Formal Aspects of Computing (2000) 3: 1–000

Representational Reasoning and Verification

Gavin J. Doherty[1,2], José C. Campos[1,3] and Michael D. Harrison[1]

Human Computer Interaction Group,
University of York, UK[1]

CLRC Rutherford Appleton Laboratory,
Oxfordshire, UK[2]

Departamento de Informática,
Universidade do Minho, Portugal[3]

Abstract. Formal approaches to the design of interactive systems rely on reasoning about properties of the system at a very high level of abstraction. Specifications to support such an approach typically provide little scope for reasoning about presentations and the *representation* of information in the presentation. In contrast, psychological theories such as distributed cognition place a strong emphasis on the role of representations, and their perception by the user, in the cognitive process. However, the post-hoc techniques for the observation and analysis of existing systems which have developed out of the theory do not help us in addressing such issues at the design stage.

In this paper we show how a formalisation can be used to investigate the representational aspects of an interface. Our goal is to provide a framework to help identify and resolve potential problems with the representation of information, and to support understanding of representational issues in design. We present a model for linking properties at the abstract and perceptual levels, and illustrate its use in a case study of a flight deck instrument. There is a widespread consensus that proper tool support is a prerequisite for the adoption of formal techniques, but the use of such tools can have a profound effect on the process itself. In order to explore this issue, we apply a higher-order logic theorem prover to the analysis.

Correspondence and offprint requests to: G. Doherty, Rutherford Appleton Laboratory, Chilton Didcot, Oxon. 0X11 0QX, UK. E-mail: G.J.Doherty@rl.ac.uk

1. Introduction

It has been proposed that formal techniques to modelling and specification can be used to improve the quality of interfaces to interactive systems [HT90, PP98]. This is especially important in safety critical domains, where "human error" is very often cited as the cause of accidents [WJCS94]. The process involves first constructing a formalisation of the aspect of the system we are interested in, and then checking to see whether certain desirable properties hold. In the area of interactive systems, these properties are intended to improve the usability of the system, and include predictability, reactivity and support for the user's task. As with the use of formal techniques in mainstream software development, this allows the designer to reason about the system at a very early stage in the development life-cycle.

Specifications which support such reasoning, for example those based on the interactor model [DH93], abstract away from the presentation as presentations typically include many details which are not relevant and are highly subject to change. Yet recent work on distributed and external cognition [WFH96, Hut95a, ZN94, SR96], postulates that representations (both internal and external) play a critical role in the cognitive process. Hutchins [Hut95b], uses this distributed view to study the role that emergent properties of a cockpit instrument can play in helping the pilot to perform his task. However, it is not clear how such analysis could be used to inform and improve the design process.

While task analysis may consider information available in the interface, it does not deal with the representation of information, unless this is explicitly manifested in alternative strategies for carrying out a task based on different information sources and representations. Theories of display structure [MSB95], based on gestalt psychology can aid in the design of displays, but do not go beyond the relationship between task and display structure. The cognitive and perceptual effort involved in using a particular presentation are not explicitly considered. We see such theory as relevant, but at a higher level of abstraction than that considered here.

Hence, the existing literature either does not consider the representation of information at the level we are interested in, or contains post-hoc techniques involving the observation of the finished artefact in use. While these techniques are undoubtedly valuable, since these properties depend on presentations (and hence representations) chosen at the design stage, it would be helpful to have a systematic analysis which could raise some of these issues at an earlier point in the life-cycle. We will see in section 2.4.1 that the manner in which the user must use information presented in an interface is often far more complex than an initial evaluation might indicate. This potential complexity in the analysis, creates the need for a systematic approach which can be applied at the specification and design stage. This motivates the use of a formal technique, which carries the additional advantage of affording the possibility of automated support [CH97].

1.1. A Formal Approach

With the above justification, the aims of this paper are twofold:

- to provide a rigorous and direct means for integrating representational reasoning in the style of [Hut95b] into the design process.
- to further explore the process of verification and show how the verification process exposes assumptions and requirements embedded in the presentation.

To address these issues we build upon the formalisation in [DH97], which we describe below. The formalisation allows us to take a rigorous and methodical approach to a form of analysis which would otherwise be conducted in an ad-hoc fashion, and which we can apply to a *specification* of the system which has a formal relationship with the artefact. In [DH97] it is shown how representational requirements could be modelled in terms of a mapping between logical operators over the abstract state and perceptual operators over the presentation. Taking a formal approach to this involves constructing a model of the abstract artefact under consideration, its representation, and the mapping between them.

This is not to say that we advocate detailed formal specification (e.g. to the level of graphics primitives) of entire presentations, a proposition we see as neither practical nor valuable, but that a limited specification, including details of the *representation* and the operators supported by the representation, is sufficient for establishing the validity of a presentation with respect to a property, such as support for a given task.

For the purposes of explanation, we could detail the proofs in a purely mathematical form, which would be more concise and readily accessible. However, from the point of view of the practitioner, use of a proof assistant not only speeds up the process greatly, but also makes it substantially less error prone. Using a theorem prover forces us to *spell out* all the assumptions we are making about the system and its presentation, and identify problems with both the presentation and the system that should be behind it. A drawback of automated tools is that the insight which is gained will probably be less than that gained from a manual proof. This poses the question of whether human-factors related conclusions can be obtained using a theorem prover, and is one we seek to address.

We hope the case study shall illustrate both the feasibility and usefulness of using tool support in the analysis, and also help identify possible problems encountered when applying the approach.

1.2. Overview

We present in the next section a description of the nature of our approach to representational properties. An example of a flight deck instrument and task to be supported are introduced[1]. Formalisation of this example reveals many representational issues. The formal notation employed is VDM-SL, but the approach can be applied to any model-based specification language. In the section following, we use this example in a proof using a higher-order logic theorem prover (PVS) to illustrate how the verification reveals further aspects of the presentation. We also consider issues involved in providing machine assistance for the process.

[1] For another case study applying the approach, see [CH99, Cam00].

2. A Model of Representation

We present in this section the model of verification for presentations of [DH97] and explore further the formalisation process.

2.1. Presentation Model

The presentation component of a user interface must provide some adequate representation of the state of the system. To model this, we begin with a simple functional model of the presentation mapping, which maps the abstract state (modelled as a set of attributes) to the presentation (modelled as a set of presentation elements, or *percepts*). We give the name ρ to the mapping from system state to presentation:

1.0 $\rho : \mathit{Attribute}\text{-}\mathsf{set} \to \mathit{Percept}\text{-}\mathsf{set}$

A given piece of information may of course have many different possible representations (ρ captures the particular choice of representation which has been made). Unlike the abstract state, the percepts represent information which can be perceived by the user, with no approximation or information loss. The mapping itself often (and necessarily) approximates the abstract attributes.

2.2. Logical and Perceptual Operators

Logical operators are those which may be defined over the abstract state (for example, magnitude comparison of two integers). Perceptual operators are those which may be defined over the presentation, and are understood to be directly performable by the user (e.g. determining if two objects on a display are adjacent). In terms of the formalisation above, logical operators are defined over *Attribute*-set, and perceptual operators are defined over *Percept*-set. Both the logical and perceptual operators can hence be formalised.

