]) that can be instantiated to the particular requirements of the devices.

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 the work has been to simplify proof by model checking parts of the proof, 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 [CGIJ100], 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 $[D \rightarrow R]$, 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 subtype is $\{x:A \mid P(x)\}$, 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
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 other common base types such as Booleans (bool) and numbers (e.g., nat, integer and real), functions, sets, and lists.

3 The PVS model of pump A

The MAL model of the infusion pump described in [HCM13] has been translated into PVS. The focus here is not a rigorous translation, rather it is concerned with an intuitive description of how the approach works. Some detail of the system modelled is provided here for clarity.

3.1 Overview of infusion pump A

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 A 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 because MAL is describing a state transition process.

\[
\begin{align*}
\text{infusionrate} > 0 \Rightarrow \text{infusionrateaux} &= \text{infusionrate} \\
\text{infusionrate} > 0 \Rightarrow \text{time} &= (\text{vtbi}/\text{infusionrateaux}) \\
\text{infusionrate} = 0 \Rightarrow \text{time} &= 0
\end{align*}
\]

This invariant asserts a relationship between vtbi, infusion rate and time to completion of the process. \text{infusionrateaux} which takes values in the range 1..maxrate is introduced to ensure
division by zero cannot happen. 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.

\[(\text{infusionstatus} = \text{infuse}) \& (\text{infusionrate} < \text{vtbi}) \Rightarrow [\text{tick}] \text{vtbi}' = \text{vtbi} - \text{infusionrate} \& (\text{elapsedtime}' = \text{elapsedtime} + 1 \& \text{volumeinfused}' = \text{volumeinfused} + \text{infusionrate} \& \text{keep}(\text{kvrate}, \text{kvoflag}, \text{infusionrate}, \text{infusionstatus}) \right)\]

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 not determined. The PVS function which is the translation of this action under the specified conditions has domain that is a sub-type of the A 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 function describes the *tick* case in the conditions described. The domain of the function is a sub-type of Alaris states that satisfy the constraints required to ensure that *vtbi* will not be exhausted in this step. The function itself sets the values of the resulting state attributes.

```pvs

\[
tick_case_infuse_and_infusionrateLvtbi
(st; \{st; \text{pump} \mid \text{infusing?}(st) \& \text{vtbi}(st) - \text{infusionrate}(st) > 0\}); \text{pump} =
st \text{WITH} \{\text{vtbi} := \text{vtbi}(st) - \text{infusionrate}(st),
    \text{time} := \text{COND} \text{infusionrate}(st) = 0 \rightarrow 0
    \text{ELSE} \rightarrow \text{floor}(\text{vtbi}(st) - \text{infusionrate}(st)) / \text{infusionrate}(st)\}) \text{ENDCOND},
    \text{volumeinfused} := \text{COND} \text{volumeinfused}(st) + \text{infusionrate}(st) \leq \text{maxinfuse}
    \rightarrow \text{volumeinfused}(st) + \text{infusionrate}(st),
    \text{ELSE} \rightarrow \text{volumeinfused}(st) \text{ENDCOND},
    \text{elapsedtime} := \text{COND} \text{elapsedtime}(st) \leq \text{maxtime} \rightarrow \text{elapsedtime}(st) + 1,
    \text{ELSE} \rightarrow \text{elapsedtime}(st) \text{ENDCOND}\}
\]
```

Figure 1: Pump ‘A’ user interface and actions
The invariant axiom 1 is replaced by an explicit specification: \( \text{floor}\left(\frac{\text{vtbi}(\text{st}) - \text{infusionrate}(\text{st})}{\text{infusionrate}(\text{st})}\right) \). The \text{floor} function ensures that the result is the truncated integer value associated with the quotient. The \text{tick} function in PVS describes the conditions in which the various transformations occur on \text{tick}.

\[
\text{tick}(\text{st}: \{\text{st}: \text{pump} | \text{per}\_\text{tick}(\text{st})\}): \text{pump} = \\
\text{COND}\ \text{infusing?(\text{st}) \& infusionrate(\text{st}) < \text{vtbi}(\text{st})} \\
\rightarrow \text{tick\_case\_infuse\_and\_infusionrateLvtbi(\text{st}),} \\
\text{infusing?(\text{st}) \& infusionrate(\text{st}) >= \text{vtbi}(\text{st}) \& \text{NOT kvoflag}(\text{st})} \\
\rightarrow \text{tick\_case\_infuse\_and\_infusionrateGEvtbi\_NOTkvoflag(\text{st}),} \\
\text{infusing?(\text{st}) \& infusionrate(\text{st}) >= \text{vtbi}(\text{st}) \& \text{kvoflag}(\text{st})} \\
\rightarrow \text{tick\_case\_infuse\_and\_infusionrateGEvtbi\_kvoflag(\text{st}),} \\
\text{NOT infusing?(\text{st}) \& \text{elapse}(\text{st}) >= \text{timeout}} \\
\rightarrow \text{st WITH } \{\ \text{elapse} := 0\ \}, \\
\text{NOT infusing?(\text{st}) \& \text{elapse}(\text{st}) < \text{timeout}} \\
\rightarrow \text{st WITH } \{\ \text{elapse} := \text{elapse}(\text{st}) + 1\ \}\text{ENDCOND}
\]

