Layers, resources and property templates in the specification and analysis of two interactive systems

José Creissac Campos
Dep. Informática / Universidade do Minho & HASLab / INESC TEC
Braga, Portugal
jose.campos@di.uminho.pt

Paul Curzon and Paolo Masci
EECS, Queen Mary University of London
Mile End, London E1 4NS, UK
p.curzon@qmul.ac.uk, p.m.masci@qmul.ac.uk

Michael Harrison
School of Computing Science, Newcastle University, Newcastle-upon-Tyne, UK
Universidade do Minho & HASLab/INESC TEC, Queen Mary University London
michael.harrison@ncl.ac.uk

ABSTRACT
The paper briefly explores a layered approach to the analysis of two interactive systems (Nuclear Control and Air Traffic Control), indicating how the analysis enables exploration of the particular features emphasised by the use cases relating to the examples. These features relate to the interactive behaviour of the systems. To facilitate the analysis, property templates are proposed as heuristics for developing appropriate requirements for the respective user interfaces.

Author Keywords
Formal methods, interactive systems, usability heuristics

ACM Classification Keywords
H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous

INTRODUCTION

Formal modelling can have substantial benefits when they enable clarification of the assumptions made about a design. This paper looks at two case studies, provided as part of the preparation for the Workshop on Formal Methods in Human Computer Interaction. In the two cases to be discussed, formalism has already been used by the providers of the case studies to explore different features of the two examples. The nature of their analyses and the illustrative examples suggest different approaches to analysis in the two cases. The first case appears to focus on features of the device interface while the second case uses a notation for describing the tasks supported by the interface. In this paper we indicate how a common modelling approach based on layers can be used to specify the systems, enabling clear distinctions between levels of analysis while at the same time maintaining the integrity of the specification. We further comment that the analysis of properties in relation to these layers can be facilitated through the use of property templates.

Appears in Benjamin Weyers, Judy Bowen, Alan Dix, Philippe Palanque (Eds) Proceedings of the Workshop on Formal Methods in Human Computer Interaction (FoMHCI), a workshop of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS), Duisburg, Germany, June, 2015.

THE USE CASES

Two examples of safety critical interactive systems[1] discussed in the paper illustrate two distinct and important focuses commonly found in the analysis of interactive systems. Both examples present the interactive behaviour of the system, and a description of normative tasks that should be followed by operators to use the system. The first, the nuclear control example, focuses on the user interface and on checklists[2], while the second, the air traffic control (ATC) example, focuses on the user interface and distributed tasks carried out by operators and pilots in coordination.

In the first case the analysis is concerned with the role of an operator in interacting with a device. It is concerned with whether the operator can control aspects of the system in a clearly understandable way and be aware of the situation and the recovery mechanism if a failure occurs that can only be managed by the protection system. The focus of a model of the nuclear use case would be the display, the graphics, the status display, the sliders, the enabled actions and how these change the display. It would also be concerned with how effectively the protection system supports failure events. The properties of the protection system will be concerned with whether the system blocks the user effectively when unsafe actions are detected, and whether the user can trace the process through the information provided by the interface.

In the second use case there are more details about the tasks that controllers and the pilots are engaged in. The two operators have different roles and these roles are made explicit through a notation for describing the tasks. The question invited by the ATC description is how the arrival sequence display supports their activities and roles.

Whatever the level of analysis of the user interface, there are low level questions about the underlying system that are necessary to understand the design of the user interface. These

[1] `https://sites.google.com/site/wsfomchi/use-cases` (downloaded 28/5/15)

[2] `http://www.hci-modeling.org/nppsimulator/BWRSimulationDescription.pdf` (downloaded 28/5/15)

include: understanding which parts of the underlying process are visible in the user interfaces; which user actions produce visible feedback that can help the operators assess what has been done, and what has been achieved; and whether there are modes, and how transparent the effect of these modes is. In fact, interactive systems of any complexity have a common characteristic that some elements of the state of the system are perceivable (for example, visible or audible), and that user actions transform the state [5]. Furthermore, not all actions are permitted all of the time, and the behaviour of actions can depend on distinguished state attributes called *modes*, see [6] for further discussion.

