Evolution of Formal Model-based Assurance Cases for Autonomous Robots

Mario Gleirscher,[*] [0000−0002−9445−6863] Simon Foster,[0000−0002−9889−9514] and Yakoub Nemouchi

Department of Computer Science, University of York, United Kingdom
`firstname.lastname@york.ac.uk`, http://www.cs.york.ac.uk

Abstract. An assurance case should carry sufficient evidence for a compelling argument that a system fulfils its guarantees under specific environmental assumptions. Assurance cases are often subject of maintenance, evolution, and reuse. In this paper, we demonstrate how evidence of an assurance case can be formalised, and how an assurance case can be refined using this formalisation to increase argument confidence and to react to changing operational needs. Moreover, we propose two argument patterns for construction and extension and we implement these patterns using the generic proof assistant Isabelle. We illustrate our approach for an autonomous mobile ground robot. Finally, we relate our approach to international standards (e.g. DO-178C, ISO 26262) recommending the delivery and maintenance of assurance cases.

Keywords: Assurance case · formal verification · refinement · autonomous robot · integrated formal methods · model-based engineering.

1 Introduction

Autonomous robots in complex multi-participant environments can engage in risky events (e.g. because of faults or partial state knowledge) possibly leading to accidents. To reduce the opportunities for all participants to engage in such events or their consequences, one wishes to observe only specific machine behaviours. *Assurance Case*s (ACs) [20] are structured arguments, supported by evidence, intended to demonstrate that such machines fulfil their *assurance guarantees* [22], subject to certain *assumptions* about their environment [18,32]. Among the wide variety of assurance objectives, we will focus on *safety* in the rest of this paper, with a careful eye on *liveness.*

Compelling ACs require models to describe the behaviour of the real-world artefacts subjected to the assurance claims, and to provide evidence for these, contingent on validation. In particular, formal methods (FMs) can be applied to the rigorous analysis of a system's state space, and to the computer-assisted verification of requirements. However, verification and validation can, in reality, fail to deliver safe systems, for example, due to an inadequate model that abstracts

[*] Supported by the German Research Foundation (DFG grant no. 381212925).

from essential detail. Such shortcomings can be difficult to identify in advance, and consequently, models and assurance cases have to *evolve* [2,7]. Particularly, one might want to modify or *extend* an existing AC, for example, weaken its assumptions, make its model more precise, or strengthen its guarantees. Several such steps might be required to arrive at an acceptable confidence level.

In their study of assurance practice, Nair et al. [27] report that *evidence completeness and change impact* is managed mostly manually using traceability matrices. These authors observe a lack of tool support for change management and that evidence structuring is done mostly textually rather than with model-based ACs. Importantly, their study raises the question of *how evolution and change impact is identified, assessed, and managed at the level of ACs*?

Contributions. We consider *formal model-based assurance cases* (FMACs) to construct assurance arguments in a rigorous and step-wise manner. We define an FMAC as an AC *module* that conveys formal verification results with respect to a model and certain environmental assumptions. Our motivation is the assurance of physical systems such as autonomous mobile robots, and so we chose to support models and requirements in differential dynamic logic ($\mathsf{d}\mathcal{L}$) [33].

We underpin our approach by AC *patterns* for the incremental construction of increasingly richer models, guarantees, and proofs that these models fulfil the guarantees. We provide two patterns, one for AC *construction* and one for AC *extension*, particularly, for increasing confidence in an AC by making the formalisation successively more precise. Both patterns provide guidance to how successive engineering steps can preserve assurance results from previous steps. We implement these patterns in Isabelle/SACM [28] and show how they can be instantiated and how assurance claims can be linked to verification results.

We complement recent approaches to robot verification (e.g. [26]) by a *data refinement* [40] for $\mathsf{d}\mathcal{L}$ that is lifted to the level of AC construction and evolution. For a mobile ground robot, we illustrate two refinement steps from a maximally abstract model to one describing safe path planning and emergency braking. We indicate how one can derive safety guarantees from hazard analysis yet avoiding too conservative solutions by adding liveness guarantees. We demonstrate our proofs in Isabelle/HOL [29] by formalising the robot model in an implementation of $\mathsf{d}\mathcal{L}$ in Isabelle/UTP [10,11].

Related Work. Bate and Kelly [2] discuss the notion of AC modules, interfaces, and their composition into an AC architecture via claim matching and *assumption/guarantee* (A/G)-style reasoning (e.g. weakening of assumptions). Following their discussion of AC change with a focus on traceability and change impact analysis, we focus on AC extension for the verified evolution of AC modules.

Prokhorova et al. [34] propose the construction of formal model-based safety cases based on a classification of safety requirements using Event-B. The authors discuss argument patterns for all classes of requirements. The patterns integrate proof and model-checking evidence[1] and cover refinement steps. Complementing

[1] From the Rodin tool and from LTL and timed CTL checkers.

their Event-B application, our approach supports hybrid system modelling. We cover their requirements classification, except for temporal and timing properties requiring binary modalities. We focus on step safety and liveness, and path safety. To support argument maintenance and scalability, we separate system modelling and proof derivation from argumentation, keeping model and proof details separate from the argument structure. This separation is facilitated in our Isabelle-based implementation by using $\mathsf{d}\mathcal{L}$ for system modelling and verification and the FMAC concept for assurance argumentation.

Oliveira et al. [30] propose hierarchical modular safety cases to reuse common product-line features in general safety arguments and to decompose and refine the general argument into feature-specific argument modules. While the authors cover hazard analysis (viz. model-based failure analysis) and product-line modelling, our notion of AC extension based on data refinement can be useful for the verified derivation of a product-specific AC from the product-line AC.

For adaptive systems, Calinescu et al. [4] elaborate on the idea of through-life argument maintenance [7, 31], focusing on the maintenance of a parametric AC whose parameters are subject of optimisation during the operation of a system. We complement their approach by a notion of data refinement to accommodate fundamental structural changes frequently desired for argument evolution.

Overview. The remainder of this article is structured as follows: We introduce the concepts in Section 2, explain our contributions in Section 3, evaluate our approach with a robot example in Section 4, discuss implications on formal robot verification and certification practice in Section 5, and conclude in Section 6.

2 Background and Formal Preliminaries

We introduce assurance cases from a practical viewpoint and provide the preliminaries on system specification and verification.

2.1 Assurance Cases

An AC is a compelling[2] argument, supported by evidence, that a system in a specific context fulfils (or refuses to fulfil) guarantees of interest, for example, freedom of hazards (*safety*), sustained correct service (*reliability*), freedom of unauthorised access (*security*), or productivity (*performance*). Intuitively, an AC is a hierarchical structure, with *claims* that are broken down into subclaims using argumentation *strategies*, and referencing an appropriate *context*, such as system element descriptions and environmental assumptions. An AC is deemed to be "finished" when all leaf claims are supported by adequate *evidence*, though there is always the possibility of evolution.

