UNIVERSITY of York

This is a repository copy of *Verified Synthesis of Optimal Safety Controllers for Human-Robot Collaboration*.

White Rose Research Online URL for this paper:
https://eprints.whiterose.ac.uk/174848/

Version: Published Version

Article:
Gleirscher, Mario orcid.org/0000-0002-9445-6863, Calinescu, Radu orcid.org/0000-0002-2678-9260, Douthwaite, James et al. (5 more authors) (2022) Verified Synthesis of Optimal Safety Controllers for Human-Robot Collaboration. Science of Computer Programming. 102809. ISSN 0167-6423

https://doi.org/10.1016/j.scico.2022.102809

Reuse
This article is distributed under the terms of the Creative Commons Attribution (CC BY) licence. This licence allows you to distribute, remix, tweak, and build upon the work, even commercially, as long as you credit the authors for the original work. More information and the full terms of the licence here: https://creativecommons.org/licenses/

Takedown
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

White Rose university consortium
Universities of Leeds, Sheffield & York

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk/

Science of Computer Programming 218 (2022) 102809

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Verified synthesis of optimal safety controllers for human-robot collaboration

Mario Gleirscher [a,c,*], Radu Calinescu [a], James Douthwaite [b], Benjamin Lesage [a], Colin Paterson [a], Jonathan Aitken [b], Rob Alexander [a], James Law [b]

[a] University of York, Deramore Lane, York YO10 5GH, UK
[b] University of Sheffield, Mappin Street, Sheffield S1 3JD, UK
[c] University of Bremen, Bibliothekstrasse 5, 28359 Bremen, Germany

ARTICLE INFO

Article history:
Received 11 June 2021
Received in revised form 23 March 2022
Accepted 29 March 2022
Available online 4 April 2022

Keywords:
Risk-informed design automation
Formal verification
Probabilistic model checking
Collaborative robot safety
Digital twins

ABSTRACT

We present a tool-supported approach to the synthesis, verification, and testing of the control software responsible for the safety of human-robot interaction in manufacturing processes that use collaborative robots. In human-robot collaboration, software-based safety controllers are used to improve operational safety, for example, by triggering shutdown mechanisms or emergency stops to reduce the likelihood of accidents. Complex robotic tasks and increasingly close human-robot interaction pose new challenges to controller developers and certification authorities. Key among these challenges is the need to assure the correctness of safety controllers under explicit (and preferably weak) assumptions. Our integrated synthesis, verification, and test approach is informed by the process, risk analysis, and relevant safety regulations for the target application. Controllers are selected from a design space of feasible controllers according to a set of optimality criteria, are formally verified against correctness criteria, and are translated into executable code and tested in a digital twin. The resulting controller can detect the occurrence of hazards, move the process into a safe state, and, under certain circumstances, return the process to an operational state from which it can resume its original task. We show the effectiveness of our software engineering approach through a case study involving the development of a safety controller for a manufacturing work cell equipped with a collaborative robot.

© 2022 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

Effective collaboration between humans and robots [1,2] can leverage their complementary skills. But such collaboration is difficult to achieve because of uncontrolled hazards and because sensing, tracking, and safety measures are either still unexploited in practice [3] or they are difficult to validate following state-of-the-art safety regulations [4]. Since the 1980s, remote programming (also called tele-programming) and simulation have led to some reduction of hazard exposure. However, the effectiveness of human-robot collaboration is still limited because of frequent conservative shutdowns,

* Corresponding author at: University of Bremen, Bibliothekstrasse 5, 28359 Bremen, Germany.
E-mail addresses: mario.gleirscher@york.ac.uk (M. Gleirscher), radu.calinescu@york.ac.uk (R. Calinescu), j.douthwaite@sheffield.ac.uk (J. Douthwaite), benjamin.lesage@york.ac.uk (B. Lesage), colin.paterson@york.ac.uk (C. Paterson), jonathan.aitken@sheffield.ac.uk (J. Aitken), rob.alexander@york.ac.uk (R. Alexander), j.law@sheffield.ac.uk (J. Law).

https://doi.org/10.1016/j.scico.2022.102809
0167-6423/© 2022 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

simplistic emergency stops, and unspecific error handling procedures. Extensive guarding arrangements interfere with manufacturing processes and mobile robot applications. But effective work processes and complex tasks require continuous close human-robot interaction (e.g., mutual take-over of tasks), mutual clarification of intent, and trading off risk [5,6]. From an operator's perspective, robot movements need to be predictable, and potential impacts on the human body need to be attenuated. From a control perspective, the confident monitoring and control of the robot speed and the separation between machines and humans require high-quality stereo vision and laser scanners to distinguish several safety zones. A decade after these issues were discussed by Alami et al. [7] and Haddadin et al. [8], Ajoudani et al. [9] emphasise that the complex safety challenges of collaborative robots (cobots for short) [10] remain largely unresolved. Increasingly complex robotic systems reduce the ability to understand and mitigate risks. Safety is a major barrier to the more widespread adoption of cobots, with organisations such as manufacturers having to resort to sub-optimal processes due to safety concerns. Methods of ensuring safe human-robot interaction will deliver $7.3bn (€6.4bn) of savings, reducing costs of US-manufactured goods by 1% [11]. However, little such method and tool support is available for engineers that have to implement and confidently assure cobot safety requirements, that is, the results of cobot hazard analyses and risk assessments [4].

Problem. Among all measures for improving cobot safety, software-based *safety controllers* are responsible for the monitoring of an application process, the handling of critical events, and the mitigation of operational risk. To facilitate smooth human-robot collaboration with minimal interruption, the software engineers responsible for developing these controllers need to closely consider the process with its variety and complexity of adverse events, such as unusual operator behaviour and equipment failure modes. This complexity leads to many requirements and large controller design spaces. Hence, these engineers must address questions in several areas:

1. *Risk assessment.* Which controller minimises the probability of incidents from human and sensor errors?
2. *Controller synthesis.* Which design minimises nuisance to the operator, maximises productivity, etc. safely?
3. *Controller verification.* How well does a controller return the system to a useful safe state after handling hazards?
4. *Controller testing.* Does an implementation of the synthesised controller exhibit the intended behaviour?