3.2 Specifying the A interface

As discussed in [HCM13] the A display is organised into three parts. \text{topline} describes the contents of the top line. This is represented in MAL by an enumeration of possible top line displays.

\[
\text{iline} = \{\text{holding, infusing, volume, dispvtbi, attention, vtbidone, dispkvo,}
\text{setvtbi, locked, options, dispinfo, vbitime, dispblank}\}
\]

\text{middisp} is a Boolean array indicating which pump or other state attributes are visible (for example it indicates whether a menu is visible). \text{fndisp1}, \text{fndisp2} and \text{fndisp3} are state attributes that describe what is indicated by the three soft keys. 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 \text{key2} is permitted. If the condition is not true then the action cannot be invoked. The modal axiom describes what happens when \text{key2} is invoked and \text{topline} indicates either \text{holding} or \text{infusing}.

\[
\text{per}(\text{key2}) \Rightarrow (\text{fndisp2} \neq \text{fnull}) \& \\
\text{topline in } \{\text{holding, infusing, volume, dispvtbi}\} \& \text{device.poweredon}
\]

This permission asserts that \text{key2} can be invoked when the soft key has a value other than null, and the top line is one of \text{holding}, \text{infusing}, \text{volume}, \text{dispvtbi} and the device is powered on. The effect of \text{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:

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

```pvs
type = { holding, infusing, volume, dispvtri, attention, vtbitime, 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) = dispvtri) & (device(st)’powered_on?)

key2_case_holding_infusing(st: (per_key2)): alaris =
    st WITH [
        topline := dispvtri,
        oldvtri := device(st)’vtbitime,
        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)) ]

Finally the whole `key2` function is described. The domain of this function is that set of states that are permitted by `per_key2`.

```pvs
4 Proving the property templates as theorems

Given the PVS version of the A pump it is possible to formulate theorems that capture the CTL properties expressed over the MAL model. These were concerned with a number of characteristics of the device. In the paper they are described as:

- 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 are all translated into PVS and proved of the translated A specification using structural induction. The A device’s states that are of interest are only those states that can be reached from the initial state. If properties are to be considered over all states they are likely to fail because all states are not permitted. 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.

Property 5 shows that when 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.

\[
AG(device\text{.}switchedon \Rightarrow \\
\text{topline in } \{\text{infusing, dispkvo, vtbidone}\} \\
\Rightarrow device\text{.}infusionstatus = \text{infuse})
\]

\[
AG(device\text{.}switchedon \& \\
\neg (\text{topline in } \{\text{locked, volume, options, dispinfo, dispvtbi}\}) \\
\Rightarrow \text{topline in } \{\text{holding, setvtbi, attention, vtbitime, clearsetup}\} \\
\iff device\text{.}infusionstatus = \text{hold})
\]

The proofs of the translated PVS theorems use structural induction. \texttt{alaris\_transitions} relates \texttt{pre: alaris} to all states that can be reached through an A action.

\[
\texttt{alaris\_transitions(pre, post: alaris): boolean =}
\]
\[
\texttt{(per\_sup(pre) \& post = sup(pre)) OR}
\]
\[
\texttt{(per\_fup(pre) \& post = fup(pre)) OR}
\]
\[
\texttt{(per\_sdown(pre) \& post = sdown(pre)) OR}
\]
\[
\texttt{(per\_fdown(pre) \& post = fdown(pre)) OR}
\]
\[
\texttt{(per\_tick(pre) \& post = tick(pre)) OR}
\]
\[
\texttt{(per\_key1(pre) \& post = key1(pre)) OR}
\]
\[
\texttt{(per\_key2(pre) \& post = key2(pre)) OR}
\]
\[
\texttt{(per\_key3(pre) \& post = key3(pre)) OR}
\]
Note that the conjunction \( \text{per}_\text{action}(\text{pre}) \land \text{post} = \text{action}(\text{pre}) \) is required because A actions are defined in PVS with a domain that is a subtype of the alaris state. Therefore any \( \text{pre}: \text{alaris} \) that is not in the subtype \( \text{action}(\text{pre}) \) produces an undefined value. Properties 5 and 6 can be proved using the following predicates that transform the CTL properties.