We consider ACs as formalised in the *Structured Assurance Case Metamodel* (SACM), an OMG standard.[3] Wei et al. [39] summarise the work around SACM

[2] Usually structured, balanced, and exhibiting many further argumentation qualities.
[3] See https://www.omg.org/spec/SACM/2.0/.

and demonstrate how established frameworks like the *Goal Structuring Notation* (GSN) [19] and *Claims Arguments Evidence* (CAE)[4] can be represented using SACM. SACM thus connects users of these techniques with rigorous model-based AC construction. SACM can be characterised by three principal concepts:

1. **arguments**, that present the claims and inferential links between them;
2. **artifacts**, evidence to support leaf claims, and the relations between them. Examples include outputs of hazard analysis, actors, test reports, system data sheets, formal models, and verification results. An AC whose evidence is based on results obtained from analysis of system models, such as formal verification, is called a *model-based assurance case* [16, 39];
3. **terminology**, to support controlled languages for expressing claims, that are otherwise specified using free-form natural language. Often, these are used to refer to model elements in model-based ACs;

In AC *modules*, certain top-level claims and artefacts can be made public by an A/G-style AC *interface*. Several modules can then be composed to produce the overall AC. Claims can be *supported* by an argument within the module or *assumed* to hold of the *context*. In the latter case, corresponding external arguments have to be imported from other AC modules. This can be achieved by A/G reasoning, as is present in the design-by-contract paradigm [25]. Additionally, AC modules adhere to the standardised SACM package concept.

AC modules often need to evolve, for example, because of updates of the system design or the hazard list. Such evolutions should be conservative, in that existing claims should remain supported, assumptions should remain satisfiable, and terminology should stay consistent. This need motivates our notion of AC *extension*, the key contribution of this paper, fostering step-wise development and evolution of ACs. For this, we further develop Isabelle/SACM, our implementation of SACM as an interactive DSL for AC construction in the proof assistant Isabelle. Isabelle/SACM extends the document model Isabelle/DOF [3] to accommodate AC concepts and to provide well-formedness checking for ACs. Isabelle/SACM allows us to describe ACs with claims and evidence obtained from various formal methods. Details on Isabelle/SACM are explained in [28].

2.2 Isabelle/UTP and Differential Dynamic Logic

The evidence for an FMAC is obtained by formal verification using an implementation of $\mathsf{d}\mathcal{L}$ [33] in our verification framework, Isabelle/UTP [10, 11]. $\mathsf{d}\mathcal{L}$ specialises Dynamic Logic by combining a modelling notation for hybrid systems, called *hybrid programs*, with a formal property language for reasoning about such programs. In a hybrid program, we can use operators like sequential composition, assignment, branches and iteration, and an operator for specifying systems of ordinary differential equations (ODEs). It can therefore be used to represent hybrid systems that combine continuous evolution and discrete control. The property language extends predicate calculus with two modalities: $[P]\,\phi$,

[4] See https://claimsargumentsevidence.org.

which specifies that ϕ holds in every state reachable from P; and $\langle P \rangle \phi$, which specifies that there is at least one state reachable from P satisfying ϕ.

Isabelle/UTP implements Hoare and He's *Unifying Theories of Programming* (UTP) [17], a framework for development of semantic models for programming and modelling languages based on heterogeneous paradigms using an alphabetised relational calculus. Isabelle/UTP develops this idea by allowing UTP semantic models to be adapted into verification tools, such as Hoare calculus deductive reasoning. Then, we can harness the array of automated proof techniques in Isabelle/HOL [29], such as integrated automated theorem provers, to discharge resulting verification conditions. We apply this approach to develop the $\mathsf{d}\mathcal{L}$ hybrid program model, and the associated proof calculus as a set of derived theorems. Moreover, we have developed a tactic, ***wp-tac***, which calculates $[P]\,\phi$ and $\langle P \rangle \phi$ conditions using Isabelle's simplifier and thus automates proof.

3 Formal Model-based Assurance Cases

In this section, we develop FMACs, that is ACs that contain a formal model from which evidence for the top-level claims is derived. The informal structure of an FMAC is provided through Isabelle/SACM. We formalise claims using the modalities from $\mathsf{d}\mathcal{L}$, which allows us to formulate LTL-style guarantees of the form $p \Rightarrow \circ q, p \Rightarrow \Diamond q$, and $p \Rightarrow \Box q$. This integration of dynamic and temporal logic supports the objective underlying many ACs, that is, to integrate evidence from different provenance. We develop a formal notion of FMAC *extension*, which employs both A/G reasoning and *data refinement* [40], which allows us to elaborate models in a style similar to Event-B refinement [34].

3.1 Assurance Case Construction

In this section, we introduce a generalised model of $\mathsf{d}\mathcal{L}$-style hybrid programs, use these to define the notion of a Cyber-Physical Machine (CPM), and then define FMACs, which assure properties of a CPM using formal verification.

Hybrid programs are defined with respect to an alphabet, $\mathcal{A}$, of typed state variable declarations $(x : t)$, whose names are drawn from the set $\mathcal{V}$. $\mathcal{A}$ induces a state space Σ, and hybrid programs are modelled as potentially heterogeneous alphabetised relations over state spaces, that is, subsets of $\Sigma_1 \times \Sigma_2$. We give the following syntax for such relations.

Definition 1 (Generalised Hybrid Programs).

$$\begin{aligned} \mathcal{P} &::= \mathcal{P} \mathbin{;} \mathcal{P} \mid \mathcal{P} \sqcap \mathcal{P} \mid \mathcal{P}^{*} \mid ?\mathcal{E} \mid \langle \mathcal{S} \rangle \mid \mathcal{V} := * \mid \{\mathcal{S} \mid \mathcal{E}\} \\ \mathcal{S} &::= \boldsymbol{id} \mid \boldsymbol{nil} \mid \mathcal{S}(\mathcal{V} \mapsto \mathcal{E}) \end{aligned}$$

Here, $\mathcal{E}$ gives syntax for expressions over $\mathcal{A}$. Hybrid programs, $\mathcal{P}$, are composed using sequential composition $(P \mathbin{;} Q)$, nondeterministic choice $(P \sqcap Q)$, Kleene star (P^{*}), conditional tests $(?b)$, assignments $(\langle \sigma \rangle)$, nondeterministic assignments $(x := *)$, and ODEs $(\{\sigma \mid b\})$. Each of these operators is semantically denoted as a relational predicate (for details see [10,11]). As usual in UTP [17], relations are partially ordered by refinement ($P \sqsubseteq Q$), which corresponds to universally closed reverse implication. Most of the operators follow the $\mathsf{d}\mathcal{L}$ hybrid program notation, the exceptions being assignments and ODEs, whose generalisations help support data refinement.

Generalised assignment, $\langle \sigma \rangle$, uses a *substitution*, σ: a potentially heterogeneous total function between state spaces, $\Sigma_1 \to \Sigma_2$. The basic substitution $\textbf{\textit{id}} : \Sigma \to \Sigma$ maps every variable to its present value. Then, $\langle \textbf{\textit{id}} \rangle$ is the ineffectual program (***skip***). Moreover, $\textbf{\textit{nil}} : \Sigma_1 \to \Sigma_2$ is a heterogeneous substitution that assigns arbitrary values to every variable, ignoring the initial state.