Preliminary solution. In [12], we provide initial answers to the first three questions, by introducing a preliminary software engineering method for the synthesis of discrete-event safety controllers that implement safety requirements and optimise process performance in human-robot collaboration. This method presents a layer of abstraction to the engineer by integrating engineering steps, modelling techniques, and tools. We model the application (e.g., a manufacturing process) as a Markov decision process (MDP), and select correct-by-construction controllers from an associated design space. The process model describes the behaviour of all involved actors including the controller. To describe critical events (e.g., hazards) and controller actions (e.g., safety mode changes), we employ the notion of *risk structures* [13,14] implemented in the risk-informed controller designer YAP [15,16] used for risk structuring, qualitative risk analysis, and synthesis of risk-informed safety controllers. In particular, YAP supports risk modelling and controller design with a domain-specific language and automates the transformation of risk structures into guarded command language and, in turn, MDPs. The preliminary approach in [12] facilitates the verification of MDP safety properties and of probabilistic reach-avoid properties of selected MDP policies including the controller. A verified controller extracted from such a policy detects hazards and controls their mitigation by the execution of a safety function, a transition to a particular safety mode, or a safer activity. Furthermore, under certain circumstances, the controller returns the process to an operational state from which it can resume its original activity.

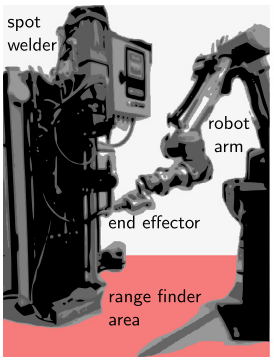
Contributions. This paper builds on our preliminary work [12,16] to substantially improve process and risk modelling to enable the synthesis of finer-grained controllers. The main idea is that a safety controller is designed for a system comprising activities where humans and robots collaborate in order to increase safety with respect to risks identified for the system.

Our approach is presented as a three-stage process of (i) the modelling of the controller design space, (ii) verified optimal controller synthesis, and (iii) controller deployment and integration testing. Within these stages, we provide a refined notion of risk factors in Section 4.2.1. We improve and extend the verification stage by checking additional properties, increasing the confidence in the synthesised controllers. We employ the stochastic model synthesis tool EVOCHECKER [17] in addition to the probabilistic model checker PRISM [18], extending the verification capabilities of the optimal synthesis procedure to significantly more complex combinations of controller requirements.

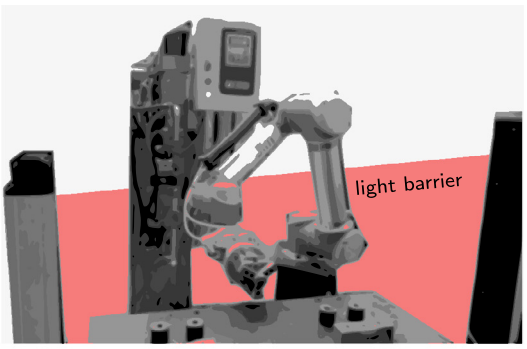
Furthermore, we translate the controllers into executable code for a Digital Twin framework (DTF) [19] to perform controller integration testing in a realistic environment. We assure the correctness of this translation by reusing properties from model checking in testing. Controller integration testing with the DTF is based on randomised tests that cover the relevant situations (i.e., process states and transitions) related to a use case of interest.

Finally, we demonstrate in the DTF how the synthesised safety controller can automatically resume the nominal procedure after the mitigation of certain hazards. We provide experimental results that confirm our safety controller's ability to achieve an increased process utility in addition to ensuring high levels of freedom from accidents.

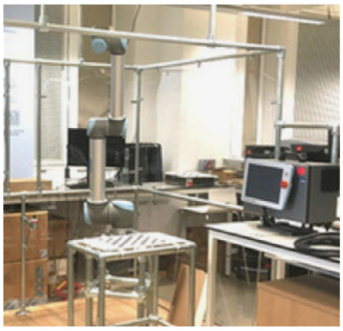
The rest of the paper is structured as follows. Section 2 introduces our running example which considers a manufacturing cell. We describe the conceptual model, the process as an activity transition system and present the initial safety analysis. Section 3 provides the theoretical background necessary to understand our contribution. Section 4 describes our approach for the synthesis of safety controllers including process modelling, risk modelling, and controller design. Section 5 integrates



(a) Safeguarded area (company)

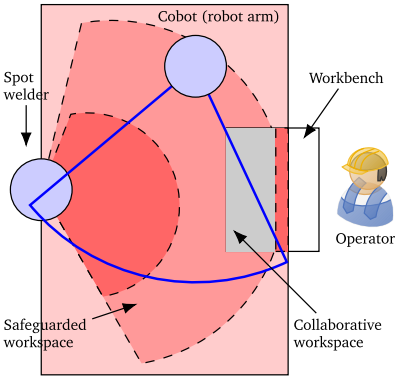


(b) Workbench (company) with two handover locations (left and right receptacle) for part exchange

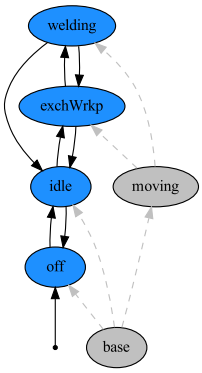


(c) Replica (research lab)

Fig. 1. Actual (a, b) and replicated (c) work cell with cobot.



(a) Work cell with cobot (conceptual, top view)



(b) Process activities

Fig. 2. Work cell concept (a) and manufacturing activities (b) performed by the operator, the robot, and the spot welder, classified by the activity groups *moving* and *base* (in light grey).

the risk and mitigation models with the process models to generate a risk-informed reward-enhanced stochastic model from which we synthesise verified safety controllers. Section 6 then demonstrates how we deploy our verified controller, in a concrete executable form, to a digital twin to allow for integration testing. Section 7 evaluates key aspects of our approach concerning safety, utility, and scalability. Section 8 highlights those works upon which our approach is founded before we conclude with a short summary and an indication of areas of future work in Section 9. For convenience we include an overview of symbols used throughout the paper in Table 7.

2. Running example: manufacturing cobots

The Figs. 1a and 1b show a cobot-equipped manufacturing *work cell* at a UK company (with the pictures anonymised for confidentiality reasons) and replicated in a testbed at the University of Sheffield (Fig. 1c). Fig. 1b depicts the workbench including a collaborative workspace. The conceptual bird's eye view of this work cell is provided in Fig. 2a.