STRUCTURE OF THE MODELS

It is important to distinguish between interactive systems and the components of the interactive systems. Interactive systems are socio-technical systems involving people, devices, and artefacts (desks, pieces of paper, pens, tablets and so on). The primary focus of the modelling approach illustrated here is based on the models of the interactive devices that are a part of the interactive system, whilst the property templates presented capture aspects of the system that can facilitate device-user interaction.

Users have difficulty understanding the progress of a system when elements of the state of the system, that are relevant to that understanding, are not visible in a form that makes sense to them. At the same time, confusion can arise when actions relevant to the current activity are either apparently or actually disabled by the system, or when the actions have an unexpected or inconsistent effect with respect to the users' knowledge and experiences of the system. Actions and states are therefore elemental in understanding interactive behaviour. Modes are also important. It is unusual that an interactive system is so simple that actions always have the same effect.

To achieve the goals and activities required of the users, most interactive systems are designed more or less effectively to ensure that the information required (we call them *information resources* [2]) are made explicitly available, and in a form that can be easily understood by the users. A role of a model of the interactive system is therefore to make these information resources explicit so that assumptions about the constraints they impose may be analysed.

The interface specification

Interactive systems can be specified by defining the set of actions, including user actions, that are possible within them. These actions affect the state of the system and are affected by many attributes of the state of the system. In the case of the interactive device they are often determined by the mode of the device. The model of the interactive system we develop aims to make explicit the relevant information resources needed for the analysis of the interactive behaviour of the system and that includes models of the interactive devices as well as the particular actions that define the activities that are the work of the system. The interface specification describes what the display shows and captures the effects of user level actions. The display will show some features of the state of the reactor, these features may be encoded as part of the interface. It will also show the user actions that are translated into actions within the reactor. The specification includes display widgets: showing simple status information. These include the RKS, RKT, KNT, TBN, $WP1$, $WP2$,CP, AU. These displays are associated with a range of colours. The change of colour presumably results from some combination of reactor states not made clear in the documentation. The display also shows actions associated with the valves: $SV1$, $SV2$,$WV1$, $WV2$.

Analysis of an interactive device is then concerned with proving that relevant feedback is given on completing an action, that relevant information is available before an action is carried out, that it is possible to recover from an action in specified circumstances, that it is always possible simply to step to some home mode whatever the state of the device and that actions can be completed consistently.

Structuring specifications

We structure our model of the interactive system as four layers. The first layer simply specifies the constants and types used throughout the specification. It includes types relating to the devices involved and the entities that are in the broader system. For example, in the case of the reactor these types would include pressure and volume defined as part of an aggregate type defining the tanks. There would also be types associated with pumps and valves. Constants would include maximum and minimum values that could raise error events in certain situations.

The second layer describes assumptions about the underlying process, managed or controlled by the devices, that are required as a basis for understanding the user interface specification. In practice this layer is often reused across families of device models when exploring the effects of differing user interfaces [7]. It will describe the most primitive representation of the nuclear process or the aircraft space required to consider the interactive system. A specification of the underlying reactor, describing the details of the relation between reactor core and turbine, would include attributes defining water level and pressure for each. The specification at this level would also define the characteristics of the pumps and valves. The pumps would be associated with rates per minute and the valves would be on or off. A number of actions will be specified at this level. An action $tick$ would represent the interval of one minute and update the levels and pressure depending on the setting of pump and valve. There will be further actions switching pumps on and off, opening and closing valves and changing the value of flow in the pump.

The third layer describes the interfaces for the various devices used in the interactive system. These models use the process descriptions described in the second layer. They make those aspects of the state visible through the interface explicit. They describe the user actions, including for example how the sliders or buttons or other display widgets work. The third layer of the specification of the nuclear control user interface specifies how the user sets, controls and views the operation of the device. It is specific to this particular interface, whereas the reactor specification may be more generic.