An existing substitution can be *updated* with a maplet $x \mapsto e$, assuming x and e have the same type. We then use the notation $[x_1 \mapsto e_1, \cdots, x_n \mapsto e_n]$ to denote $\textbf{\textit{id}}(x_1 \mapsto e_1, \cdots, x_n \mapsto e_n)$, that is, the substitution that assigns n expressions to n variables, whilst leaving all other variables in the alphabet unchanged. Then, the usual singleton assignment $x := e$ can be represented as $\langle [x \mapsto e] \rangle$. Similarly, the notation $(\!| x_1 \mapsto e_1, \cdots |\!)$ constructs a heterogeneous substitution where x_1 and e_1 are from different state spaces. Moreover, substitutions can be applied to expressions using $\sigma \dagger e$, which substitutes all variables in e with those specified in σ. ODEs, $\{\sigma \mid b\}$, are modelled similarly but here σ represents the mapping of variables to their derivatives, and b is a boundary condition, as in $\mathsf{d}\mathcal{L}$.

In our model of hybrid programs, we define the modalities $\langle P \rangle \phi$ and $[P]\,\phi$ from $\mathsf{d}\mathcal{L}$ using the corresponding UTP definitions for weakest precondition (***wp***) and weakest liberal precondition (***wlp***) [17], respectively:

Definition 2 (Modalities). $\langle P \rangle \phi \triangleq (\exists\, \textbf{\textit{v}}' \bullet P \fatsemi ?\phi) \qquad [P]\,\phi \triangleq \neg \langle P \rangle (\neg \phi)$

Here, $\textbf{\textit{v}}'$ refers to the final value of the state. Thus, $\langle P \rangle \phi$ is the relational preimage of P under ϕ, and $[P]\,\phi$ is its dual defined by conjugation. From these definitions the usual laws of $\mathsf{d}\mathcal{L}$ can be proved as theorems. We use hybrid programs to represent CPMs, whose form is inspired by Parnas' four-variable model [32]:

Definition 3. *A CPM is a tuple $\mathcal{M} = (\mathcal{A}, \mathcal{I}, \mathit{Inv}, \mathcal{T})$ where*

- *$\mathcal{A}$ is an alphabet formalising the state space, which is divided into disjoint regions for controlled (**ctrl**), monitored (**mon**), and internal variables (**st**);*
- *$\mathcal{I} \subseteq \mathcal{A}$ is an initialiser that assigns initial values to state variables;*
- *$\mathit{Inv} \subseteq \mathcal{A}$ is an invariant predicate over **st** and **ctrl**;*
- *$\mathcal{T} \subseteq \mathcal{A} \times \mathcal{A}$ is the machine's transition relation.*

To reduce dependencies on the environment, we chose to not allow Inv to use monitored variables. The transition relation specifies the steps the machine can take, and is formulated using hybrid programs of the form

$$\mathcal{T} = (?g_1 \fatsemi P_1 \sqcap ?g_2 \fatsemi P_2 \sqcap \cdots \sqcap ?g_n \fatsemi P_n)$$

which corresponds to a set of non-deterministic guarded commands ($g_i \to P_i$). Then, a CPM behaves like a cyclic executive that reads monitored variables (***mon***), executes the transition relation ($\mathcal{T}$), and writes controlled variables (***ctrl***), in the style of Parnas [32]. We impose the following validity constraints on CPMs:

Definition 4. *A CPM is valid if the following conditions hold:*

1. $\mathcal{I} \cap \mathit{Inv} \neq \emptyset$ *— there is a valid initial state satisfying the invariant;*
2. $\mathit{Inv} \Rightarrow [\mathcal{T}]\, \mathit{Inv}$ *— the invariant is maintained by all transitions;*
3. $\mathit{Inv} \Rightarrow \langle\mathcal{T}\rangle \mathit{true}$ *— if the invariant holds, there is an enabled transition;*
4. $\forall\, r \bullet \langle\mathcal{T}\rangle r \Rightarrow (\exists(\textbf{ctrl}, \textbf{st}) \bullet r)$ *— only controlled and state variables are changed by the body; any predicate r, refering to* ***mon*** *only, is invariant.*

The conditions together ensure the machine is well-formed, maintains the invariant, and is free of deadlock. We can now use CPMs to define FMACs:

Definition 5. *An FMAC is a tuple $AC = (\mathcal{M}, As, Gr)$ with*

- *a valid cyber-physical machine ($\mathcal{M}$) describing the system behaviours;*
- *a set of environmental assumptions (As), specified as predicates on* ***mon****;*
- *a set of guarantees (Gr), specified as predicates on* ***mon****,* ***ctrl****,* ***st****.*

The assumption As constrains the environment with a predicate on the monitored variables. The guarantee predicates are LTL formulas corresponding to a subset of $\mathsf{d}\mathcal{L}$ formulae, namely:

- $p \Rightarrow \circ q$: if p holds currently, then q holds in the next state;
- $p \Rightarrow \Box q$: if p holds, then q holds in all subsequent states;
- $p \Rightarrow \Diamond q$: if p holds, then q holds in at least one subsequent state.

Below, Gr^s denotes a set of *(s)afety* predicates of the kind $p \Rightarrow \circ q$ and $p \Rightarrow \Box q$, and Gr^l a set of *(l)iveness* predicates ($p \Rightarrow \Diamond q$). In Section 4, we use this convention to identify corresponding predicates. Next, we define a satisfaction relation $\mathcal{M} \models \phi$ (spoken: "the machine $\mathcal{M}$ satisfies the formula ϕ").

Definition 6 (Satisfaction Relation).

$$\mathcal{M} \models (p \Rightarrow \circ q) \triangleq (As \wedge \mathit{Inv} \wedge p \Rightarrow [\mathcal{T}]\, q)$$

$$\mathcal{M} \models (p \Rightarrow \Box q) \triangleq (\exists\, I \bullet (I \Rightarrow [\mathcal{T}]\, I) \wedge (As \wedge \mathit{Inv} \wedge p \Rightarrow I) \wedge (As \wedge \mathit{Inv} \wedge I \Rightarrow q))$$

$$\mathcal{M} \models (p \Rightarrow \Diamond q) \triangleq (As \wedge \mathit{Inv} \wedge p \Rightarrow \langle\mathcal{T}^*\rangle q)$$

$\mathcal{M}$ satisfies $p \Rightarrow \circ q$ when ***wlp*** of $\mathcal{T}$ under q—i.e., the set of states from which $\mathcal{T}$ leads to a state satisfying q or is undefined—is implied by p. Similarly, $\mathcal{M}$ satisfies $p \Rightarrow \Diamond q$ when ***wp*** of $\mathcal{T}^*$ under q is implied by p. For universal properties, our definition requires an invariant. $\mathcal{M}$ satisfies $p \Rightarrow \Box q$ if there is an expression I such that (1) I is an invariant of $\mathcal{T}$; (2) I is implied by $As \wedge \mathit{Inv} \wedge p$; and (3) I, conjoined with As and Inv, implies q. From this definition, we obtain a property similar to the other definitions as a theorem:

Theorem 1 (*-Global). *If $M \models (p \Rightarrow \Box q)$ then $As \wedge \mathit{Inv} \wedge p \Rightarrow \left[\mathcal{T}^*\right] q$.*

Finally, we define a notion of validity for FMACs themselves:

Definition 7 (Validity). *FMAC is valid if $\mathcal{M}$ is a valid CPM, and all guarantees are satisfied, that is, $\forall\, g \in Gr \bullet \mathcal{M} \models g$.*

With a formal definition of FMACs in Isabelle, we can now show how this information is presented in an AC module. A GSN diagram visualising the SACM pattern for an FMAC is shown in Figure 1. This pattern refers to the CPM model (Definition 3), with its state space, invariant, and transition relation. The AC module has a top-level claim of relative safety with respect to the model, $\mathcal{M}$. It requires a set of hazards, an assumption, and a set of guarantees that mitigate the hazards. The main claims, $C1$–$C4$, are made public (indicated by the folder icon), so they can be used as components in another AC. As indicated by the diamonds, the reasoning for hazard mitigation and guarantee satisfaction is left to be developed as part of the instantiation of the pattern.