In the cell, an operator, a stationary collaborative robotic manipulator (cobot or robot arm for short), and a spot welder (Fig. 2a) work together on a repetitive welding and assembly task. The operator handles a workpiece and passes it to the robot arm using two receptacles. The robot arm exchanges the workpiece (*exchWrkp*) between the workbench and the spot welder. In between, a *welding* task is performed by the robot arm and spot welder. The work cell can also be *idle* and switched *off*. Fig. 2b shows an activity[1] transition system with the described concrete activities (in blue), the solid arcs indicating how the robot arm and spot welder can traverse the activities, and the activity groups *moving* and *base* (in light grey) modelling features common across several activities (dashed arcs). In the remainder, we denote the collaboration abstracted by the activity transition system as the process $\mathcal{P}$, including a safety controller *sc*. We refer to $\mathcal{P}$'s implementation using the DTF as *MCobot*.

[1] For the sake of simplicity, we use the notion of an *activity* as a hypernym describing a task, a situation, a use case, or a scenario.

Table 1
Our partial safety analysis of the manufacturing cell referring to the measures recommended in ISO/TS 15066 [20].

Id	**Critical Event** (modelled as risk factor)	**Safety Requirement**
	Accident (to be prevented or alleviated)	**Safety Goal**
RC	The Robot arm harshly Collides with an operator.	The robot shall *reduce the likelihood* of harsh active collisions with the operator.
WS	Welding Sparks cause operator injuries (skin burns).	The welding process shall *reduce the likelihood* of sparks injuring the operator.
RT	The Robot arm Touches the operator.	The robot shall *reduce the likelihood* of active contact with the operator.
	Latent Cause (to be mitigated timely)	**Controller Requirement**[†,‡]
HRW	The Human operator and the Robot use the Workbench at the same time.	(ϕ_m) The robot shall perform an appropriate mitigation (e.g., a safety-rated monitored stop) and (ϕ_r) resume *normal operation* after the *operator* has left the *shared workbench*.
HW	The Human operator is entering the Workbench while the robot is away from the workbench.	(ϕ_m) If the robot moves a workpiece to the workbench then it shall switch to *power & force limiting* mode and (ϕ_r) resume *normal operation* after the *operator* has left the *workbench*.
HS	The Human operator has entered the Safeguarded area while the robot or the spot welder are active.	(ϕ_m) The *spot welder* shall be *switched off*, the *robot* to *speed & separation monitoring*, and the operator be notified to leave. (ϕ_r) Robot and spot welder shall resume normal mode after the operator has left.
HC	The Human operator is Close to the welding spot while the robot is working and the spot welder is active.	(ϕ_m) The *spot welder* shall be *switched off*, the *robot* to *safety-rated monitored stop*. (ϕ_r) Both shall resume *normal or idle mode with a reset procedure* after the operator has left.

[†] ϕ_m: mitigation requirement, ϕ_r: resumption requirement; [‡] subjected to generalisation to define a controller design space.

Previous safety analysis (i.e., hazard identification, risk assessment, and requirements derivation) resulted in two areas safeguarded by sensors. The first area (highlighted at the bottom of Fig. 1a) uses a range finder with a rotating laser beam to approximate the distance (two-staged as indicated by two concentric dashed sectors in Fig. 2a) between the spot welder, the robot arm (solid sector), and a person or an object intruding into that area. The range finder triggers a slow down or an emergency stop if the intruder approaches the spot welder. The second area is the collaborative workspace (indicated in Fig. 2a) behind a light barrier (the highlighted curtain indicated in Fig. 1b) triggering such a stop if something like a person's arm reaches across the workbench while the robot or the spot welder are active.

Table 1 shows our partial safety analysis of the cell following the guidance in Section 1. The right column specifies probabilistic safety goals against each accident and non-probabilistic controller requirement candidates ϕ_m and ϕ_r (e.g., mode-switch requirements), handling each latent cause in the left column and indicating with ϕ_m how the corresponding hazard is to be removed. The running example is part of a case study organised around the AAIP project CSI: Cobot.[2] Due to Covid-19 restrictions limiting access to physical facilities during a critical phase of the project, we use a digital twin of the work cell as a target platform to deploy and test the synthesised safety controllers.

3. Background

This section summarises the background of the presented approach, particularly, robot safety, probabilistic model analysis, risk modelling for controller synthesis, integration testing, and digital twins.

3.1. Robot safety: from industrial to collaborative

Hazards from robots have been studied since the advent of industrial robotics in the 1970s, resulting in risk taxonomies based on workspaces, tasks, and human body regions [21,2,7,8,22–25]. The majority of hazards are *impact hazards* (e.g., unexpected movement, reach beyond a permitted area, dangerous workpieces, hazardous manipulation), *trapping hazards* (e.g., an operator locked in a cage), and *failing equipment* (e.g., valve, cable, sensor, controller). In the 1980s, robots were programmed interactively by operators being in the cage while powered, which had caused frequent accidents from trapping and collision. However, from the late 1990s on, the increased use of tele-programming contributed to the reduction of accidents related to such hazards.

Addressing hazards outside the programming stage involves the examination of each *mode of operation* (e.g., normal, maintenance) for its hazardous behaviour, and the use of safety controllers to trigger mode-specific *safety measures* [2]. Malfunction diagnostics (e.g., fault detection, wear-out monitoring) can further inform these controllers. Table 2 shows a variety of measures [3] to prevent or mitigate hazards and accidents by reducing the probability of the occurrence and the severity of the consequences of these hazards. If these measures use electronic or mechatronic equipment, we speak of *functional*[3] measures (e.g., safety modes as exemplified below) and of *intrinsic* measures otherwise (e.g., a fence around a robot, flexible robot surfaces). Functional measures focusing on the correctness and reliability of a controller (a programmable electronic

[2] See https://www.sheffield.ac.uk/sheffieldrobotics/about/csi-cobot.

[3] *Functional safety* (see IEC 61508, ISO 26262) deals with the dependability, particularly, correctness and reliability, of *critical* programmable electronic systems. Safety functions or "functional measures" are the archetype of such systems.

Table 2
Cobot safety measures associated to stages in the causal chain of events.