The concepts of logical and perceptual operators have previously been applied by Casner [Cas91] who constructed a system for automatic presentation generation by replacing logical operators in a task description by graphical components supporting perceptual versions of these operators. We take the converse view, and formulate our requirements on the presentation (for the specific example of task support[2]), as follows:

Task support requirement: *To support a given task, the presentation should provide perceptual equivalents of the logical operators in the task.*

By perceptual equivalents, we mean that some combination of perceptual operations can be used to achieve the same result. We believe that formalising the transformation from logical to perceptual operators provides an explicit and rigorous basis for reasoning about representational issues. For example, the scale

[2] This refers to the portion of the task to be performed by the user.

types of Stevens [Ste46], applied by Zhang [Zha96] to the analysis of relational information displays, can be formalised in terms of the groups of logical and perceptual operators that each scale supports. In this way, we can use the operators to *characterise* the representation. By trying to formulate perceptual operators over the presentation model, we can expose hidden referents in our tasks. This process serves both to increase our understanding of the system, and acts as an aid to design.

2.3. A Model of Presentation Based Properties

One way to consider the validity of a presentation is to view the perceptual model as a reification of the abstract model. We would then have to show that abstract state and presentation changes are consistent. However proving the consistency of state changes between the abstract and reified models tells us nothing about how the presentation supports the desired properties from the users perspective.

From a perceptual standpoint, for the reification to be valid, we must know that if a property holds on the abstract specification then it also holds on the presentation. We accomplish this by relating the abstract and perceptual models to a model in which the property can be expressed. Establishing the validity of the presentation then becomes a matter of showing that the logical operators over the abstract state and the perceptual operators over the representation of this abstract state yield the same result in terms of the property (see figure 1). We can express this formally as:

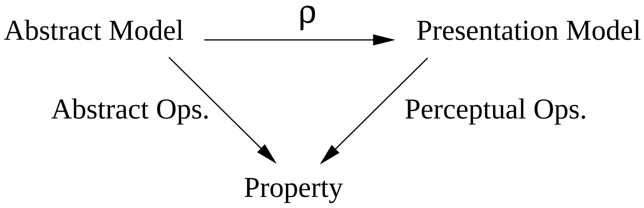

Fig. 1. Alternative approach to verification

2.0 $\mathit{perceptEquiv} : \mathit{Abstract\text{-}op} \times \mathit{Perceptual\text{-}op} \times \mathit{Attribute}\text{-}\mathsf{set} \rightarrow \mathbb{B}$

.1 $\mathit{perceptEquiv}\ (\mathit{abs}, \mathit{per}, \mathit{attrs}) \triangleq$

.2 $\quad \mathit{abs}(\mathit{attrs}) = \mathit{per}(\rho(\mathit{attrs}))$

2.4. An illustrative example

In this section, we present a formalisation of a case study, described in Hutchins [Hut95b]. Hutchins' approach involves a broader contextual view of the cockpit system, using understanding derived from distributed cognition. The example concerns the use of 'speed bugs' to record minimum manoeuvering speeds on an aircraft air speed indicator (ASI). While the example concerns a low-technology cockpit (the McDonnell-Douglas MD-80), the role played by the instrument in the pilot's tasks is sufficiently interesting to warrant analysis. Hutchins gives three descriptions of the use of 'memory' for speeds in the cockpit. The first is a procedural account including computation of the appropriate speeds from the weight of the aircraft, setting the speed bugs, and the various checks and cross-checks which are carried out during the approach. At this level of abstraction, we could apply standard techniques for task analysis [Dia89]. The second focusses on the use of external representations - cards which relate aircraft weights to sets of speeds, flight instrument displays and so on. Issues considered at this level could include the physical location of information, the duration of representations, and their malleability. The third description builds on the above and concerns the representations and processes presumed to be internal to the pilots, and in particular how the external representations can be applied in performance of the pilot's tasks.

Our analysis relates these different levels. We can see the initial description of the operators in the task as a formalisation of aspects of the first, procedural level. The formalisation of the percepts and perceptual operators is part of the second level (the external representations). The proposition that the elements of these two levels are equivalent (explored in the proof process) involves reasoning of relevance to the third level (the presence of hidden referents in the operators, the complexity of the operations performed on representations, and so on).

By using this example, we hope to illustrate how our approach achieves a good coverage of the aspects of the analysis concerning the external representation, and indicate how this might relate to a design context.

The indicator takes the form of a circular scale on which a needle indicates the current air speed (see figure 2). The 'configuration change bugs' take the form of movable tabs on the perimeter of the instrument. As the aircraft slows for landing, the wings generate less lift, and so the pilot must change between wing configurations (effectively altering the shape of the wing, and consisting of slat and flap settings) in order to generate more lift. We shall refer to this as the *configuration change task*. Each of these configurations has an associated 'minimum manoeuvering speed' below which the wings are not guaranteed to generate enough lift for safe manoeuvering of the aircraft. The configuration change bugs record these minimum manoeuvering speeds, and are set by the flight crew prior to the approach.

2.4.1. Logical Model

What is interesting about this artefact and the tasks it supports is that although on the surface it appears simple, there are in fact many pieces of information required to perform the operations involved in the tasks. In fact, we will find that under some circumstances the information presented by the artefact will not be enough (cf. section 3.4.2). The hidden complexity just mentioned becomes apparent when we formulate an initial description of the configuration management task:

```
if current speed is within acceptable margin speed
    of minimum manoeuvering speed for current configuration
then change to next configuration
```

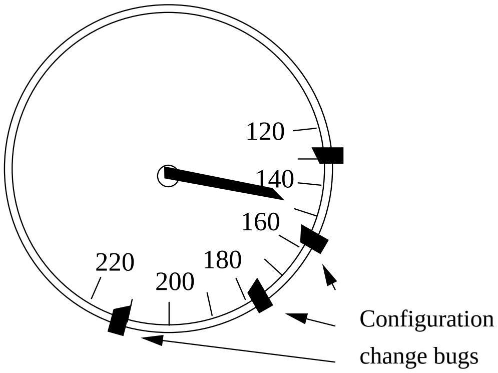

Fig. 2. Simplified Air Speed Indicator, adapted from [Hut95b]

We can see from this that there are four referents in the operation of changing configuration: the current speed, the margin speed, the minimum manoeuvering speed and the configurations being changed between (treated as one since they are paired). An operator to support this task could be one which checks a speed and configuration and determines whether it is appropriate to change to the next configuration. The first step in our formalisation is to construct a model of the ASI:

types

3.0 $\mathit{Speed} = \mathbb{R};$

4.0 $\mathit{Configuration} = \mathbb{N}_1\,;$

5.0 $\mathit{AbstractASI} :: V_c : \mathit{Speed}$ — Current Speed
.1 $C_c : \mathit{Configuration}$ — Current configuration
.2 $S_{mm} : \mathit{Speed}^{+}$ — Minimum manoeuvering speeds
.3 $V_{ref} : \mathit{Speed}$ — Reference speed of approach

values

6.0 $S_{margin} : \mathit{Speed} = 10$ Knots — Margin for configuration change

We can now formalise the logical operation to support the configuration management task as:

7.0 $\mathit{configChangeCheck} : \mathit{AbstractASI} \to \mathbb{B}$
.1 $\mathit{configChangeCheck}\,(\mathit{asi}) \triangleq$
.2 $\quad \mathit{asi}.V_c \le \mathit{asi}.S_{mm}(C_c) + S_{margin}$

We can see that this formal model provides a concise 'computational' view of the operations, and the information required to carry them out.