The fourth, and final, layer makes explicit the information resources that are required for different actions in different circumstances. It captures constraints on action based on the goals and activities that the user achieves or carries out [2]. This layer contains an interactive system view. The activities and actions are "resourced" by user interfaces for devices that are used in the interactive system or, indeed, any other source of relevant information that is present within the interactive system. It adds attributes that are not captured by the devices and includes (meta-)actions that describe activities that may involve actions of the interactive devices. An example of this fourth layer used in a different context can be found in [9].

Tool support

Full details of the models that are developed of the two use cases are not the focus of this paper, rather we intend an indication of our approach. Indeed different languages might be used within the context of the approach proposed depending on the type of analysis intended.

Two approaches to specification and proof are possible with the example just given: model checking and theorem proving. In the present case we focus on a theorem proving approach because an important feature of the analysis, that has issues from a user interface point of view, concerns the mechanisms for number entry. Since the domain of numbers is relatively large, proof using model checking can result in analyses of very large models that can be intractable.

The automated theorem prover used is *Prototype Verification System (PVS)* [11]. 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` on `A`. 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 and numbers (e.g., `nat`, `integer` and `real`), functions, sets, and lists.

The standard format of the specifications is that it contains a definition of a set of actions

```
action: TYPE = [state -> state]
```

which are permitted in particular situations, sometimes all situations. For each action there is a predicate

```
per_action: TYPE = [state -> boolean]
```

that indicates whether the action is permitted.

MODELLING THE CASE STUDIES

Model of the Nuclear Control User Interface

> "The operation of a nuclear power plant includes the full manual or partially manual starting and shut down of the reactor, adjusting the produced amount of electrical energy, changing the degree of automation by activating or deactivating the automated steering of certain elements of the plant, and the handling of exceptional circumstances. In case of the latter, the reactor operator primarily observes the process because the safety system of today's reactors suspends the operator step by step from the control of the reactor to return the system back to a safe state."

The interface involves schematics of the process, the availability of actions as buttons and graphical indications of key parameters, for example temperature and levels. The specification of the model can be layered according to the levels described above as follows. **The first layer** includes definitions of constants such as the maximum and minimum water levels in the reactor tank and condenser.

```
min_wl: nonneg_real
max_wl: {x: nonneg_real | x>min_wl}
```

The second layer specifies those aspects of the underlying reactor that are required to produce a model of the interface. It describes details of the relation between reactor core and turbine. These details include state attributes defining water level and pressure for each component of the core and turbine. It also includes a definition of the characteristics of the pumps and valves. Pump behaviour is abstracted as a number representing the pumping rate. Valves are abstracted as on/off switches. Actions are specified that model the events that are automatically triggered within the system. For example, an action $tick$ represents the periodic update of rate and pressure depending on the setting of pump and valve.

```
tick(st: process_state): process_state =
  p WITH [ reactor := tick(p`reactor)
           condensor := tick(p`condensor)
         ]
...
tick(r: reactor_state): reactor_state =
  r WITH [ rate := ...,
           pressure :=  ...
         ]
```

Further actions represent functions for switching pumps on and off, opening and closing valves and changing the value of flow in the pump.

The third layer describes what the reactor user interface shows on displays, what user actions are permitted, and how the system changes state in response to user actions. For the considered user interface, the model includes a specification of the actual status indicators (RKS, RKT, etc.), as well as the level and pressure of reactor and condenser.

```
state: TYPE = [#
    r: process_state
    rksm, rkt: Colour,
    rct_pressure: nonneg_real,
    SV1_state_open: boolean,
    ... #]
```

The colours of the indicators are linked to states of the underlying reactor (modelled in the second layer). The model also specifies that the user can perform open/close actions on valves, change the level of control rods, and change the rate of the pumps by interacting with controls on the user interface. All these actions are defined in terms of actions specified in the second layer of the model.

```
click_close_SV1(st: state): state =
   COND st'SV1_state_open ->  st WITH
    [r :=
     close_valve(st`process_state,sv1)],
    ELSE -> st
   ENDCOND
```

The fourth layer describes constraints on the action offered by the user interface based on the goals and activities that the user achieves or carries out [2]. For example, the action **OpenSV1** which opens a particular valve in the reactor will be appropriate in certain circumstances and for particular purposes. The information required by the operator to judge those circumstances should be visible to the operator. This information includes water levels and pressures for the relevant tank. To enable specification of these constraints an understanding of the supported activities is required. The effect of this layer of specification is to further constrain the behaviours of the user interface model to intended or plausible behaviours. The purpose of this constraint is to consider whether plausible behaviours are excluded or whether additional behaviours would be allowed by the specification that could indicate user confusions.