Stage	Type of Measure	Examples
Hazard prevention	1. Safeguard/barrier	Fence, cage, interlock
	2. IT safety	*Verified*[†] *safety controller*
	3. IT security	Security-verified[†] (safety) controller
Hazard mitigation & accident prevention	4. Reliability	Fault-tolerant scene interpretation
	5. Workspace intrusion detection	Speed & separation monitoring, safety-rated monitored stop
	6. Shift of control	Hand-guided operation
Accident mitigation (alleviation)	7. Power & force limitation	Low weight parts, flexible surfaces; variable impedance, touch-sensitive, & force-feedback control
	8. System halt	Emergency stop, dead-man's switch

[†]avoidance of development or programming mistakes.

or software system) are called *dependability* measures [7,26]. Functional measures are said to be *active* if they focus on accident prevention (e.g., collision avoidance) and *passive* if they are limited to severity reduction (e.g., force-feedback control for limiting contact forces). In this work, we focus on the verified synthesis [27] of safety controllers that realise active functional safety measures.

Standardisation of safety requirements for industrial robots [21] culminated in ANSI/RIA R15.06, ISO 10218 [28], ISO 13482, and ISO/TS 15066 [20]. Following ISO 10218, such robot systems comprise a robot arm, a robot controller, an end-effector, and a workpiece (see, e.g., Fig. 2a). According to Helms et al. [29] and Kaiser et al. [23], one can distinguish four scenarios of human-robot interaction: (i) Encapsulation in a fenced robot workspace, (ii) co-existence without fencing but separation of human and robot workspace, (iii) cooperation with alternative exclusive use of the shared workspace, and (iv) collaboration with simultaneous use of the shared workspace and close interaction. Cooperation and collaboration are the two most interactive of these scenarios and motivate our work. In collaborative operation, the operator and the cobot [10] can occupy the collaborative workspace simultaneously while the cobot is performing tasks [20, Pt. 3.1]. The *collaborative workspace* has to be a subset of the *safeguarded workspace*, that is, a subset of the areas observed by sensors (Fig. 2a).

Based on these definitions, ISO 10218 and ISO/TS 15066 [20, Clause 5.5] recommend four *safety modes*. First, a *safety-rated monitored stop* is an active functional measure realised as a mode where the robot is still powered but there is no simultaneous activity of the robot and the operator in the shared workspace. Second, *hand-guided operation* refers to a mode with zero-gravity control, that is, control without actuation beyond the compensation of gravity, solely guided by an operator. Hand guidance requires the robot to be in a compliant state, with control exerted by the operator through physical manipulation. Third, *speed & separation monitoring* as an active functional measure refers to a mode where speed is continuously adapted to the distance of the robot and an operator. Forth, *power & force limiting* is a mode with reduced impact of the robot on the human body and the robot's power and applied forces are limited. In this mode, a robot should not impact a human with more than a defined force, with acceptable forces mapped out for different impact points on the body. Heinzmann and Zelinsky [30] propose such a mode always active during a collaborative activity described as a discrete-event controller. Long et al. [31] propose a distance-triggered scheme to switch between nominal (max. velocity), reduced (speed limiting), and passive (hand-guided) operating modes. Kaiser et al. [23] and Villani et al. [6] describe and combine these modes with work layouts.

In addition to these safety modes, Alami et al. [7] highlight the necessity of a shift from robots whose motion is controlled only by following a pre-specified route of positions (also known as position control) to robots whose motion control minimises contact forces and energy (also known as interaction control). Overall, interaction-controlled robots, with less pre-planning and fewer assumptions on workspace structure and robot actions, can exploit the mentioned safety modes more effectively than position-controlled robots with extensive pre-planning and stronger assumptions.

3.2. Probabilistic model checking

The proposed approach employs MDPs, and corresponding sets of discrete-time Markov chains (DTMCs), as a formal semantics of the process $\mathcal{P}$, and uses the policy set of an MDP, describing the degrees of freedom for decision making, to model the design space for verified controller synthesis.

Definition 1. Markov Decision Process (MDP). Given all distributions $Dist(\cdot)$ over a sort (e.g., an action alphabet $A_{\mathcal{P}}$ of a process $\mathcal{P}$), an MDP is a tuple $\mathcal{M} = (S, s_0, A_{\mathcal{P}}, \delta_{\mathcal{P}}, L)$ with a set S of states, an initial state $s_0 \in S$, a (partial) probabilistic transition function $\delta_{\mathcal{P}} \colon S \times A_{\mathcal{P}} \to Dist(S)$, and a map $L \colon S \to 2^{AP}$ labelling S with atomic propositions AP [32,33].

Note that $\delta_{\mathcal{P}}$ induces a *transition relation* $\delta_{\mathcal{P}}^{\rightarrow} \subseteq S \times S$ where $\delta_{\mathcal{P}}^{\rightarrow} = \{(s, s') \in S \times S \mid \exists \alpha \in A_{\mathcal{P}} \colon \delta_{\mathcal{P}}(s, \alpha)(s') > 0\}$. Given a map $A \colon S \to 2^{A_{\mathcal{P}}}$ defined as $A(s) = \{\alpha \in A_{\mathcal{P}} \mid \delta_{\mathcal{P}}(s, \alpha) \text{ is defined}\}$, $|A(s)| > 1$ signifies non-deterministic choice in state s, giving rise to policies for $\mathcal{M}$.

Definition 2. Memoryless Policy. A *memoryless policy* is a map $\pi: S \to Dist(A_{\mathcal{P}})$ such that $\pi(s)(\alpha) > 0 \Rightarrow \alpha \in A(s)$. π is *deterministic* if $\forall s \in S\ \exists \alpha \in A(s):\ \pi(s)(\alpha) = 1 \wedge \forall \alpha' \in A_{\mathcal{P}} \setminus \{\alpha\}: \pi(s)(\alpha') = 0$.

Let $\Pi_{\mathcal{M}}$ be the set of all policies for $\mathcal{M}$ and $\mathcal{M}_\pi$ be the DTMC induced by $\pi \in \Pi_{\mathcal{M}}$. Then, *action rewards* defined by a map $rw^q_{\text{action}}: S \times A_{\mathcal{P}} \to \mathbb{Q}_{\geq 0}$ allow the comparison of policies in $\Pi_{\mathcal{M}}$ based on a quantity q. In this paper, we do not use state rewards and restrict ourselves to the consideration of deterministic policies.[4] Verification of $\mathcal{M}$ can be done with probabilistic reward computation tree logic (PRCTL; [34, p. 826]) whose properties over AP are formed by

$$\phi ::= ap \mid \neg\phi \mid \phi \vee \phi \mid \mathbf{P}_{\sim pr|[\min|\max]=?}\,\phi_{\text{path}} \mid \mathbf{R}^q_{\sim b|[\min|\max]=?}[\mathbf{F}\,\phi \mid \mathbf{C}^{[\leq t]}] \mid \mathbf{S}_{\sim pr|=?}\,\phi \text{ and} \qquad \text{(state formula)}$$