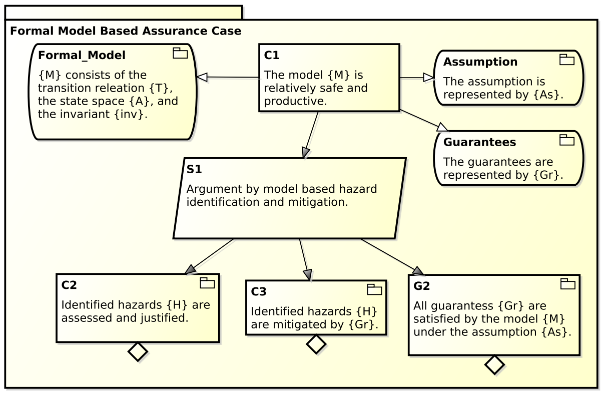

Formal Model Based Assurance Case
Formal_Model
{M} consists of the transition releation {T}, the state space {A}, and the invariant {inv}.
C1
The model {M} is relatively safe and productive.
Assumption
The assumption is represented by {As}.
Guarantees
The guarantees are represented by {Gr}.
S1
Argument by model based hazard identification and mitigation.
C2
Identified hazards {H} are assessed and justified.
C3
Identified hazards {H} are mitigated by {Gr}.
G2
All guarantess {Gr} are satisfied by the model {M} under the assumption {As}.

Fig. 1. FMAC pattern

3.2 Assurance Case Extension

FMAC extension allows us to extend an existing AC by refining the CPM model, weakening the assumptions, and adding new guarantees. In this way, the guarantees of the original FMAC can be carried over from the old to the new AC. For this, we define a notion of machine refinement.

Definition 8. *A machine refinement is a triple $(\mathcal{M}_a, \mathcal{M}_c, \rho)$, for retrieve function $\rho\colon \Sigma_c \to \Sigma_a$, such that the following conditions hold:*

1. $Inv_c \Rightarrow (\rho \dagger Inv_a)$ *— the abstract invariant is strengthened by the concrete invariant;*
2. $(?Inv_c \fatsemi \langle\rho\rangle \fatsemi \mathcal{T}_a) \sqsubseteq (?Inv_c \fatsemi \mathcal{T}_c \fatsemi \langle\rho\rangle)$*—when the concrete invariant holds initially, each transition in the abstract machine can be matched by a transition in the concrete machine (simulation [40]).*

We write $\mathcal{M}_a \sqsubseteq^s_\rho \mathcal{M}_c$ when $(\mathcal{M}_a, \mathcal{M}_c, \rho)$ is a machine refinement.

Typically, ρ shows how the variables of $\mathcal{A}_a$ are defined in terms of the variables of $\mathcal{A}_c$, with the following form:

$$\rho \triangleq (\!| x_1 \mapsto e_1(y_1, \cdots, y_n), x_2 \mapsto e_2(y_1, \cdots, y_n), \cdots |\!), \text{ for } x_i \in \mathcal{A}_a \text{ and } y_i \in \mathcal{A}_c$$

Each abstract variable is mapped to an expression e_i in terms of the concrete variables. Definition 8 encodes a backwards functional refinement [40] between

$\mathcal{M}_a$ and $\mathcal{M}_c$. We require that (1) Inv_a is strengthened by Inv_c, when the retrieve function ρ is applied; and (2) $\mathcal{T}_a$ is simulated by $\mathcal{T}_c$ modulo ρ, which is expressed using a refinement statement of the usual form [40].

From Definition 8, we prove the following theorem about safety invariants:

Theorem 2. *If $\mathcal{M}_a \sqsubseteq^s_\rho \mathcal{M}_c$ and $(Inv_a \wedge \phi) \Rightarrow [\mathcal{T}_a]\,\phi$, that is ϕ is an invariant of $\mathcal{M}_a$, then it follows that $(Inv_c \wedge \rho \dagger \phi) \Rightarrow [\mathcal{T}_c]\,(\rho \dagger \phi)$, where $\rho \dagger \phi$ is the retrieve function ρ applied as a substitution to ϕ.*

This theorem shows that any invariant of the abstract CPM is also an invariant of the concrete CPM, modulo ρ. Consequently, we now have a method for adapting safety guarantees from an abstract to a concrete assurance case via data refinement. We now use this to define the extension operator for FMACs.

Definition 9. *Given AC_a and AC_c according to Definition 5, then we define $AC_a \oplus_\rho AC_c \triangleq (\mathcal{M}_c, As_c, Gr_c \cup \{r \uparrow_\rho \mid r \in Gr^s_a\})$ where*

$$(p \Rightarrow \circ q)\uparrow_\rho \triangleq ((\rho \dagger p) \Rightarrow \circ(\rho \dagger q)) \qquad (p \Rightarrow \Box q)\uparrow_\rho \triangleq ((\rho \dagger p) \Rightarrow \Box(\rho \dagger q))$$

In an AC extension, every abstract safety guarantee is lifted to a concrete guarantee through the retrieve function. By applying ρ as a substitution, we compute the meaning of each of the safety guarantees in the refined state space. We do not map $\Diamond q$ guarantees, as these are not in general preserved by refinement. Refinements allow one to restrict behaviours to specific trace subsets. Traces establishing liveness guarantees might get excluded while meeting invariants and safety guarantees. Here, we leave liveness guarantees to be translated manually from Gr_a to Gr_c. Finally, we demonstrate when an AC extension is valid:

Theorem 3. *$AC_a \oplus_\rho AC_c$ is a valid FMAC provided that:*

1. *$\mathcal{M}_a$ and $\mathcal{M}_c$ are both valid CPMs;*
2. *AC_a is a valid FMAC;*
3. *$\mathcal{M}_a \sqsubseteq^s_\rho \mathcal{M}_c$ — machine refinement holds;*
4. *$(\rho \dagger As_a) \Rightarrow As_c$ — the assumption is weakened modulo ρ;*
5. *$\forall\, g \in Gr_c \bullet \mathcal{M}_c \models g$ — all additional guarantees are satisfied.*

This theorem shows that the existing safety guarantees can be verified with respect to the refined model. Essentially, Definition 7 is met because (1) any invariant can be transferred from abstract to concrete (Theorem 2); and (2) satisfaction of $\Box q$ properties requires an explicit invariant (Definition 6).