2.4.2. Presentation Model

The two main percepts are the ASI needle and the speed bugs. The scale is also a percept, but we use it only to establish a relationship between angles and sections of arc on the display, and absolute speeds.

We begin the specification with the data types representing the percepts. Perceptually the ASI needle is simply an angle from the upright (0 deg) position. A speed bug has both a position (again, an angle from the upright), and an extent, an angle which describes an arc to one side, clockwise from the position. The perceptual function of the scale is to relate angles on the display to speeds.

```
    types

8.0     $Angle = \mathbb{R}$;

9.0     $ASINeedle :: posn : Angle$ ;

10.0    $ASISpeedBug :: posn : Angle$
 .1                      $extent : Angle$ ;

11.0    $ASIScale :: interpret : Angle \to \mathbb{R}$

    values

12.0    $ScaleFactor : \mathbb{R}$ is not yet defined — Unit speed per scale degree ;

13.0    $BugExtent : Angle$ is not yet defined
```

The full instrument integrates these three components, note that we have a *sequence* of speed bugs, arranged in order of decreasing angle (and hence represented speed):

```
14.0  $ASI\_Instrument :: needle : ASINeedle$
 .1                        $bugs : ASISpeedBug^{+}$
 .2                        $scale : ASIScale$

 .3  inv  $asi \triangleq$ — Speed bugs in the sequence $bugs$ cannot overlap
 .4      $\forall\, i, j \in$ inds $bugs \cdot$
 .5          $i < j \;\Rightarrow\; asi.bugs(i).posn > asi.bugs(j).posn$
 .6           $\wedge\; asi.bugs(i).posn > asi.bugs(j).posn + asi.bugs(j).extent$
```

We are now in a position to formalise the presentation mapping between these two models. The presentation ρ, maps an *AbstractASI* value to an *ASI_Instrument* value which represents it. Defined below is the top-level ρ function and the lower level *ρ-Needle*, *ρ-BugSeq* and *ρ-Scale* functions which map the appropriate elements of the abstract instrument to the corresponding elements of the presentation.

```
15.0  $\rho : AbstractASI \to ASI\_Instrument$

 .1   $\rho\,(a) \triangleq$
 .2      $mk\text{-}ASI\_Instrument\,(\rho\text{-}Needle(a.V_c), \rho\text{-}BugSeq(a.S_{mm}),$
 .3                                  $\rho\text{-}Scale)$
```

```
16.0  $\rho\text{-}Needle\,(v) \triangleq$
  .1      $mk\text{-}ASINeedle\,(v/ScaleFactor)$
```

```
17.0  $\rho\text{-}BugSeq\,(s : Speed^{+})\ bs : ASISpeedBug^{+}$
  .1  post  $\forall\, i \in \mathsf{dom}\ s \cdot bs(i) = mk\text{-}ASISpeedBug\,(s(i)/ScaleFactor,$
  .2                                           $BugExtent)$
  .3          $\wedge\ len(s) = len(bs)$
```

```
18.0  $\rho\text{-}Scale\,() \triangleq$
  .1      $mk\text{-}ASIScale\,(\lambda\, a : Angle \cdot a * ScaleFactor)$
```

Now that we come to define the perceptual operators to support the configuration management task, we must compare the current speed (as represented by the needle) to the minimum manoeuvering speeds (as represented by the speed bugs). It is acceptable to make changes within a certain margin above the minimum manoeuvering speed, but it is not acceptable to go below this speed before making the configuration change. Thus the perceptual operation must be one which determines (one sided) proximity of the ASI needle to the speed bug.

```
19.0  $configBugCheck\,(needle, ccbug) \triangleq$
  .1      $in\_arc\,(needle, ccbug.posn, ccbug.posn + (S_{margin}/ScaleFactor))$
```

```
20.0  $in\_arc\,(needle, a_{start}, a_{end}) \triangleq$
  .1      $a_{start} \leq needle.posn \leq a_{end}$
```

The presence of an element of the abstract state (S_{margin}) indicates a hidden referent in the operation. This does not necessarily indicate a serious inadequacy of the presentation, (for example, information may sometimes be provided by other artifacts), although it does point to a lack of integration with the other percepts involved in the operation. In this case S_{margin} is a constant which is constrained by "operational considerations" [Hut95b].

But there is also the issue of the current and next configuration. Ultimately, the perceptual operator must relate the abstract artefact (a sequence of minimum manoeuvering speeds) to a sequence of speed bugs around the perimeter of the ASI. We could use the ordering of the bugs as a simple formalisation, thus we employ a perceptual operator which indexes the sequence of speed bugs with the current configuration:

```
21.0  $getCurrentBug\,(C_c, bugs) \triangleq$
  .1      $index\,(C_c, bugs)$
```

We can see in this expression a requirement that the user already know the current configuration (C_c) or that it be represented in another artefact (which itself must enable the user to extract the information perceptually). Another (and perhaps more realistic) formalisation would be based on proximity of the ASI needle. Thus the 'next lowest' speed bug on the ASI is the one we want.

```
22.0  $getCurrentBug\,(needle, bugs) \triangleq$
  .1      $next\_counterclockwise\,(needle, bugs)$
```

23.0 $\mathit{next\_counterclockwise}\ (\mathit{needle} : \mathit{ASINeedle}, \mathit{bugs} : \mathit{ASISpeedBug}^{+})\ \mathit{bug} :$ $\mathit{ASISpeedBug}$

.1 **pre** $\exists\, i \in \mathbb{N} \cdot \mathit{bugs}(i).\mathit{posn} < \mathit{needle}.\mathit{posn}$

.2 **post** $\exists\, i \in \mathbb{N} \cdot \mathit{bugs}(i) = \mathit{bug} \wedge \mathit{bugs}(i).\mathit{posn} \leq \mathit{needle}.\mathit{posn}$

.3 $\wedge\, \forall\, j \in \mathbb{N} \cdot j < i \;\Rightarrow\; \mathit{bugs}(j).\mathit{posn} > \mathit{needle}.\mathit{posn}$

Integrating these two operations into a composite operation to support the configuration change task yields:

24.0 $\mathit{asiConfigCheck} : \mathit{ASI\_Instrument} \to \mathbb{B}$

.1 $\mathit{asiConfigCheck}\ (\mathit{asi}) \triangleq$

.2 $\mathit{configBugCheck}\ (\mathit{asi}.\mathit{needle}, \mathit{getCurrentBug}\ (\mathit{asi}.\mathit{needle}, \mathit{asi}.\mathit{bugs}))$

We can see from the above that the process of formalisation itself contributes to our understanding of the system, and possible shortcomings. The next section will illustrate how more complex assumptions embedded in the representation may be discovered.