$$\phi_{\text{path}} ::= \mathbf{X}\,\phi \mid \phi\,\mathbf{U}^{[\leq t]}\,\phi \qquad \text{(path formula)}$$

with $ap \in AP$ and the (quantification) operator $\mathbf{P}_{\sim pr|[\min|\max]=?}\,\phi_{\text{path}}$ with $\sim \in \{\leq, <, \geq, >\}$ to verify (or with $=?$, to quantify) probabilities.[5] We use the abbreviations $true \equiv \phi \vee \neg\phi$, $false \equiv \neg true$, $\phi \wedge \psi \equiv \neg(\neg\phi \vee \neg\psi)$, $\phi \to \psi \equiv \neg\phi \vee \psi$, $\mathbf{E}\,\phi \equiv \mathbf{P}_{>0}\,\phi$, $\mathbf{A}\,\phi \equiv \neg\mathbf{E}\neg\phi$, $\mathbf{F}\,\phi \equiv true\,\mathbf{U}\,\phi$, $\mathbf{G}\,\phi \equiv \neg\mathbf{F}\neg\phi$, and $\phi\,\mathbf{W}\,\psi \equiv \phi\,\mathbf{U}\,\psi \vee \mathbf{G}\,\phi$. The operator $\mathbf{R}^q_{\sim b|[\min|\max]=?}[\mathbf{F}\,\phi \mid \mathbf{C}^{[\leq t]}]$ evaluates (or with $=?$, calculates) reachability rewards ($\mathbf{F}\,\phi$) and (total or bounded) cumulative (action) rewards ($\mathbf{C}^{[\leq t]}$) for a quantity q. In **P** and **R**, min and max are only used with MDPs. For an induced DTMC $\mathcal{M}_\pi$, the (non-PRCTL-standard) operator $\mathbf{S}_{\sim pr|=?}\,\phi$ evaluates (or with $=?$, quantifies) the steady-state or long-run probability of ϕ. **P** and **S** allow a probability bound $pr \in [0, 1]$; **U**, **F**, **G**, and **C** an optional upper[6] time bound $t \in \mathbb{N}_+$; and **R** a reward bound $b \in \mathbb{Q}_{\geq 0}$ [34, p. 781].

With $\mathcal{M}, s \models \phi$, we state that the MDP $\mathcal{M}$ from state s *satisfies* the property ϕ under all policies in $\Pi_{\mathcal{M}}$. $\pi, s \models \phi$ denotes that the policy $\pi \in \Pi_{\mathcal{M}}$ from state s satisfies ϕ and $s \models \phi$ that the state $s \in S$ satisfies the state predicate ϕ. We abbreviate $\mathcal{M}, s_0 \models \phi$ to $\mathcal{M} \models \phi$ and use $\mathcal{M} \not\models \phi$ if and only if $\mathcal{M} \models \phi$ does not hold. For a state predicate ϕ, $[\![\phi]\!]_S$ then denotes the largest subset $S' \subseteq S$ with $\forall s \in S': s \models \phi$. Let $Paths(\mathcal{M}, s)$ denote the set of all (infinite) state sequences $\omega: \mathbb{N} \to S$ of $\mathcal{M}$ based on the reachability graph induced by $\delta_{\mathcal{P}}$ from s. We abbreviate $Paths(\mathcal{M}, s_0)$ with $Paths(\mathcal{M})$ and use $\omega^t(i) = \omega(i + t)$ with $i \in \mathbb{N}$ for the suffix of ω from position or time t. By $\omega \models \phi$, we express that ω satisfies ϕ. Based on L and $\Pi_{\mathcal{M}}$, the semantics of PRCTL can be defined inductively:

$$\begin{aligned}
\mathcal{M}, s &\models true && \text{always}\\
\mathcal{M}, s &\models ap && \iff ap \in L(s) \quad \text{for} \quad ap \in AP\\
\mathcal{M}, s &\models \neg\phi && \iff \mathcal{M}, s \not\models \phi\\
\mathcal{M}, s &\models \phi \vee \psi && \iff \mathcal{M}, s \models \phi \quad \text{or} \quad \mathcal{M}, s \models \psi\\
\mathcal{M}, s &\models \mathbf{P}_{\sim pr}\,\phi_{\text{path}} && \iff \forall \pi \in \Pi_{\mathcal{M}}: Pr^{\pi}_{\mathcal{M},s}\{\omega \in Paths(\mathcal{M}, s) \mid \omega \models \phi_{\text{path}}\} \sim pr\\
\mathcal{M}, s &\models \mathbf{R}^q_{\sim b}[\mathbf{C}^{[\leq t]}] && \iff \forall \pi \in \Pi_{\mathcal{M}}: \mathbb{E}^{\pi}_{\mathcal{M},s}\big(\Sigma_{i=0}^{[t-1|\infty]} rw^q_{\text{action}}(s_i, \alpha_i)\big) \sim b\\
\mathcal{M}_\pi, s &\models \mathbf{S}_{\sim pr}\,\phi && \iff \Sigma_{\mathcal{M}_\pi, s' \models \phi} Pr^{\pi}_{\mathcal{M},s}(s') \sim pr\\
\omega &\models \mathbf{X}\,\phi && \iff \omega(1) \models \phi\\
\omega &\models \phi\,\mathbf{U}^{[\leq t]}\,\psi && \iff \exists 0 \leq j\,[\leq t]: \omega(j) \models \psi \wedge \big(\forall 0 \leq k < j: \omega(k) \models \phi\big)
\end{aligned}$$

where $\alpha_i = \arg\max_{\alpha \in A_{\mathcal{P}}} \pi(s_i)(\alpha)$ for any $\omega = s_0 s_1 \cdots \in Paths(\mathcal{M}_\pi, s)$ and $i \in \mathbb{N}$. Further details on the probability measure $Pr^{\pi}_{\mathcal{M},s}$ (defined over sets of paths from s and states of $\mathcal{M}$) and a comprehensive treatment of stochastic model checking are available from Kwiatkowska et al. [32], Forejt et al. [33] and Baier and Katoen [34].