Figure 2 summarises the formal relationships between the artefacts of the extension argument. We then claim that AC_c extends AC_a modulo ρ. The following steps are carried through in Isabelle/UTP for each extension of AC_a:

1. We define the retrieve function ρ.
2. We prove that the concrete assumptions weaken the abstract assumptions translated using ρ, that is, $As_c \Leftarrow (\rho \dagger As_a)$.
3. By establishing the refinement $\mathcal{M}_a \sqsubseteq^s_\rho \mathcal{M}_c$, we ensure that $\mathcal{M}_c$ preserves all safety guarantees in Gr_a modulo ρ.

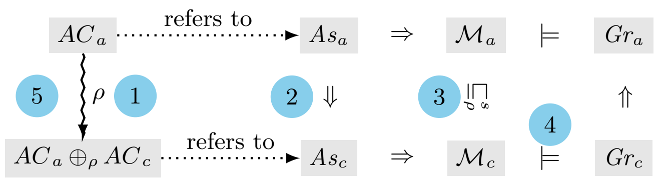
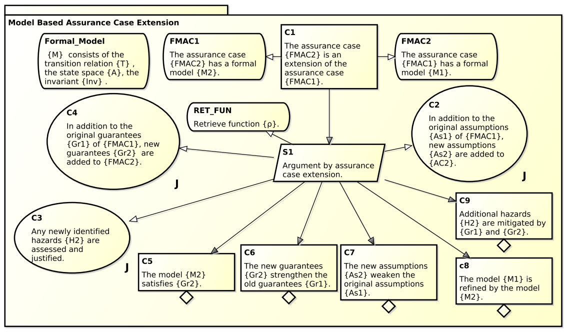


Fig. 2. Artefacts and satisfaction relationships of an extension step

4. We establish the satisfaction relationship $Gr_c \mathrel{=\!\!|} \mathcal{M}_c$ to verify all safety and liveness guarantees introduced by the extension AC_c.[5]
5. By help of ρ, the steps 2 to 4 establish the *extension* $AC_a \oplus_\rho AC_c$ representing the extended assurance case AC_c.

Figure 3 summarises the extension argument pattern. The FMAC extension readjusts or increases *confidence* over AC_a by following three principles:

1. Regarding the existing assumptions and guarantees, and modulo ρ, the claims **C5** and **C8** establish *consistency* of AC_c with the previous AC_a and thus help to *preserve* argument confidence.
2. Based on **RET_FUN**, claim **C6** aims at *increased precision*, that is, any strict data refinement of the existing alphabet, the guarantees (i.e., existing hazards), and the transition relation increases argument confidence.
3. The claims **C6** and **C7** aim at *completion*, that is, any strict extension of the set of hazards and, potentially, guarantees, while *not* strengthening the assumptions, increases argument confidence.



Model Based Assurance Case Extension
Formal_Model
{M} consists of the transition relation {T} , the state space {A}, the invariant {Inv} .
FMAC1
The assurance case {FMAC2} has a formal model {M2}.
C1
The assurance case {FMAC2} is an extension of the assurance case {FMAC1}.
FMAC2
The assurance case {FMAC1} has a formal model {M1}.
C4
In addition to the original guarantees {Gr1} of {FMAC1}, new guarantees {Gr2} are added to {FMAC2}.
J
RET_FUN
Retrieve function {ρ}.
C2
In addition to the original assumptions {As1} of {FMAC1}, new assumptions {As2} are added to {AC2}.
J
S1
Argument by assurance case extension.
C3
Any newly identified hazards {H2} are assessed and justified.
J
C9
Additional hazards {H2} are mitigated by {Gr1} and {Gr2}.
C5
The model {M2} satisfies {Gr2}.
C6
The new guarantees {Gr2} strengthen the old guarantees {Gr1}.
C7
The new assumptions {As2} weaken the original assumptions {As1}.
c8
The model {M1} is refined by the model {M2}.

Fig. 3. GSN version of the FMAC extension pattern

We implemented the FMAC patterns in Isabelle/SACM, particularly, their argument structure and the linking of claims with artefacts in Isabelle/UTP (Section 2.1). Unlike GSN, SACM allows us to structure contextual elements, e.g. one can express inferential links between **Formal_Model** (Figure 1) and its components *As*, *Gr*, and *Inv*. This way, SACM exceeds GSN's expressiveness. Our implementation in Isabelle/SACM is described in more detail in [13].

[5] The steps 3 and 4 imply that the concrete safety guarantees strengthen the abstract safety guarantees, that is, $Gr_c^s \Rightarrow (\rho \dagger Gr_a^s)$.

Table 1. Overview of the guarantees for the initial assurance case and its extensions

AC	Guar.	Informal Description
AC_0	Gr_0^s	The robot shall not cause harm to anything.
	Gr_0^l	The robot shall be able to perform movements.
AC_1	Gr_1^s	a) While the robot traverses a location, no other objects occupy this location. b) The robot only traverses locations that are not occupied by other objects.
	Gr_1^l	If there are free locations other than the robot's current location, the robot immediately leaves its current location and moves to another free location.
AC_2	$Gr_{2.1}^s$	The route planned to an intermediate location does not overlap with the detected occupancy of the workspace by other (fixed and moving) objects.
	$Gr_{2.2}^s$	If overlaps are detected while moving the robot intervenes by rerouting and/or by braking.
	Gr_2^l	The robot eventually reaches its goal (a prespecified location) given that it is reachable via a finite sequence of routes/path segments.

4 Application to Mobile Ground Robot

Mobile ground robots are required to achieve a variety of tasks by moving through a workspace while manipulating and transferring objects. Such robots are used for transport in warehouses or hospitals, for cleaning in buildings, for manufacturing in factories. In this paper, we focus on safety and liveness of the *movement* part in such tasks. Hence, our FMAC will argue about safe steps of movement, route planning with obstacle avoidance, and emergency braking.

We now instantiate the FMAC patterns from Section 3 to get three successively more detailed assurance cases of such a robot, called AC_0, AC_1, and AC_2. Table 1 summarises the guarantees for the corresponding evolution steps.

4.1 AC_0: Initial Assurance Case

In $\mathcal{M}_0$, we only consider the propositional variable *harm* which is true if any harm to any asset has occurred *because* of the robot's behaviour.

$$\begin{aligned}\mathcal{I}_0 &\triangleq [harm \mapsto \texttt{false}]\\ Inv_0 &\triangleq Gr_0 \triangleq \neg harm\\ \mathcal{T}_0 &\triangleq Move \triangleq ?\texttt{true} \mathbin{;} \textbf{\textit{skip}}\end{aligned}$$

Because we do not have any elements in the model to express bad things, *Move* has to cover only and exactly the safe movements the robot can make in order to fulfil Gr_0. If *harm* is `true` then it is not because of *Move*. Hence, it must have been `true` before but this contradicts our assumption. Although the latter technically provides an argument for safety, clearly, this argument is not anywhere near a compelling or meaningful one. Thus, we have to increase confidence.