Actions may be specified at the level of user activity in this layer. For example, consider the user activity $recover$ in contrast to the autonomous action that causes recovery. This action would specify constraints. For example, it would specify that "increasing pressure" using the relevant action in the third layer would occur only if other actions had already been completed and the displayed tank, valve and pump parameters specified in the second layer were displayed (in the third layer) indicating particular values.

Further activities include for example "monitor recovery". This would be expressed as an action that describes the constraints on the operator when monitoring an autonomous recovery. The specification of the action would include the information resources that would be required in the monitoring process at different stages of the recovery and would specify the conditions in which any user actions would take place.

Model of the Air Traffic Controller Radar Screen

> "The AMAN (Arrival MANager) tool is a software planning tool suggesting to the air traffic controller an arrival sequence of aircraft and providing support in establishing the optimal aircraft approach routes. Its main aims are to assist the controller to optimize the runway capacity (sequence) and/or to regulate/manage (meter) the flow of aircraft entering the airspace ...'

In this case **the first layer** defines constants such as known constraints for flight (e.g., aircraft performance model parameters and constraints) and runways (e.g., maximum capacity).

The second layer captures the logic of the arrival manager planning software for suggesting arrival sequence and optimal approach routes. State attributes would specify dynamic parameters of the system, like flight plan, radar data, and weather information. An action $tick$ specifies how the suggested arrival sequence and optimal approaches are updated by the system on the basis of the actual values of flight plans, radar data, etc. Further actions specify the logic for updating dynamic parameters of the system. These will include: a trajectory predictor algorithm, a sequencer module, a weather data source, etc. Additional actions can be introduced for modelling more complex scenarios in which pilots can request emergency landing.

It is worth noting that each action is a self-contained description of how the system state changes when a given event occurs. Because of this, although adding new actions to the model makes the overall behaviour of the model more complex, it does not necessarily increase the complexity of the model. The same applies for the complexity of the analysis: if a new action does not affect the value of state variables relevant to the analysis of a property, then the complexity of the analysis of that property remains unchanged with and without the new action. This is true of theorem proving. With model checking this analysis is less straightforward.

The third layer describes what information is presented on the screens to the plan controller and executive controller. The model of the arrival manager display will therefore include a specification of the arrival timeline, time-management information, aircraft callsign, and wake turbulence category. The model of the radar screen display will include which aircraft labels are visible, their positions on the screen, and their speed vectors.

The fourth layer is based on the task descriptions provided in the use case. The task representations provide the display context required to constrain the actions. In this case actions will be activity actions, actions that are permitted by the states of the display but do not themselves change the display. They specify the assumptions about when the operators' actions are permitted.

Issues

Models of the type outlined have been developed for other interactive systems using both a model checking approach and a theorem proving approach [9, 7, 8, 1]. The advantage of model checking is that it is possible to explore, more readily, reachability properties as well as potential non-determinisms. The disadvantage is that the size of model is seriously limited. While it is possible to explore the essential details of the control of the nuclear plant using a model checking approach, this is not possible of the ATC system. Aspects such as the trajectory predictor algorithm mean the second layer of the model would be too complex. Making it abstract enough to make analysis feasible, would restrict what could be asked of the model, in terms of relevant properties to prove, making the analysis less relevant.

Theorem proving allows analysis of larger models but properties may be more difficult to formulate and prove. In particular, while model checking allows simple formulations of reachability properties, these are difficult to specify using a theorem proving approach.