3.3. Modelling of and policy synthesis for Markov decision processes

Probabilistic model checkers such as PRISM [32,18] and STORM [35] use efficient symbolic model checking algorithms to verify PRCTL-encoded properties of DTMCs and MDPs. These models can be defined in a high-level modelling language also known as probabilistic guarded command language (pGCL). pGCL allows one to describe a system as a program, in particular, as a composition of a set of *modules*, each module consisting of a set of *guarded commands* as defined below. The *state* of each module is given by a set of finite-range local variables. Let X be the set of all state variables and $\mathbb{U}$ the universe including $\mathbb{B}$, $\mathbb{N}$, $\mathbb{Q}$, and enumerations, and for $X' \subseteq X$, $\text{type}\,X' \subseteq \mathbb{U}^{|X'|}$ the domain (or set of valid tuples) of X'. For $x \in X$ with $\text{type}\,x = T$, we allow the convention $x: T$. Then, a state $s \in S$ is a valuation function $s: X \to \mathbb{U}$ with $\forall x \in X: s(x) \in \text{type}\,x$. Map restriction $\cdot|_Y$ restricts a (set of) state(s) from $X \to \mathbb{U}$ to $X \cap Y \to \mathbb{U}$. Let AP be a set of expressions of the form $x \sim u$ with a variable $x \in X$ and a function (or constant) $u: S \to (\mathbb{Q} \cup \mathbb{B})$. AP is interpreted over S by L. Hence, $L(s)$ determines all true atomic propositions for state s by evaluating each proposition in AP for $s \in S$. For example, "$x = u$" $\in L(s)$ iff $s(x) = u(s)$.

[4] Memoryless deterministic policies reduce the flexibility of our MDP policy synthesis, however, not detrimental to our purposes.

[5] With $[\cdot]$ and $\cdot \mid \cdot$, we denote optional and alternative syntactic choice in sub- or superscript language elements. $[\cdot]$ is also a mandatory notational element to encapsulate a formula following a temporal operator.

[6] The availability of bounded, particularly lower-bounded, operators depends on the stochastic model, property, and tool used. However, the use of upper time bounds is sufficient for our purposes.

As already hinted at, in tools such as PRISM, the concise construction of $\delta_{\mathcal{P}}$ (Definition 1), the behaviour of $\mathcal{P}$, is facilitated by a flavour of pGCL. Guarded commands have the form

$$[\underbrace{\alpha}_{\text{event}}]\ \underbrace{\gamma}_{\text{guard}} \longrightarrow \underbrace{pr_1 : \upsilon_1 \ + \ \ldots \ + \ pr_i : \overbrace{x'_{i,1} = u_{i,1} \ \& \ \ldots \ \& \ x'_{i,m_i} = u_{i,m_i}}^{\text{multiple assignment } \upsilon_i} \ + \ \ldots \ + \ pr_n : \upsilon_n}_{\text{probabilistic update } \upsilon} \tag{1}$$

with an event (or action)[7] label $\alpha \in A_{\mathcal{P}}$ and a probabilistic update υ applicable to $s \in S$ only if $s \models \gamma$, where γ is a propositional formula.[8] Update expressions have the form $\upsilon ::= pr_1 : \upsilon_1 + \cdots + pr_n : \upsilon_n$ with "+" for probabilistic choice, "&" to compose simultaneous assignments, $\Sigma_{i \in 1..n} pr_i = 1$, and υ_i being a multiple assignment $\upsilon_i ::= x'_{i,1} = u_{i,1}$ & ... & $x'_{i,m_i} = u_{i,m_i}$ to state variables $x_{i,j} \in X$ based on update functions $u_{i,j}$ (like above) for $i \in 1..n$, $j \in 1..m_i$. In an assignment, x' denotes the value of variable x after an update. When α is present, all modules comprising commands with α have to *synchronise*, that is, to perform all of these commands simultaneously. Unlike DTMCs, an MDP includes modules where multiple commands can be enabled in the same state (i.e., their γ's may overlap): these commands define the non-deterministic choices of the MDP. We denote by $\mathcal{M}[\mathcal{P}]$ the MDP model constructed from a pGCL program $\mathcal{P}$ applied to $s_0 \in S$. Moreover, $\mathcal{P} \setminus \mathcal{Q}$ denotes the pGCL program resulting from removing guarded commands of $\mathcal{Q}$ from $\mathcal{P}$ if they are syntactically equal and appear in the same modules occurring in both $\mathcal{P}$ and $\mathcal{Q}$.

MDP policy synthesis using probabilistic model checkers is only feasible for simple combinations of a few *constraints* (i.e., PRCTL properties with specified bounds) and *optimisation objectives* (i.e., PRCTL properties requiring the minimisation or maximisation of a probability or reward). For complex combinations like those encountered in our project, policy synthesis tools such as EVOCHECKER [17] and EVOPOLI [37] are needed. These tools employ multi-objective genetic algorithms to synthesise sets of MDP policies that satisfy multiple constraints and are Pareto-optimal with respect to multiple optimisation objectives.

3.4. Risk modelling for controller design

Risk assessment is about analysing how accidents (or *mishaps*) can occur and how likely they are to occur [38,39]. Having identified the causal chains of *critical events* leading to the mishaps (e.g., Table 1), one can design effective safety measures (Table 2). We are interested in identifying the *causes* of accidents, that is, events and states that when avoided will make an accident much less likely. To simplify accident prevention, causes are usually broken down into *causal factors*, each of which we can attempt to remove to avoid the cause. One of these factors represents the *hazard*. However, because of many reasons the hazard itself might often not be the (first) causal factor to be removed.

We assume an application, such as illustrated in Section 2, to be understood as a process $\mathcal{P}$. Furthermore, we assume that $\mathcal{P}$ includes a safety controller *sc* responsible for hazard mitigation and accident prevention. Based on the aforementioned concepts, risk modelling for $\mathcal{P}$ can be facilitated by specifying risk factors and combining them into *risk structures* [14].