4.2 AC_1: First Extension

Given *LOCATION* as the non-empty set of locations, $p\colon LOCATION$ denotes the current position of the robot, and $aim\colon LOCATION$ the current choice of the next location to move to. The operation *Move* now gets the atomic assignment $p := aim$. Furthermore, in $\mathcal{M}_1$, we include a state function $occ\colon LOCATION \Rightarrow bool$ which is `true` for all locations occupied by objects other than the robot, `false` otherwise. For a weak notion of liveness with respect to *Move*, the robot is required to choose *aim* such that it keeps moving as long as *occ* is `false` for some location in *LOCATION*. In $\mathcal{M}_1$, the choice of *aim* is implemented by a type-safe and constrained non-deterministic assignment. The following listing summarises the FMAC consisting of the model $\mathcal{M}_1$ and the assurance case AC_1:

$$\begin{aligned}
\mathcal{I}_1 &\triangleq \textit{\textbf{id}} \\
Inv_1 &\triangleq \neg occ(p) \\
\mathcal{T}_1 \triangleq Move &\triangleq aim := * \mathbin{⨾} ?aim \in freeLocs \wedge aim \neq p \wedge \neg occ(aim) \mathbin{⨾} p := aim \\
\hline
As_1 &\triangleq \neg occ(p) \wedge freeLocs \neq \varnothing \\
Gr_1 &\triangleq ?(p = aim) \mathbin{⨾} \neg occ(aim) \\
\rho_1 &\triangleq (\!|harm \mapsto occ(p)|\!)
\end{aligned}$$

We assume that while the robot performs a *Move* to location *aim*, the environment will not occupy *aim*. Regarding the confidence of AC_1, this assumption is realistic if locations are close enough to each other and the maximum speed of other objects is low enough. With As_1, we assume that *occ* changes (i.e., other objects can be randomly (re)located in *LOCATION*) beyond that restriction only after a *Move* of the robot is completed.[6] Also, Gr_1 encodes the assumption that whenever the robot has reached *aim* in this state, no other object can have reached *aim* in the same state. Hence, a location is deemed occupied if a moving object is expected to touch this location during the current *Move* of the robot. A more detailed model for AC_1 is given in [13, Secs. 4 and 5].

Now, to argue that AC_1 is an extension of AC_0 as explained in Section 3.2, we first prove that As_0 is weakened by As_1 modulo ρ and show that the existing safety guarantees are preserved by establishing the refinement $\mathcal{M}_0 \sqsubseteq^s_\rho \mathcal{M}_1$. Finally, we show by establishing Definition 7 in Isabelle that AC_1 is valid.

Regarding the confidence of AC_1, the introduced location model, the conditional *Move*, and the occupancy-based safety guarantee illustrate how we slightly but correctly increased the precision of our argument for the claim that *the robot is safe*. The extension from AC_0 to AC_1 is an instance of the pattern in Figure 3. A complete pattern instance for this step is provided in [13, Fig. 6].

4.3 AC_2: Second Extension

For $\mathcal{M}_2$, we refine the data model of $\mathcal{M}_1$, where each location is reachable from everywhere, by a relation $Connection \subseteq LOCATION \times LOCATION$. Our model also contains a notion of distance between locations and allows to mark a specific location as the *goal*. Based on *Connection*, we extend $\mathcal{A}_2$ by a variable $trj\colon LOCATION\ list$ to manage routes and *oldDist* to measure the progress

[6] In the CPM model, environmental changes are encoded by $mon := *$, see [13].

towards the final *goal*. The following list summarises the model $\mathcal{M}_2$ and the corresponding extension AC_2:

$$
\begin{aligned}
\mathcal{I}_2 &\triangleq [\mathit{trj} \mapsto [], \mathit{occ} \mapsto \mathit{occ}] \\
\mathit{Inv}_2 &\triangleq \neg \mathit{occ}(p) \\
\mathcal{T}_2 &\triangleq \mathit{Plan} \sqcap \mathit{MicroMove} \sqcap \mathit{EmgBrake}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{As}_2 &\triangleq \mathit{minBrakingPrefix}(\mathit{trj}) \in \mathit{clearPaths} \\
\mathit{Gr}^s_{2.1} &\triangleq \mathit{hazardousMove} \Rightarrow \neg \mathit{occ}(p) \\
\mathit{Gr}^l_2 &\triangleq \mathit{clearPaths} \neq \varnothing \wedge p \neq \mathit{goal} \wedge (\neg \mathit{hazardousMicroMove}) \\
&\quad \Rightarrow \mathit{oldDist} > \mathit{dist}(p, \mathit{goal}) \\
\rho_2 &\triangleq \llparenthesis \mathit{occ}(p) \mapsto \mathit{minBrakingPrefix}(\mathit{trj}) \notin \mathit{clearPaths}, \mathit{occ}(\mathit{aim}) \mapsto \mathit{occ}(\mathit{aim}) \rrparenthesis
\end{aligned}
$$

Safety in $\mathcal{M}_2$ relies on the assumption As_2 that any update of *occ* by the environment will not lead to an occupancy of any prefix of the planned route *trj* shorter than the minimum braking distance, that is, $\mathit{minBrakingPrefix}(\mathit{trj}) \notin \mathit{clearPaths}$. Based on As_2, $Gr^s_{2.1}$ guarantees safe emergency braking, that is, not actively hitting any (moving) objects beyond minimum braking distance. This corresponds to the notion of *passive safety* in [26].

For liveness in $\mathcal{M}_2$, we use a conjunct $p \neq \mathit{goal}$ in the precondition for Gr^l_2 that specifies the termination of the robot's goal seeking activity. The postcondition of Gr^l_2 states that after each *MicroMove*, the robot should strictly get closer to the goal. Gr^l_2 is required for the desired liveness property of $\mathcal{M}_2$. However, only if $\mathit{clearPath} = \varnothing$ cannot occur infinitely often, Gr^l_2 implies termination.

The proof that AC_2 actually extends AC_1 works in a way similar to the extension proof for AC_1, now based on ρ_2. For $\mathcal{M}_2$, we use further parameters and definitions. These as well as the definitions of the three operations *Plan*, *MicroMove*, and *EmgBrake* are provided in the model in [13, Sec. 6].

5 Discussion

Here, we put our FMAC patterns into the context of formal robot verification, robotic engineering practice, and practically relevant standards. We also relate our contribution to model validation arguments.

Formal Robot Verification. Early work by Rahimi et al. [35] models a robot controller as a set of actions specified by pre/post conditions derived from hazard analysis. The authors use real-time logic to verify whether action implementations in software comply with these conditions. Beyond their work, our robot example demonstrates proof automation, refinement verification, and integration of proof evidence into a maintainable AC for certification.

Based on a CSP-inspired process algebra with the operational semantics of message-synchronous port automata, Lyons et al. [24] propose a plant model composed of environment, machine, and controller. Their controller model corresponds to our Kleene-starred CPM transition relation (cf. Theorem 1). The authors verify an elaborate plant model against *performance* (i.e., a generalisation of safety and liveness) guarantees by proving observational equivalence with reducing the embedded SAT problem to a filtering problem on Bayesian networks. Our approach based on Isabelle/UTP facilitates more generic abstraction and proof assistance based on relations and $\mathsf{d}\mathcal{L}$.