3. Verification

Having defined both the abstract and presentation models, operators over these models and the mapping between them, we can proceed with the verification phase of our analysis. Recall that the form of verification we are using involves establishing an equivalence between logical operators over the abstract state and perceptual operators over the presentation.

At this point, it is worth reiterating the aims of the process. The aim is not to show that the proofs can be done (indeed, formally they are quite simple), but rather to show that something can be learned regarding the interaction between system and user. We do not argue that the current analysis could not be conducted without the aid of a theorem prover, but as models and proofs grow in size and complexity, the ability to manage and partly automate the verification becomes very useful, as work on the verification of protocols has shown. Our aim is to show how we can derive insight into the interaction between human and machine. Since the analysis of such interaction involves a number of different concerns, ranging from software engineering to psychology, it is not necessarily obvious that such insights can be obtained.

3.1. The prover - PVS

In this section we will introduce PVS, the theorem prover that will be used in the verification process that follows. Our aim is to enable the reader to follow the subsequent description of the performed verification. See [OSR93] for a more thorough introduction to the system.

PVS is a typed higher-order logic theorem prover, which provides an integrated environment for development and analysis of specifications. Specifications are organised in theories. Typically a theory will introduce a number of types and constants (which can be functions), and formulas associated with them (axioms and theorems, for instance). Theories can be parameterised on types and constants. Entities declared in a theory can be made available to others by exporting them (using the EXPORTING clause). By default all declarations are exported. Entities that are exported in a theory can be imported by another using the IMPORTING clause.

PVS features a powerful type system. This is very useful when writing specifications, but means that type checking becomes undecidable. To cope with this, the type checker generates proof obligations (TCCs - Type Correctness Conditions) that must be established by the theorem prover. If the system is unable to prove a TCC automatically, then the user is asked to do it.

The usual types are available in PVS: natural numbers (nat), real numbers (real), sequences (sequence[$\mathcal{X}$]), sets (set[$\mathcal{X}$]), tuples ([#...#]), etc. These types are either built-in in the system or defined in the prelude library. PVS also allows for the definition of predicate subtypes, dependent types and abstract data types. Although we will not use these directly, they are used in the prelude to define some of the types we will be using. A library is a collection of theorems. The prelude is a special library whose theories are always available, without the need for explicit importing.

PVS is used interactively. Its interface is mainly implemented as an Emacs major mode, which integrates functionality for editing specifications and proving theorems. When performing a proof, the system presents a goal in the form of a sequent, and prompts the user for an appropriate command. If the command does not solve the sequent, it will generate a new sequent or a number of new sequents (ie. subgoals), and the user will be asked to prove them in turn. When all subgoals are proved the original goal has been proved. The user interacts with the prover by issuing commands to be applied to the sequent. There are commands for induction, quantifier reasoning, rewriting, simplification using decision procedures and type information, and propositional simplification using binary decision diagrams. In the analysis that follows we will be using PVS version 2.3 (patch level 1.2.2.36).

3.2. Writing the specification in PVS

The translation from VDM to PVS is straightforward [Age96]. The specification is organised into three theories. One for the abstract model (theory ASI), another for the perceptual model (theory perceptualASI), and a final theory that introduces the equivalences to be proved (theory ASIverification).

Figure 3 presents the PVS theory for the abstract model as described in section 2.4.1. The theory starts by introducing the types. Besides the three types present in the VDM model, a fourth type, Speeds (sequences of Speed), is declared for use in abstractASI. Note the syntax for tuples used in abstractASI. After the types, a constant of type Speed is introduced: Smargin. Note that the constant is left uninterpreted (ie. no actual value is given). Finally, the theory declares the logical operator (configChangeCheck). Note that the syntax for access to composite types is of a functional style. The theory in figure 3 was obtained by translating the VDM specification. As will be shown in sections 3.3 and 3.4 the verification process will prompt us to introduce changes to the specification. In

```
ASI : THEORY
  BEGIN

  Speed : TYPE   = real
  Speeds : TYPE   = sequence[Speed]
  Configuration : TYPE   = nat
  AbstractASI : TYPE   = [# Vc : Speed,
                            Cc : Configuration,
                            Smm : Speeds,
                            Vref : Speed#]

  Smargin : Speed

  configChangeCheck((asi : AbstractASI)) : bool =
                    Vc(asi) ≤ Smm(asi)(Cc(asi)) + Smargin

  END ASI
```

Fig. 3. Initial version of the abstract model

Appendix A we present the final ASI theory that resulted from the verification process.

The perceptualASI theory is defined similarly. The final version (including the changes prompted by the verification process) is given in Appendix B. Note how invariants have been defined using predicate sub-typing. A predicate is introduced stating the invariant, and the type is defined as all those values that verify the predicate. The notation (pred) is a shorthand for $\{x : T \mid \mathit{pred}(x)\}$ (see predicate inv_ASIInstrument and type ASIInstrument in Appendix B).

```
ASIverification : THEORY
  BEGIN

  IMPORTING ASI, perceptualASI

  abs_asi : VAR AbstractASI

  configuration_change_task : CONJECTURE
    configChangeCheck(abs_asi) = asiConfigCheck(ρ(abs_asi))

  END ASIverification
```

Fig. 4. ASIverification theory

The last theory, ASIverification (see figure 4), introduces the equivalence to be proved as a conjecture. Again, the final version of the theory is presented in Appendix C.

By attempting to prove the equivalence in figure 4, which correspond to the analysis outlined in section 2.3, we hope to raise questions about representational properties of the artefact, and in particular to derive hidden assumptions about the representation. This process contributes to an increased understanding of the system; it may indicate potential problems with the system and may also suggest ways in which the presentation might better support performance of the users task. Before we proceed with the proofs, however, we must type check the specification.

3.3. Type Checking

The first step after writing a PVS specification is type checking it. This might generate a number of Type Correctness Conditions (TCCs). PVS will attempt to prove all TCCs automatically. When some proof fails, it is left for the user to finish. When attempting the type checking of the theories above, the following proof obligation is generated:

$$\begin{array}{l}\text{rho\_TCC1}: \textsc{obligation}\\ \quad \forall(a: \text{abstractASI}): \text{inv\_ASIInstrument}((\#\ \begin{array}[t]{l}\text{needle:} = \text{rho\_Needle}(\text{Vc}(a)),\\ \text{bugs:} = \text{rho\_BugSeq}(\text{Smm}(a)),\\ \text{scale:} = \text{rho\_Scale}\#))\end{array}\end{array}$$

which can be more easily understood if rewritten as:

$$\begin{array}{l}\text{rho\_TCC1}: \textsc{obligation}\\ \quad \forall(a: \text{abstractASI}): \text{inv\_ASIInstrument}(\rho(a))\end{array}$$

That is, PVS needs to prove that, for all values in abstractASI, ρ will generate a valid presentation. When PVS attempts to prove the obligation, two sequents are reached which cannot be solved. The first sequent:

Sequent 1. rho_TCC1.1:

$$\begin{array}{ll}\{\text{-1}\} & \text{i!1} \geq 0\\ \{\text{-1}\} & \text{j!1} \geq 0\\ \{\text{-3}\} & \text{i!1} < \text{j!1}\\ \hline \{1\} & \text{Smm(a!1)(i!1)/ScaleFactor} > \text{Smm(a!1)(j!1)/ScaleFactor}\end{array}$$

amounts to having to prove that the list of speeds at the abstract level (Smm) is sorted. Since no such restriction exists at the abstract level, it can be concluded that not all values of Smm will have a valid representation. To solve this, we add an invariant to the abstract level theory stating the Smm must be sorted (see predicate inv_abs_asi and definition of AbstractASI in Appendix A).