At an abstract level, a *risk factor* f describes a critical event (e.g., one of the rows in Table 1) by a life cycle consisting of phases. We model f as a labelled transition system (LTS) with the states called *phases*. In its basic form, f uses the phases inactive ($\not f$), active (f), mitigated ($\bar{f}$), and mishap ($\underline{f}$). Transitions between these phases are labelled with endangerment events (e, e.g., the detection of a hazard or other causal factor) as well as mitigation (m) and resumption (r) actions (e.g., the mitigation of a hazard by removal of a causal factor). A risk structure from a risk factor set F (e.g., the column **Id** in Table 1) induces a *risk (state) space* $R(F) = \bigtimes_{\mathsf{f} \in F} Ph_{\mathsf{f}}$ by the Cartesian product of $Ph_{\mathsf{f}} = \{\not f, f, \bar{f}, \underline{f}\}$. The transitions describe how this risk space can be traversed. We assume factors and phases to be uniquely identified. Below, ρ as an index marks states as risk states, for example, $s_\rho \in R(F)$. Let $Ph_F = \bigcup_{\mathsf{f} \in F} Ph_{\mathsf{f}}$ be the disjoint union of the phase sets of F. Thus, we can interpret risk states as phase sets and obtain

$$[\![ph]\!]_{R(F)} = \{s_\rho \in R(F) \mid ph \in s_\rho\} \subseteq R(F)\ . \tag{2}$$

One can identify relationships between critical events using factor dependencies, such as requires or prevents [15]. Given factor sets $F_1, F_2 \subseteq F$, a factor dependency F_1 dep F_2 is a relation over $R(F)$ defined as

$$[\![F_1 \text{ dep } F_2]\!]_{R(F)\times R(F)} = \{(s_\rho, s'_\rho) \in R(F) \times R(F) \mid \text{a predicate } \phi(F_1, F_2, s_\rho, s'_\rho) \text{ describing dep holds}\}\ .$$

For example, the dependency F_1 requiresOcc F_2 with $F_1 \cap F_2 = \emptyset$, defined as

$$[\![F_1 \text{ requiresOcc } F_2]\!]_{R(F)\times R(F)} = \left\{(s_\rho, s'_\rho) \in R(F) \times R(F) \mid \exists \mathsf{f} \in F_1 : s'_\rho(\mathsf{f}) = f \Rightarrow \forall \mathsf{g} \in F_2 : s_\rho(\mathsf{g}) \in \{g, \bar{g}\}\right\},$$

[7] *Events* describe state transitions. If an event describes the effect of an actor's (e.g., safety controller's) action then we speak of it as an *action*. Compliant with [36], if an event describes a state change of a passive entity or an actor's observation, possibly altering its control state, then we still only speak of an event. An action is a specific event where a specific actor actually did something.

[8] We use $\longrightarrow$ to separate guard and update expressions and $\rightarrow$ both for logical implication and the definition of mappings.

describes that factors from F_1 can only be activated from risk states where all factors from F_2 have been activated. Such relationships can be justified by, for example, a hazard operability study, a failure mode effects analysis, or a fault tree analysis. A risk structure results from the constrained parallel composition of the factor LTSs from F and models a causal network (see, e.g., Fig. 7) describing mishaps, their causes, and causal factors (e.g., hazards). An F_1-requiresOcc-F_2 dependency then constrains the transition relation of the causal network such that no endangerments can occur (or are considered) from F_1 unless F_2 is active or mitigated. To remain within scope of this work, we refer the reader to our investigation of factor dependencies and composition in [14]. In Section 4.2.1 (Fig. 6b), we refine the basic factor notion as part of the contribution of this work.

At the level of MDPs, we use F as state variables with phases as the possible values. By $F \subset X$, we require at least one variable to exist in $X \setminus F$ through which the safety controller can observe the process. Then, $R(F)$ is naturally embedded into S resulting from the Cartesian product of the types of X. To combine $R(F)$ with the complementary state space $S|_{X\setminus F}$ of the MDP, for each phase ph, we specify an invariant $inv(ph)\colon S \to \mathbb{B}$ that defines, using variables in $X \setminus F$, when ph holds in a state $s \in S$. We require these phase invariants to form a partition of S. Consequently, each phase $ph \in Ph_{\mathsf{f}}$ of a factor $\mathsf{f} \in F$ carries a state proposition over AP with the interpretation

$$[\![ph]\!]_S = \{s \in S \mid s \models inv(ph) \ \wedge \underbrace{\qquad s(\mathsf{f}) = ph \qquad}_{\text{corresponds to } ph \in s_\rho \text{ for an } s_\rho \in [\![ph]\!]_{R(F)}} \} \subseteq S\,. \tag{3}$$

We conjoin the invariant with the risk state to use $R(F)$ as control states of the safety controller sc and phase invariants to define what sc needs to observe and when it needs to react to align its control state with these invariants. Then, the convention $R(F) = S|_F$ is meaningful. We generalise to $[\![Ph]\!]_* = \bigcup_{ph\in Ph} [\![ph]\!]_*$ for $Ph \subseteq Ph_F$ and $* \in \{S, R(F)\}$ and allow conventions such as $s \in ph$, $S \setminus Ph$, and $ph \subset R(F)$ to denote $s \in [\![ph]\!]_S$, $S \setminus [\![Ph]\!]_S$, and $[\![ph]\!]_{R(F)} \subset R(F)$ if clear from the context. Note that the semantics of $ph \in AP$ in a temporal logic formula implies $\{s \in S \mid ph \in L(s)\} = [\![ph]\!]_S$. We further define

$$[\![\Xi]\!]_S = \{s \in S \mid \exists \mathsf{f} \in F\colon s(\mathsf{f}) = f\} \subseteq S \tag{4}$$

to be the set of states labelled with at least one active factor, describing the abstract state Ξ where some cause or at least a causal factor has been sensed by the controller (e.g., HC with its handling about to start, i.e., the controller transitioning from ~~*HC*~~ to *HC*). We call Ξ the (non-accident) *F-unsafe region* of $R(F)$ respectively S. Moreover, *mishaps*

$$[\![\underline{f}]\!]_S = \{s \in S \mid \exists \mathsf{f} \in F\colon s(\mathsf{f}) = \underline{f}\} \subset S \tag{5}$$

are undesired states (e.g., all states where a person is injured by welding sparks). For a mishap $\underline{f} \in Ph_F$, we define

$$[\![\Xi_{\underline{f}}]\!]^{\mathcal{P}}_S = \{s \in S \mid \mathcal{M}[\mathcal{P}], s \models \mathbf{P}_{>pr}[\mathbf{F}^{\leq t}\, \underline{f}]\} \subset S \tag{6}$$

$\underline{f}$ is reachable from in a process $\mathcal{P}$, where pr and t are application-specific probability and time bounds to be defined based on empirical and expert knowledge. We call the state set $\Xi_{\underline{f}}$ the *cause*[9] of $\underline{f}$. Consider as an example for $\Xi_{\underline{f}}$ the set of all states in S where the operator is very close to the spot welder while the latter is active.