Mitsch et al. [26] model robots in $\mathsf{d}\mathcal{L}$ and verify successively refined notions of safety and liveness. These include *static safety* (i.e., no collision with static obstacles), *passive safety* (i.e., no active collision), *passive friendly safety* (i.e., safe braking circle does not intersect with other moving objects' safe braking circle), and *passive orientation safety* (i.e., braking cone does not intersect with other moving objects' braking cones). While our robot model is less detailed, we formalise the transition between increasingly precise notions of safety by data refinement, assumption weakening, and guarantee strengthening. Beyond [24] and [26], we demonstrate how robot validity and refinement proofs can be evolved within a standardised AC framework.

Though [26] does not explicitly invoke refinement in their stepwise development, refinement in $\mathsf{d}\mathcal{L}$ was previously investigated by Loos and Platzer [23]. They extend $\mathsf{d}\mathcal{L}$ with a refinement operator $\alpha \le \beta$, that specifies that a hybrid program α is more deterministic than a program β. If a safety guarantee, $\theta \Rightarrow [\beta]\phi$, can be proved for β, then the guarantee can automatically be derived for the refinement α (Theorem 2). Their notion of refinement permits local reasoning, such that subcomponents of a program need only demonstrate refinement under the condition they are reachable. Our refinement notion is global; however it is possible to derive their localised refinement relation in our setting. Effectively, our work can be seen as an extension of [23] with data refinement, which we believe can support stepwise development in the style of [26].

For a multi-robotic system, Desai et al. [8] verify safety and liveness properties of a trajectory coordination protocol based on a verified state-machine abstraction of almost-synchronously clocked plan execution units and asynchronous analysis and planning units. They apply SMT and A*-search for safe plan generation and model-checking of the coordination protocol. While their assumptions for modelling multi-robot coordination differ strongly from the assumptions applied in our single robot example, we can see the opportunity to enhance their fixed-model approach with data refinement to integrate multi-robot verification evidence into an extensible FMAC.

Industrial Standards and Verification Practices. Cooper et al. [6] demonstrate how formal methods (e.g. Z [38]) can be effectively practiced for security certification according to the Common Criteria standard [5]. However, back then, proof automation in AC construction was less researched and developed. Inspired by such examples, it is reasonable to aim for a transfer of our approach to the robotics and other safety-critical domains where FMs and ACs are highly recommended. For example, in the context of RTCA DO-178C, the FM supplement DO-333 [36] recommends the creation of "formal analysis cases" providing evidence for a variety of claims (e.g. Clauses FM 6.3.1-6.3.4), particularly, the satisfaction of high- and low-level safety guarantees. The automotive standard

ISO 26262 (e.g. Part 2, Clause 6.4.5.4) recommends a safety case for each system component with a safety goal and that these safety cases are subject of configuration and change management, thus, maintenance and evolution. Overall, these standards provide many opportunities for FMACs and Isabelle/SACM.

Adequacy and Completeness of the Formalisation. For controller design and synthesis, control engineers perform *model validation experiments* to assess how well a model of the process, they want to control, complies with the real world [37]. Likewise, the formal model associated with an FMAC (extension) has to be accompanied by an argument (potentially based on experiments using simulation and test [9]) that this model *faithfully abstracts from and predicts* [21] the implemented controller (i.e., the potentially distributed embedded system) and the surrounding plant. Isabelle/SACM allows us to enhance arguments accordingly. However, a further discussion of model validity arguments is out of scope.

Safety guarantees result from accident experience, domain expertise, and *hazard analysis* [22]. Regarding continuous hazard analysis, the pattern in Figure 3 accommodates changes of the hazard list and the corresponding guarantees as assumed claims (**C3,C4**) and the corresponding hazard mitigation as an undeveloped claim (**C9**). The step from hazard analysis to the derivation of new guarantees and model improvements is discussed in more detail in [12].

6 Conclusions

Assurance cases have to evolve to readjust or increase confidence [7]. Hence, we propose a framework for *formal model-based assurance case* construction and extension. Our framework is based on Isabelle/UTP whose semantic foundations allow one to express the system model for the construction of the assurance case in various but precisely linked formalisms, for example, relations and $\mathsf{d}\mathcal{L}$. This linking, paramount to the engineering of many critical systems [14], enables the step-wise refinement of the system model including data refinement and the simultaneous extension of an existing assurance case, resulting in an evolved assurance case readjusting or increasing the level of confidence of the argument. In [15], we discuss how model-based engineering can accommodate the way how innovation typically drives the evolution of requirements and designs. Extensible FMACs further develop this idea towards continuous model-based assurance.

Beyond the State of the Art. We propose the application of verification principles as recommended by, for example, DO-178C to mobile robot controllers. Our approach fosters *scalability* in two directions: first, via AC modules (i.e., A/G-style reasoning) devoted to specific assurance aspects, second, via compositional reasoning in Isabelle/UTP to isolate and reason about parts of large robot models. Regarding the former, we support *arguments at scale* by separation of the detailed proof structure in Isabelle/UTP from the overarching argument and evidence structure using Isabelle/SACM. This separation keeps the argument lean while maintaining traceability to all proof and model details.

Next Steps. The use of A/G-style specification for FMACs paired with invariants as constraints on the controlled and internal state variables improves the preservation of properties of FMAC compositions. To deal with more complex FMACs, we want to simplify composition and refinement at the level of FMACs by providing a new operator, $AC_a \sqsubseteq^{arg}_{\rho} AC_c$. We also want to improve the *modifiability* of an existing FMAC, particularly, support the deletion or substitution of guarantees on updates from hazard analysis. Furthermore, we want to enhance the handling of liveness guarantees across an extension step via $\oplus_{\rho}$.

Inspired by [1,34], we want to investigate the benefits of a further integration of Isabelle/Isar with the argument structure in Isabelle/SACM. Particularly, Basir [1] discusses how natural deduction program proofs (e.g. using Hoare logic) can formally underpin an argument and how interactive theorem proving can aid in checking the soundness of this argument.

Acknowledgements

This work is partly supported by the EPSRC projects CyPhyAssure[7], grant reference EP/S001190/1, and RoboCalc, grant reference EP/M025756/1.

References

1. Basir, N.: Safety cases for the formal verification of automatically generated code. Ph.D. thesis, University of Southampton (2010)
2. Bate, I., Kelly, T.: Architectural considerations in the certification of modular systems. Reliability Engineering & System Safety **81**(3), 303–324 (Sep 2003). https://doi.org/10.1016/S0951-8320(03)00094-2
3. Brucker, A.D., Aït-Sadoune, I., Crisafulli, P., Wolff, B.: Using the Isabelle ontology framework - linking the formal with the informal. In: CICM. Lecture Notes in Computer Science, vol. 11006, pp. 23–38. Springer (2018)
4. Calinescu, R., Weyns, D., Gerasimou, S., Iftikhar, M.U., Habli, I., Kelly, T.: Engineering trustworthy self-adaptive software with dynamic assurance cases. IEEE Transactions on Software Engineering **44**(11), 1039–1069 (nov 2018). https://doi.org/10.1109/tse.2017.2738640
5. Common Criteria Consortium: Common criteria for information technology security evaluation – part 1: Introduction and general model. Tech. Rep. CCMB-2017-04-001 (2017), https://www.commoncriteriaportal.org
6. Cooper, D., et al.: Tokeneer ID Station: Formal Specification. Tech. rep., Praxis High Integrity Systems (August 2008), https://www.adacore.com/tokeneer
7. Denney, E., Pai, G., Habli, I.: Dynamic safety cases for through-life safety assurance. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. IEEE (5 2015). https://doi.org/10.1109/icse.2015.199
8. Desai, A., Saha, I., Yang, J., Qadeer, S., Seshia, S.A.: DRONA: a framework for safe distributed mobile robotics. In: Proceedings of the 8th International Conference on Cyber-Physical Systems - ICCPS'17. ACM Press (2017). https://doi.org/10.1145/3055004.3055022