The second sequent:

Sequent 2. rho_TCC1.2:

$$\begin{array}{ll}\{\text{-1}\} & \text{i!1} \geq 0\\ \{\text{-1}\} & \text{j!1} \geq 0\\ \{\text{-3}\} & \text{i!1} < \text{j!1}\\ \hline \{1\} & \text{Smm(a!1)(i!1)/ScaleFactor}\\ & \qquad\qquad > \text{BugExtent} + \text{Smm(a!1)(j!1)/ScaleFactor}\end{array}$$

amounts to having to prove that the speed bugs resulting from any given Smm list will not overlap. This is interesting since it seems to be imposing a condition on the abstract state, which is derived from the perceptual level. However, because ScaleFactor and BugExtent are not known at the abstract level, this restriction cannot be expressed with an invariant of AbstractASI, as above. Instead, it must be included in the model as a pre-condition to ρ (see the definitions of pre_rho and pre_rho_BugSeq in Appendix B). This pre-condition can be interpreted as saying that any given representation will have limitations regarding the range of information it can present. In this case, BugExtent must be such that allows all relevant lists of minimum manoeuverability speeds to be represented.

These two examples show how formalising the different models in a theorem prover helps in guaranteeing consistency between the models, and in finding potential sources of problems with the proposed design. With these two additions the theories now type check without problems.

3.4. Proving equivalence

We will now attempt to prove the equivalence of the logical and perceptual operators for the configuration change task. The conjecture that represents the equivalence is introduced in theory ASIequivalence (see figure 4). The first sequent for this conjecture is:

Sequent 3. configuration_change_task:

$$\begin{array}{ll} \hline \{1\} & \forall\ (\mathrm{abs\_asi}:\ \mathrm{AbstractASI}): \\ & \quad \mathrm{configChangeCheck}(\mathrm{abs\_asi}) = \mathrm{asiConfigCheck}(\rho(\mathrm{abs\_asi})) \end{array}$$

We start the proof by skolemising and expanding definitions. Eventually we reach a point where, after introducing the definition of next_counterclockwise, the proof splits into two subgoals: *configuration_change_task.1* and *configuration_change_task.2*. (A schematic diagram of the proof is shown in figure 5).

Proceeding with the first subgoal, after expanding definitions and some arithmetic simplifications, we arrive at the following sequent, where ccbugindex is a skolem constant representing the index of the first bug just below the needle (which we interpret in configChangeCheck as representing the current configuration):

Sequent 4. configuration_change_task.1:

$$\begin{array}{ll} \{\text{-}1\} & \mathrm{posn}(\mathrm{rho\_BugSeq}(\mathrm{Smm}(\mathrm{abs\_asi'}))(\mathrm{ccbugindex})) \leq \\ & \quad \mathrm{posn}(\mathrm{rho\_Needle}(\mathrm{Vc}(\mathrm{abs\_asi'}))) \\ \{\text{-}2\} & \forall\ (j:\ \mathrm{nat}): \\ & \quad j\ <\ \mathrm{ccbugindex} \Rightarrow \\ & \qquad \mathrm{posn}(\mathrm{rho\_BugSeq}(\mathrm{Smm}(\mathrm{abs\_asi'}))(j)) \\ & \qquad\qquad >\ \mathrm{posn}(\mathrm{rho\_Needle}(\mathrm{Vc}(\mathrm{abs\_asi'}))) \\ \hline \{1\} & \mathrm{Vc}(\mathrm{abs\_asi'}) \leq \mathrm{Smm}(\mathrm{abs\_asi'})(\mathrm{Cc}(\mathrm{abs\_asi'}))\ +\ \mathrm{Smargin} = \\ & \quad \mathrm{configBugCheck}(\mathrm{rho\_Needle}(\mathrm{Vc}(\mathrm{abs\_asi'})), \\ & \qquad \mathrm{rho\_BugSeq}(\mathrm{Smm}(\mathrm{abs\_asi'}))(\mathrm{ccbugindex})) \end{array}$$

The sequent can be read as: if *needle* points to a velocity above or equal to the bug *ccbugindex* (antecedent -1), and all bugs above *ccbugindex* are also above the needle (antecedent -2), then, testing that the current (abstract) velocity $Vc(abs\_asi')$ is below the minimum manoeuvering speed of the current configuration $Smm(abs\_asi')(Cc(abs\_asi'))$ plus the safe margin, yields the same result as performing a *configBugCheck* on the needle and the bug just below the needle.