\[
\text{tlinfusionstatusinfuse}(\text{st}: \text{alaris}): \text{bool} = \\
(\text{device}(\text{st})\:\text{powered_on} \land \text{topline}(\text{st}) = \text{infusing} \\
\land \text{topline}(\text{st}) = \text{dispkvo} \land \text{topline}(\text{st}) = \text{vtbidone}) \Rightarrow \text{device}(\text{st})\:\text{infusing}?
\]

\[
\text{tlinfusionstatushold}(\text{st}: \text{alaris}): \text{bool} = \\
(\text{device}(\text{st})\:\text{powered_on} \land \\
\neg (\text{topline}(\text{st}) = \text{locked} \lor \text{topline}(\text{st}) = \text{volume} \lor \\
\text{topline}(\text{st}) = \text{options} \lor \text{topline}(\text{st}) = \text{dispinfo} \lor \\
\text{topline}(\text{st}) = \text{dispvtbi}) \Rightarrow \\
(\text{topline}(\text{st}) = \text{holding} \lor \text{topline}(\text{st}) = \text{setvtbi} \lor \\
\text{topline}(\text{st}) = \text{attention} \lor \text{topline}(\text{st}) = \text{vtbitime} \lor \text{topline}(\text{st}) = \text{clearsetup}) \\
\Leftrightarrow \neg \text{device}(\text{st})\:\text{infusing}?)
\]

The theorem combines the two properties. The PVS proof is much quicker than the equivalent property check using model checking. The standard tactic is to skolemise the property, split it so that the initial condition can be proved separately, expand \( \text{alaris transitions} \) 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.

\[
% \text{QED } \text{Run time } = 44.38 \text{ secs. 12/3/2013}
\text{tlinfusionstatus}: \text{THEOREM} \\
\forall \text{pre, post}: \text{alaris}: \\
(\text{init}(\text{pre}) \Rightarrow \text{tlinfusionstatusinfuse}(\text{pre}) \land \text{tlinfusionstatushold}(\text{pre})) \land \\
(\text{alaris transitions}(\text{pre}, \text{post}) \land \text{tlinfusionstatusinfuse}(\text{pre}) \\
\land \text{tlinfusionstatushold}(\text{pre})) \\
\Rightarrow \text{tlinfusionstatusinfuse}(\text{post}) \land \text{tlinfusionstatushold}(\text{post}))
\]

### 4.2 Checking consistency of action

As a second illustration consider the consistency of 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; whether a particular soft display only appears associated with the same key. The first property was only true in some circumstances:

\[
AG((\text{findisp3} \neq \text{fnull}) \land \\
!(\text{entrymode} \in \{\text{bagmode}, \text{tbagmode}\}) \land \\
!(\text{topline} \in \{\text{attention}, \text{vtbidone}, \text{setvtbi}\})) \\
\Rightarrow \text{findisp3} = \text{fquit})
\]

and was easily translated into the following positive equivalent form:
conditions_for_quit?(st:alaris) : bool =
(((topline(st) = dispvtbi) AND (entrymode(st) = vtmode)) OR
((topline(st) = vbitime) AND (entrymode(st) = vttmode)) OR
(topline(st) = options) OR (topline(st) = volume))
=> (fndisp3(st) = fquit)}

with corresponding theorem:

%QED Run time = 34.56 secs. 12/3/2013
alwaysquitx: THEOREM
FORALL (pre, post: alaris):
  (init?(pre) => conditions_for_quit?(pre)) AND
  (alaris_transitions(pre, post) AND
   conditions_for_quit?(pre)
   => conditions_for_quit?(post))

The second type of property was of the form:

$$AG((\text{fndisp1} \neq \text{fquit} \& \text{fndisp2} \neq \text{fquit}))$$ (8)

and had translation:

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))

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

which can be expressed as

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

with corresponding theorem:

%QED 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

As a final illustration, properties concerned with whether an action can be undone by another action were described as having the following standard form:

\[
AG(\text{attribute} = value) \Rightarrow
\begin{align*}
AX(\text{action1} \Rightarrow EX(\text{action2}) & \\
AX(\text{action2} \Rightarrow (\text{attribute} = value))) &
\end{align*}
\tag{10}
\]

The permissions were sufficiently tight in this case that the theorem could be proved of all states. A typical illustration could be translated into PVS as follows:

```
% QED 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
```

```
% QED 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

It is feasible to combine model checking and theorem proving techniques to analyse the interactive behaviour of real-world interactive systems. For reasons of space it was only possible to demonstrate results for a subset of properties proved in [HCM13]. The mapping from MAL to PVS and from CTL properties to PVS theorems is straightforward and can be achieved systematically. Proof is straightforward, however when a proof fails, the ability to diagnose what has gone wrong 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 is to explore paths that achieve specific goals, considering counter-examples of properties such as \( AG(device.volumetinfused ! = n) \).

Future work is concerned with exploring FDA requirements and 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

This work has been funded by the EPSRC research grant EP/G059063/1: CHI+MED (Computer–Human Interaction for Medical Devices).

Bibliography