[7] CyPhyAssure Project: https://www.cs.york.ac.uk/circus/CyPhyAssure/

9. Edwards, S., Lavagno, L., Lee, E.A., Sangiovanni-Vincentelli, A.: Design of embedded systems: formal models, validation, and synthesis. Proceedings of the IEEE **85**(3), 366–90 (1997). https://doi.org/10.1109/5.558710
10. Foster, S., Baxter, J., Cavalcanti, A., Woodcock, J., Zeyda, F.: Unifying semantic foundations for automated verification tools in Isabelle/UTP. Submitted to Science of Computer Programming (March 2019), preprint: https://arxiv.org/abs/1905.05500
11. Foster, S., Zeyda, F., Nemouchi, Y., Ribeiro, P., Wolff, B.: Isabelle/UTP: Mechanised Theory Engineering for Unifying Theories of Programming. Archive of Formal Proofs (2019), https://www.isa-afp.org/entries/UTP.html
12. Gleirscher, M., Carlan, C.: Arguing from hazard analysis in safety cases: A modular argument pattern. In: High Assurance Systems Engineering (HASE), 18th Int. Symp. (1 2017). https://doi.org/10.1109/hase.2017.15
13. Gleirscher, M., Foster, S., Nemouchi, Y.: Evolution of formal model based assurance cases for autonomous robots. Supplemental material, University of York (2019). https://doi.org/10.5281/zenodo.3344489
14. Gleirscher, M., Foster, S., Woodcock, J.: New opportunities for integrated formal methods. Unpublished working paper, Department of Computer Science, University of York (2018)
15. Gleirscher, M., Vogelsang, A., Fuhrmann, S.: A model-based approach to innovation management of automotive control systems. In: 8th Int. Workshop on Software Product Management (IWSPM). IEEE digital library (2014). https://doi.org/10.1109/IWSPM.2014.6891062
16. Hawkins, R., Habli, I., Kolovos, D., Paige, R., Kelly, T.: Weaving and assurance case from design: A model-based approach. In: Proc. 16th Intl. Symp. on High Assurance Systems Engineering. IEEE (2015)
17. Hoare, C.A.R., He, J.: Unifying Theories of Programming. Prentice-Hall (1998)
18. Jackson, M.A.: Problem Frames: Analysing & Structuring Software Development Problems. Addison-Wesley (2001)
19. Kelly, T.: Arguing Safety – A Systematic Approach to Safety Case Management. Ph.D. thesis, University of York (1998)
20. Kelly, T.P., McDermid, J.A.: Safety case construction and reuse using patterns. In: Daniel, P. (ed.) SAFECOMP, 16th Int. Conf. pp. 55–69. Springer (1997). https://doi.org/10.1007/978-1-4471-0997-6_5
21. Lee, E.A., Sirjani, M.: What good are models? In: Formal Aspects of Component Software, pp. 3–31. Springer International Publishing (2018). https://doi.org/10.1007/978-3-030-02146-7_1
22. Leveson, N.G.: Engineering a Safer World: Systems Thinking Applied to Safety. Engineering Systems, MIT Press (1 2012). https://doi.org/10.7551/mitpress/8179.001.0001
23. Loos, S.M., Platzer, A.: Differential refinement logic. In: Proc. 31st Intl. Symp. on Logic in Computer Science (LICS). ACM (July 2016)
24. Lyons, D.M., Arkin, R.C., Jiang, S., Liu, T.M., Nirmal, P.: Performance verification for behavior-based robot missions. IEEE Transactions on Robotics **31**(3), 619–636 (jun 2015). https://doi.org/10.1109/tro.2015.2418592
25. Meyer, B.: Applying "design by contract". IEEE Computer **25**(10), 40–51 (1992)
26. Mitsch, S., Ghorbal, K., Vogelbacher, D., Platzer, A.: Formal verification of obstacle avoidance and navigation of ground robots. CoRR (2016), http://arxiv.org/abs/1605.00604

27. Nair, S., de la Vara, J.L., Sabetzadeh, M., Falessi, D.: Evidence management for compliance of critical systems with safety standards: A survey on the state of practice. Information and Software Technology **60**, 1–15 (apr 2015). https://doi.org/10.1016/j.infsof.2014.12.002
28. Nemouchi, Y., Foster, S., Gleirscher, M., Kelly, T.: Mechanised assurance cases with integrated formal methods in Isabelle. In: Submitted to iFM 2019 (2019), preprint: https://arxiv.org/abs/1905.06192
29. Nipkow, T., Wenzel, M., Paulson, L.C.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic. Lecture Notes in Computer Science, Springer, Berlin, 1 edn. (2002). https://doi.org/10.1007/3-540-45949-9
30. de Oliveira, A.L., Braga, R.T., Masiero, P.C., Papadopoulos, Y., Habli, I., Kelly, T.: Supporting the automated generation of modular product line safety cases. Advances in Intelligent Systems and Computing **365**, 319–330 (2015). https://doi.org/10.1007/978-3-319-19216-1_30
31. Palin, R., Habli, I.: Assurance of automotive safety – a safety case approach. In: Schoitsch, E. (ed.) Computer Safety, Reliability, and Security, LNCS, vol. 6351, pp. 82–96. Springer (2010). https://doi.org/10.1007/978-3-642-15651-9_7
32. Parnas, D.L., Madley, J.: Function documents for computer systems. Science of Computer Programming **25**, 41–61 (1995)
33. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom Reasoning **41**, 143–189 (June 2008)
34. Prokhorova, Y., Laibinis, L., Troubitsyna, E.: Facilitating construction of safety cases from formal models in event-b. Information & Software Technology **60**, 51–76 (2015). https://doi.org/10.1016/j.infsof.2015.01.001
35. Rahimi, M., Xiadong, X.: A framework for software safety verification of industrial robot operations. Computers & Industrial Engineering **20**(2), 279–287 (jan 1991). https://doi.org/10.1016/0360-8352(91)90032-2
36. RTCA: DO-333: Formal Methods Supplement to DO-178C and DO-278A (2012)
37. Smith, R.S., Doyle, J.C.: Model validation: a connection between robust control and identification. IEEE Trans. Automatic Control **37**(7), 942–52 (Jul 1992). https://doi.org/10.1109/9.148346
38. Spivey, J.: The Z Notation: A Reference Manual. Prentice Hall (1992)
39. Wei, R., Kelly, T., Dai, X., Zhao, S., Hawkins, R.: Model based system assurance using the structured assurance case metamodel. Journal of Software and Systems **154**, 211–233 (August 2019)
40. Woodcock, J., Davies, J.: Using Z: Specification, Refinement, and Proof. Prentice Hall (1996)