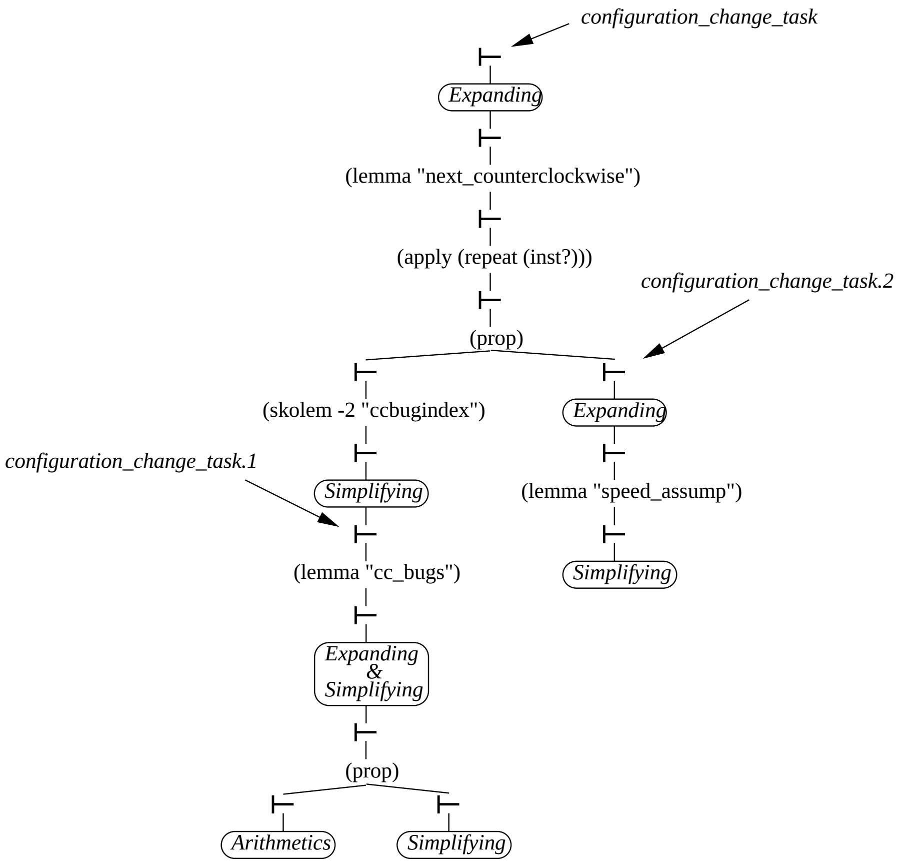

Fig. 5. Schematic proof tree for configuration_change_task

This is not unexpected and should be true, as we interpret the bug just below the needle as indicating the current configuration. To prove it, we must be able to establish that $\mathrm{Cc}(\mathrm{abs\_asi}')$ (the configuration index at the abstract level) and ccbugindex (the configuration index at the perceptual level) point to the same minimum manoeuvering speed.

3.4.1. Assumption 1

Thus, by following a methodical approach to the analysis of a specification, we raise one of the representational issues which came out of the post-hoc observation of the system in use - that the bug just below the needle indicates the current configuration. The effect of this assumption is to assign a meaning (in terms of configuration) to regions of the airspeed indicator face - a vital representational property that Hutchins dwells on in some detail [Hut95b]. Particularly, he notes:

> Once the bugs have been set, the pilots do not simply take in sensory data from the ASI; rather, the pilots impose additional meaningful structure on the image of the ASI. They use the tags to define regions on the face of the ASI, and they associate particular meanings with those regions.

To proceed with the proof, we formalise the assumption with the following axiom:

$$
\begin{array}{l}
\text{cc\_bugs}: \ \textsc{axiom} \\
\quad \forall \ (i: \text{nat}): \\
\qquad ((\text{posn}(\text{bugs}(\text{asi})(i)) \times \text{ScaleFactor} \leq \text{Vc}(\text{abs\_asi})) \wedge \\
\qquad\quad (\neg\exists \ (j: \text{nat}): \\
\qquad\qquad j \ < \ i \wedge \\
\qquad\qquad\ (\text{posn}(\text{bugs}(\text{asi})(j)) \times \text{ScaleFactor} \leq \\
\qquad\qquad\quad \text{Vc}(\text{abs\_asi})))) \Rightarrow i = \text{Cc}(\text{abs\_asi})
\end{array}
$$

By forcing us to add this relation between the needle and the bugs to the perceptual model, the proof process is unveiling relationships between the different components of the presentation which were not initially considered. In this case it is pointing out that pilots will use bugs in unforeseen ways, in order to detect the current configuration of the airplane. The possibility of being able to predict how users might interpret the presentation is valuable to the design process. In particular we might ask whether the bugs are a suitable representation for the current configuration. Next, we will see how considerations about this also arise from the proof.

After applying the axiom, we proceed once again by expanding definitions and simplifying. Eventually, the subgoal is further subdivided into two subgoals, both of which are easy to prove. We are left with subgoal *configuration_change_task.2*. This subgoal is represented by the sequent:

Sequent 5. configuration_change_task.2:

$$
\begin{array}{ll}
\hline
\{1\} & (\exists \ (i: \text{nat}): \\
 & \quad (\text{Smm}(\text{abs\_asi}')(i) \ / \ \text{ScaleFactor} \leq \text{Vc}(\text{abs\_asi}') \ / \ \text{ScaleFactor}))
\end{array}
$$

3.4.2. Assumption 2

This condition is generated by the precondition to *next_counterclockwise*. Combining this with the axiom *cc_bugs* yields a second assumption about the system. As failing to change configuration before dropping below the minimum manoeuvering is a critical item, it is generally assumed that this is never the case. Without knowledge of this constraint, we would have to conclude that the presentation is inadequate for representing configuration. Formalising the assumption, we introduce the following axiom for ASI:

$$\mathrm{speed\_assump}: \ \textsc{axiom}\ \forall(\mathrm{abs\_asi}:\mathrm{ASI}):\mathrm{Vc}(\mathrm{abs\_asi}) \geq \mathrm{Smm}(\mathrm{abs\_asi})(\mathrm{Cc}(\mathrm{abs\_asi}))$$

A very interesting aspect of this assumption is that it is an *operational* constraint (that the pilot *must* keep the aircraft in an appropriate configuration for the speed), on which the success of the representation (that it reflects the current configuration) is based, and further illustrates how abstract and representational properties are intertwined. Being operational in nature, it prompts us to analyse how realistic it is to assume that it will hold, and whether the proposed presentation should be changed or used in conjunction with some other perceptual artifact explicitly representing the current configuration. We might even decide to go back to the abstract state and make a change there (for instance, introducing interlocks). We see here how the proof process prompts us to reason about the artifact in the context of the overall design of the cockpit.

Using the above invariant the proof for this subgoal can be finished. The proof tree for the final proof is shown in figure 5.

3.5. Summary

Through the verification process, we identified significant changes, in the form of assumptions about the system, to the specification. Each of these assumptions is derived directly from the information provided by the proof process.

The first highlighted aspects of the abstract level that needed to be better represented at the perceptual level. The relationship between bugs and current configuration had to be made explicit for the configuration change task to be supported. The second brought out the implications of some of the assumptions we were making about the interface and showed us that those issues had not been included in our abstract model. Each of the assumptions added to enable completion of the proof had a representational basis and concerned vital properties of the presentation. We can place these assumptions in the context of the system, leading to an altered view of the analysis, depicted in figure 6.

In summary, checking for consistency between an abstract specification of the interactive system and the presentation for that system, allowed us not only to better understand the issues involved at the interface level and what assumptions were being made, but also to see how assumptions made at the perceptual level relate back to the system itself. Additionally, the verification process was valuable in that it allowed us to identify a number of minor bugs in both specifications. Finally, although these proofs are not so complex that it would be infeasible to do them by hand, we found the prover to be helpful in two ways:

- PVS has a number of powerful proving commands that, most of the time, can save us a lot of time and patience — as we use more and more concrete specifications of the perceptual level, so the theorem prover will become more and more useful in this regard

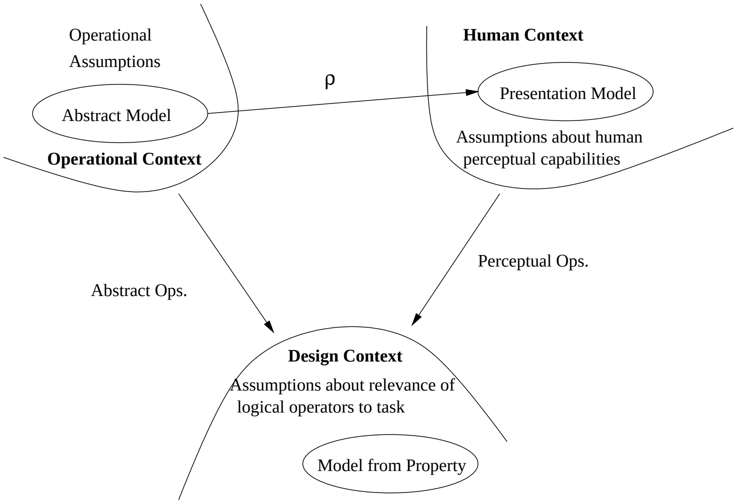

Fig. 6. Context in representational analysis

- by being *totally impartial*, the theorem prover better exposes assumptions we are making about the system. Had the verification been performed by hand, some of those assumptions might have crept into the proof unnoticed. Of course, a possibility remains that we might include invalid assumptions in our model, but such assumptions are more easily discovered and challenged when explicitly encoded in the specification, than when they are hidden among the steps of a manual proof.