There is a tradeoff to be made between the effort needed to develop a model amenable for verification and the effort need to carry out the proofs. Typically a theorem proving based approach will gain advantage in the former, because of more expressive languages, and model checking in the latter, because of more automated analysis. In all cases, how to identify and express the properties of interest is also an issue.

PROPERTY TEMPLATES

The analysis approach uses property templates as heuristics to generate properties that are tailored to the device. Tailoring the heuristics leads to insight about the device design as well as producing properties that will, if true of the design, lead to an interface being more predictable and easy to use. The heuristics that will be considered in more detail and were described in [3] are: *completeness*, *consistency*, *feedback*, *reversibility* and *visibilty*.

The heuristics have the following characteristics:

completeness captures the notion that it should be possible to reach any other state (more likely mode) in one (or a few) steps. A typical example of this property is that the design has some "home" state and a single action is sufficient to reach that home state regardless of what state the device is currently in. The first use case would suggest a completeness property which ensures that critical actions can always be taken in one step. The user interface never "modes the operator in" so that responding to a critical situation requires a complex interaction.

consistency requires that an action will always change the state of the device in a consistent way. Consistency is also concerned that similar actions have similar effects. These properties can be expressed in a number of ways and require some invention to be assured that a property is of the appropriate form. Examples of consistency properties for the Nuclear Control interface are: control rods can always be stopped; switches for valves on the user interface can always be used to change the valve states. A number of consistency properties are like these based on actions. However there are also properties that specify that a value can only change as a result of a certain type of action, or if the state is in a particular mode. An example is: all steam valves can be closed only when all feed water valves are closed, and the water level in the reactor is stable. An example of a property that related to sets of actions is that all actions that use a slider will involve similar effects.

feedback requires that an action that has an effect on the state of the system has also an effect that can be perceived by the user. For the Nuclear Control interface, an example feedback property will check that the water level indicators of the user interface correctly report changes to the actual value of the corresponding state variable of the reactor. For the Air Traffic Controller interface, an example feedback property will stipulate that all actions that have an effect on the system state will be signposted on the user interface (e.g. whether or not the speed vectors take into account changes in heading is a feedback issue).

reversibility ensures that an action can always be reversed. It is important that certain actions can be reversed. There will be constraints that limit this reversibility. For example it may be the case that an action such as opening a valve can be reversed within a specific time interval only in certain circumstances to prevent an inadvertent and extremely costly action.

visibility specifies that a state attribute in the second layer of the model is always "mirrored" by an appropriate state attribute in the third layer model. An example of such a property is that the tank level is always represented by the tank graphic in the interface or that the ATM display always correctly represents the states of the aircraft that are on approach.

For each heuristics a template is provided to express properties relevant for the heuristics. In its more general form these properties are expressed over a state machine. Hence, in its simplest form, the fact that action *ac* causes a perceivable effect (captuted by $effect_{percv}$) is expressed by

$$\forall_{s \in State} \bullet effect_{percv}(s, ac(s))$$

For the model checking case, CTL (Computational Tree Logic) and LTL (Linar Time Logic) (see [4] for an introduction) templates are provided by the IVY tool [7]. For PVS, translation of the templates is relatively straightforward, resorting to induction over the reachable states of the model.

The instances of the properties generated from the templates are usually described in terms of concepts of the third layer model. The significance of the fourth layer is that it constrains the paths for which the properties are true. The fourth layer of the specification identifies sets of plausible behaviours and in many cases the properties to be considered are required to be true only for these behaviours. For example it may be appropriate to require that any action that may occur in a path constrained by information resources will have visible feedback.

DISCUSSION AND CONCLUSIONS

Two approaches to specification and proof are possible with the considered examples: model checking and theorem proving. Model checking is the more intuitive of the two approaches. The language adopted Modal Action Logic with interactors (MAL) [3] expresses state transition behaviour in a way that is more acceptable to non-experts. The problem with model checking is that state explosion can compromise the tractability of the model so that properties to be proved are not feasible. Model checking, hence, is more convenient for analysing high level behaviour, for example when checking the modal behaviour of the user interface. Theorem proving, while being more complex to apply, provides more expressive power. This makes it more suitable when verifying properties requiring a high level of details, such as those related to a number entry system, because the domain of numbers is relatively large.