A further practical issue is that of generality and reuse. We would point out that there are two levels at which reuse can take place. Firstly, the formalisations of the abstract and presentation models and associated operators can be shared over different analyses involving the same components. In the present context, the analysis will be relevant for all cockpit designs that use this type of airspeed indicator. Secondly, the same proof can be used over different versions of the same design. This is useful both when testing alternative design solutions for a given problem, and to guarantee that a new design preserves all the properties of the previous version. Additionally, it is also sometimes possible to reuse portions of a previously conducted proof within another.

4. Conclusions

The stated aims of this paper were to provide a means for integrating representational reasoning into a design process, and to explore further the verification process. Since reasoning about interaction falls at least partially in the realms of human-factors related sciences, being able to conduct such reasoning from a mathematical standpoint is an interesting result.

We have shown that by employing a *formal* model which allows us to address representational issues, we provide both a rigorous and precise framework for reasoning about representation, and confidence that reasoning over the abstract specification holds at the presentation level. The primary concern of our analysis is the *choice of representation*, it is this choice which the analysis can guide, and which we ensure is consistent with the chosen properties. The analysis requires that we know the properties to be supported and the attributes to be presented. Our treatment has led us to formulate a revised view of the analysis, in which the context, and particularly assumptions about this context are an integral part of the process.

We have also shown how the verification process improves our understanding of the specification, and in particular brings out assumptions about the system which are embedded in the representation. We consider it an interesting aspect of the process that some issues have emerged purely from the effort of formalisation, whereas others emerge only when we attempt to verify the relationship between the logical and perceptual operators.

We have to an extent also answered the question posed in section 1.1, by illustrating that we can engage in human factors reasoning based on the proof process while using automated support. An interesting question concerns the step between reaching a point in the proof which indicates some assumption is missing from the specification, and the actual assumption introduced. We can make a number of comments on this. Firstly, our reference point is the system being described; the assumptions are not arbitrary, but must make sense in the context of the system. Secondly, the process is about discovering potential problems with the system and improving our understanding of the system, rather than ‘proving’ the system to be usable. One notes also that if we desire to know if an assumption is *necessary* in the context of a property, then a proof by contradiction is possible; this is an area for further exploration.

Each of the assumptions brought out by the verification process had an important representational significance and a direct correspondence with Hutchins analysis. The first concerned the *implicit* representation of current configuration by the configuration change bugs. The second assumption exposed an operational constraint, which is not part of the abstract system model, yet which is vital for the success of the presentation. Thus it is not merely a question of errors or omissions in the specification, but additional information and understanding which emerges from the analysis.

External representations are, of course, only one aspect of the distributed cognitive system; one interesting area for future work would be to consider a wider range of cognitive resources, for example by employing the resources model of [WFH96].

Acknowledgements

José Campos was supported by FCT - Fundação para a Ciência e a Tecnologia, Portugal - under grant PRAXIS XXI/BD/9562/96. During part of the preparation of this paper Gavin Doherty was a Visiting Researcher at CNR Istituto CNUCE, Pisa, Italy, and has been supported by the TACIT network under the European Union TMR programme, contract ERB FMRX CT97 0133. The authors would like to thank the anonymous reviewers and the organisers and participants of the BCS workshop on Formal Aspects of the Human Computer Interface.

References

[Age96] S. Agerholm. Translating specifications in VDM-SL to PVS. In *Proceedings of 9th International Conference on Theorem Proving in Higher Order Logics*, volume 1125 of *Lecture Notes in Computer Science*, 1996.

[Cam00] José C. Campos. *Automated Deduction and Usability Reasoning.* DPhil thesis, Department of Computer Science, University of York, 2000.

[Cas91] S.M. Casner. A task-analytic approach to the automated design of graphic presentations. *ACM Transactions on Graphics*, 10(2):111–151, April 1991.

[CH97] J. C. Campos and M. D. Harrison. Formally verifying interactive systems: A review. In Harrison and Torres [HT97], pages 109–124.

[CH99] José C. Campos and Michael D. Harrison. Using automated reasoning in the design of an audio-visual communication system. In D.J. Duke and A. Puerta, editors, *Design, Specification and Verification of Interactive Systems '99*, pages 167–188. Springer-Verlag/Wien, 1999.

[DH93] D. J. Duke and M.D. Harrison. Abstract interaction objects. *Proceedings of Eurographics '93,* Computer Graphics Forum, 12(3), 1993.

[DH97] G. Doherty and M. D. Harrison. A representational approach to the specification of presentations. In Harrison and Torres [HT97].

[Dia89] Dan Diaper, editor. *Task Analysis for Human-Computer Interaction.* Ellis Horwood Books in Information Technology. Ellis Horwood, 1989.

[HT90] M. D. Harrison and H. W. Thimbleby, editors. *Formal methods in Human Computer Interaction.* Cambridge University Press, 1990.

[HT97] M.D. Harrison and J.C. Torres, editors. *Proceedings, 4th Eurographics Workshop on Design, Specification, and Verification of Interactive Systems*, Springer Computer Science. Springer Wien, 1997.

[Hut95a] E. Hutchins. *Cognition in the Wild.* MIT Press, 1995.

[Hut95b] E. Hutchins. How a cockpit remembers its speeds. *Cognitive Science*, 19:265–288, 1995.

[MSB95] J. May, S. Scott, and P.J. Barnard. *Structuring Displays: A Psychological Guide.* Eurographics Tutorial Notes Series. EACG: Geneva, 1995. Appeared as Amodeus report B04.

[OSR93] S. Owre, N. Shankar, and J. M. Rushby. *User Guide for the PVS Specification and Verification System.* Computer Science Laboratory, SRI Internatinal, Menlo Park CA 94025, USA, (beta release) edition, March 1993.

[PP98] P. Palanque and F. Paternó, editors. *Formal Methods in Human-Computer Interaction.* Formal Approaches to Computing and Information Technology. Springer-Verlag, 1998.

[SR96] M. Scaife and Y. Rogers. External cognition: how do graphical representations work? *International Journal of Human-Computer Studies*, 45:185–213, 1996.

[Ste46] S.S. Stevens. On the theory of scales of measurement. *Science*, 103:677–680, 1946.

[WFH96] P.C. Wright, B. Fields, and M.D. Harrison. Distributed information resources: An new approach to interaction modelling. In T.R.G. Green, J.J. Canas, and C.P. Warren, editors, *Proceedings of ECCE8: European Conference on Cognitive Ergonomics*, pages 5–10. EACE, 1996.

[WJCS94] D.D. Woods, L.J. Johannesen, R.I. Cook, and N.B. Sarter. Behind human error: Cognitive systems, computers and hindsight. State-of-the-Art Report SOAR 94-01, Crew Systems Ergonomics Information and Analysis Center (CSERIAC), Wright- Patterson Airforce Base, Ohio., December 1994.

[Zha96] J. Zhang. A representational analysis of relational information displays. *International journal of human computer studies*, 45, 1996.

[ZN94] J. Zhang and D. A. Norman. Representations in distributed cognitive tasks. *Cognitive Science*, 18:87–122, 1994.

A. ASI **PVS theory**

This appendix presents the final version of the ASI PVS theory introduced in section 3.2. This theory is the PVS equivalent to the VDM model described in section 2.4.1, with the addition of an invariant for AbstractASI, and an assumption about user behaviour.

```
ASI : THEORY
   BEGIN

% Types needed for the model

   Speed : TYPE   = real
   Speeds : TYPE   = sequence[Speed]
   Configuration : TYPE   = nat
   AbstractASIaux : TYPE   = [# Vc : Speed,
                                Cc : Configuration,
                                Smm : Speeds,
                                Vref : Speed#]

   % Invariant — Smm is sorted
   inv_abs_asi(abs_asi : AbstractASIaux) : bool  =
      ∀ (i, j : nat) : i < j ⇔ Smm(abs_asi)(i) > Smm(abs_asi)(j)

   AbstractASI : TYPE (inv_abs_asi)

% Values — the safe margin to change configuration

   Smargin : Speed

% Assumptions

   abs_asi : VAR AbstractASI

   % the speed never goes below the minimum maneuverability speed
   % for the current configuration
   speed_assump : AXIOM Vc(abs_asi) ≥ Smm(abs_asi)(Cc(abs_asi))

   % Logical operator — should configuration change?

      configChangeCheck((asi : AbstractASI)) : bool  =
                              Vc(asi) ≤ Smm(asi)(Cc(asi)) + Smargin

      END ASI
```

B. perceptualASI PVS theory

In this appendix the final version of the perceptualASI PVS theory is presented. This theory is the PVS equivalent to the VDM model described in section 2.4.2, with the addition of assumptions and pre-conditions.

```
perceptualASI : THEORY
   BEGIN

% ASI must be imported for we need Smargin

   IMPORTING ASI
```

```
% Types

  Angle : TYPE  = real
  ASINeedle : TYPE  = [# posn : Angle#]
  ASISpeedBugs : TYPE  = sequence[ASISpeedBug]
  ASIScale : TYPE  = [# interpret : [Angle → real] #]
  ASIInstrumentaux : TYPE  = [# needle : ASINeedle,
                                bugs : ASISpeedBugs,
                                scale : ASIScale#]

  % Invariant — Bugs do not overlap
  inv_ASIInstrument(asi : ASIInstrumentaux) : bool =
    ∀ (i, j : nat):
      i < j ⇒
        (posn(bugs(asi)(i)) > posn(bugs(asi)(j))∧
         posn(bugs(asi)(i)) > posn(bugs(asi)(j)) + extent(bugs(asi)(j)))

  ASIInstrument : TYPE  = (inv_ASIInstrument)

% Constant values

  ScaleFactor : posreal
  BugExtent : Angle

% Assumptions

  asi : VAR ASIInstrument
  abs_asi : VAR AbstractASI

  % Bugs identify configurations
  cc_bugs : AXIOM
    ∀ (i : nat):
      ((posn(bugs(asi)(i)) × ScaleFactor ≤ Vc(abs_asi))∧
         (¬∃ (j : nat):
           j < i∧
             (posn(bugs(asi)(j)) × ScaleFactor ≤
               Vc(abs_asi)))) ⇒
        i = Cc(abs_asi)

% Presentation mapping

  rho_Needle((v : Speed)) : ASINeedle = (#posn: = v / ScaleFactor#)

  pre_rho_BugSeq((s : Speeds)) : bool =
   ∀ (i, j : nat) : i < j ⇒ (s(i)/ScaleFactor > s(j)/ScaleFactor + BugExtent)
  rho_BugSeq((s : (pre_rho_BugSeq))) : ASISpeedBugs =
   Λ (i : nat) : (#posn: = s(i) / ScaleFactor, extent: = BugExtent#)

  rho_Scale : ASIScale = (#interpret: = Λ (a : Angle) : ScaleFactor × a#)

  pre_rho((a : AbstractASI)) : bool = pre_rho_BugSeq(Smm(a))
  ρ((a : (pre_rho))) : ASIInstrument =
   (#needle: = rho_Needle(Vc(a)),
      bugs: = rho_BugSeq(Smm(a)),
      scale: = rho_Scale#)
```

```
% Perceptual Operators

  % is the needle inside an arc?
  in_arc((needle : ASINeedle), (astart, aend : Angle)) : bool =
          astart ≤ posn(needle) ∧ posn(needle) ≤ aend

  % identify the bug below the needle (defined axiomatically)
  next_counterclockwise : [[ASINeedle, ASISpeedBugs] → ASISpeedBug]
  next_counterclockwise : AXIOM
    ∀ (needle : ASINeedle, bugs : ASISpeedBugs, bug : ASISpeedBug):
     (∃ (i : nat) : posn(bugs(i)) ≤ posn(needle)) ⇒
      (next_counterclockwise(needle, bugs) = bug ⇔
       (∃ (i : nat):
        bugs(i) = bug∧
         posn(bugs(i)) ≤ posn(needle)∧
          (∀ (j : nat):
           j < i ⇒ posn(bugs(j)) > posn(needle))))

  % identify the current speed bug
  getCurrentBug((needle : ASINeedle), (bugs : ASISpeedBugs)):
   ASISpeedBug = next_counterclockwise(needle, bugs)

  % is the needle inside the safe margin of a bug?
  configBugCheck((needle : ASINeedle), (bug : ASISpeedBug)) : bool =
   in_arc(needle, posn(bug), posn(bug) + Smargin / ScaleFactor)

  % should configuration change?
  asiConfigCheck((asi : ASIInstrument)) : bool =
   configBugCheck(needle(asi), getCurrentBug(needle(asi), bugs(asi)))

  END perceptualASI
```

C. ASIverification **PVS theory**

In this appendix the final version of the ASIverification PVS theory is presented. Note that the only difference between this theory and the original version presented in figure 4 is the use of (pre_rho) instead of abstractASI in the declaration of abs_asi. This guarantees that abs_asi will validate the precondition to ρ.

```
ASIverification : THEORY
  BEGIN

  IMPORTING ASI, perceptualASI

  abs_asi : VAR (pre_rho)

  configuration_change_task : CONJECTURE
    configChangeCheck(abs_asi) = asiConfigCheck(ρ(abs_asi))

  END ASIverification
```