To employ the strengths of the two approaches simple rules have been used to translate from the MAL model to the PVS model that is used for theorem proving. Actions are modelled as state transformations, and permissions that are used in MAL to specify when an action is permitted are described as predicates. The details of the specification carefully reflects its MAL equivalent. This enables us to move between the notations and verification tools, choosing the more appropriate tool for the verification goals at hand.

One aspect that has not been discussed herein is the analysis and interpretation of verification results. The possibility of animating the formal models to create prototypes of the modelled interfaces, and the possibilities these prototypes raise in terms of discussing the results of verification with stakeholders has been discussed in [10]. Such prototypes can be used either to *replay* traces produced by a model checker or interactively to both discuss the findings of the verification or help identify relevant features of the system that should be addressed by formal analysis.

ACKNOWLEDGMENTS

José Creissac Campos and Michael Harrison were funded by project ref. NORTE-07-0124-FEDER-000062, co-financed by the North Portugal Regional Operational Programme (ON.2 O Novo Norte), under the National Strategic Reference Framework (NSRF), through the European Regional Development Fund (ERDF), and by national funds, through the Portuguese foundation for science and technology (FCT). Paul Curzon, Michael Harrison and Paolo Masci were funded by the CHI+MED project: Multidisciplinary Computer Human Interaction Research for the design and safe use of interactive medical devices project, UK EPSRC Grant Number EP/G059063/1.

REFERENCES

1. Campos, J., Sousa, M., Alves, M. C. B., and Harrison, M. Formal verification of a space system's user interface with the ivy workbench. *IEEE Transactions of Human Machine Systems* (2015). In Press.
2. Campos, J. C., Doherty, G., and Harrison, M. D. Analysing interactive devices based on information resource constraints. *International Journal of Human-Computer Studies 72* (2014), 284–297.
3. Campos, J. C., and Harrison, M. D. Systematic analysis of control panel interfaces using formal tools. In *Interactive systems: Design, Specification and Verification, DSVIS '08*, N. Graham and P. Palanque, Eds., no. 5136 in Springer Lecture Notes in Computer Science, Springer-Verlag (2008), 72–85.
4. Clarke, E. M., Grumberg, O., and Peled, D. A. *Model Checking*. MIT Press, 1999.
5. Duke, D. J., and Harrison, M. D. Abstract interaction objects. *Computer Graphics Forum 12*, 3 (1993), 25–36.
6. Gow, J., Thimbleby, H., and Cairns, P. Automatic critiques of interface modes. In *Proceedings 12th International Workshop on the Design, Specification and Verification of Interactive Systems*, S. Gilroy and M. Harrison, Eds., no. 3941 in Springer Lecture Notes in Computer Science, Springer-Verlag (2006), 201–212.
7. Harrison, M., Campos, J., and Masci, P. Reusing models and properties in the analysis of similar interactive devices. *Innovations in Systems and Software Engineering 11*, 2 (June 2015), 95–111.
8. Harrison, M. D., Masci, P., Campos, J., and Curzon, P. Demonstrating that medical devices satisfy user related safety requirements. In *Proceedings of Fourth Symposium on Foundations of Health Information Engineering and Systems (FHIES) & Sixth Software Engineering in Healthcare (SEHC) Workshop*, Springer-Verlag (2014). accepted.
9. Masci, P., Huang, H., Curzon, P., and Harrison, M. D. Using PVS to investigate incidents through the lens of distributed cognition. In *NASA Formal Methods*, A. Goodloe and S. Person, Eds., vol. 7226 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, 273–278.
10. Masci, P., Oladimeji, P., Curzon, P., and Thimbleby, H. PVSio-web 2.0: Joining PVS to Human-Computer Interaction. In *27th International Conference on Computer Aided Verification (CAV2015)*, Springer (2015). Tool and application examples available at http://www.pvsioweb.org.
11. Shankar, N., Owre, S., Rushby, J. M., and Stringer-Calvert, D. 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`.