Formula (6) is agnostic of f's activation f, allowing causes to be identified before breaking them down into *causal factors* (see, e.g., Section 3.1 and Table 1). For controller design, one can then identify factors related to the subject of protection (e.g., the operator to be protected by the safety controller when being near the spot welder) and a *hazard* related to the system (e.g., the spot welder being active; [39,38]). We call causes latent or *controllable*[10] if there are sufficient resources to remove some of the identified factors to prevent the accident. For example, the larger t and the smaller pr in Formula (6), the larger is the state set $\Xi_{\underline{f}}$ and, thus, the flexibility in $\mathcal{P}$ for leaving f by transition to $S \setminus \Xi_{\underline{f}}$. Observe that the minimum requirement for every factor $\mathsf{f} \in F$ is that $[\![\Xi_{\underline{f}}]\!]^{\mathcal{P}}_S \subset [\![\Xi_{\underline{f}}]\!]^{\mathcal{P}\setminus sc}_S$, that is, a process $\mathcal{P}$ using a safety controller for f exhibits a strict (ideally, smallest possible) subset of the causes of a process without such a controller, $\mathcal{P} \setminus sc$. In other words, the control scheme in sc is capable of mitigating the causes.

In our example, controllability is justified by assuming that if the spark flow is low there is a small time span left for the spot welder to be stopped or the operator to be instructed to leave without leading to an accident. Note that $[\![\Xi_{\underline{f}}]\!]_S \subseteq [\![\underline{f}]\!]_S$. Following Formula (6), the controller sc should use f to also detect states in $S \setminus \Xi_{\underline{f}}$ being critical because certain events (e.g., an operator approaches the spot welder) cause a transition to $\Xi_{\underline{f}}$, and possibly $\underline{f}$, if f stays active, further conditions hold, and no safety measures are put in place promptly, especially by sc.

The input language of YAP [16,15] supports the modelling of risk structures and will be provided with an MDP semantics in Section 4. Based on the semantics of risk factors (Formulas (2) and (3)) and risk structures (Formulas (4), (5), and (6)), we

[9] For the sake of simplicity, we apply the informal notion of a cause as a governing sufficient intersection of factors to make an accident very likely, without establishing the formal counterfactual claim that without any of these factors the accident would not have happened.

[10] As opposed to immediate causes with limited or no risk handling controls.

will further formalise risk modelling. Particularly, in Section 4.2.1, we will discuss how to derive trigger conditions covering causes and extract causal factors from these conditions for the controller to check whether it was successful in removing a cause. A further formal investigation is, however, out of scope of this work.

3.5. Use case-based controller integration testing

Use case-based testing can be employed to check the correct integration of the synthesised concrete safety controller with its operational environment. Given a realistic platform (e.g., the DTF), an implementation, say *MCobot*, comprising a safety controller *sc* is integrated with a controlled process (e.g., robot and spot welder *rw*) and its environment (e.g., operator *op*, work cell).

Let a use case U be a reference scenario, formally, a path $U \in \mathit{Paths}(\mathcal{M})$. From $\mathit{Paths}(\mathcal{M})$, the set $\mathit{TPaths}(\mathcal{M})$ of all timed paths can be obtained by associating to each path $\omega \in \mathit{Paths}(\mathcal{M})$ all timestamps $ts\colon \mathit{Paths}(\mathcal{M}) \times \mathbb{N} \to \mathbb{Q}_{\geq 0}$ such that $ts(\omega, i) < ts(\omega, j)$ for all states s_i, s_j in ω with $i < j$. To obtain further scenarios similar to U, consider a *vicinity predicate* $\mathcal{V}_{\mathcal{M}}\colon \mathit{TPaths}(\mathcal{M}) \times \mathit{Paths}(\mathcal{M}) \to \mathbb{B}$, for example, (i) the minimal number of edits to translate two paths into one another is below a certain bound, (ii) the initial states of two paths only differ in variables $D \subset X$, (iii) two paths are interleavings of two sequential processes in $\mathcal{P}$ (e.g., *op*, *rw*), or (iv) two paths only differ in the time difference between certain pairs of states (e.g., *op* waits before doing something). These criteria can be used to derive a set of *timed paths in vicinity* $\mathcal{V}_{\mathcal{M}}$ *of* U as $\mathcal{T}_{\mathcal{V}}(U) = \{\omega \in \mathit{TPaths}(\mathit{MCobot}) \mid \mathcal{V}_{\mathit{MCobot}}(\omega, U)\} \subseteq \mathit{TPaths}(\mathit{MCobot})$. One way of validating $\mathcal{T}_{\mathcal{V}}(U)$ is to consider a misuse case *MU* that is close to U but not in $\mathcal{T}_{\mathcal{V}}(U)$.

For testing, our properties ϕ (e.g., Table 3, Properties (11) and (12)) given in PRCTL (Section 3.2) can be translated into the fragment of metric interval temporal logic (MITL) whose properties ϕ_{MITL} are formed by propositions over *AP* and by the MITL path operator $\mathbf{U}_\iota$ with a time interval $\iota \subset \mathbb{Q}_{\geq 0} \times \mathbb{Q}_{\geq 0}$. ι is required to be a non-empty convex set. We suppress it if $\iota = (0, \infty)$. $\mathbf{F}_\iota$ and $\mathbf{G}_\iota$ are defined in analogy to the abbreviations in Section 3.2. For $\omega \in \mathit{TPaths}(\mathcal{M})$, the satisfaction relation $\models_{\mathsf{MITL}}$ for $\phi_1 \mathbf{U}_\iota \phi_2$ is defined as

$$\omega \models_{\mathsf{MITL}} \phi_1 \mathbf{U}_\iota \phi_2 \iff \exists t \in \iota\colon \omega^t \models_{\mathsf{MITL}} \phi_2 \wedge \forall t' \in (0, t)\colon \omega^{t'} \models_{\mathsf{MITL}} \phi_1 \,.$$

For further details on MITL, refer to Alur et al. [40].

A translation of ϕ into ϕ_{MITL} is possible by removing branching operators **A** and **E** and replacing $\mathbf{U}^{[\sim t]}$ with $\mathbf{U}_\iota$. A positive test verdict for each $\omega \in \mathcal{T}_{\mathcal{V}}(U)$ is then achieved if $\omega \models_{\mathsf{MITL}} \phi_{\mathsf{MITL}}$. We then say that *sc*, as deployed on *MCobot*, *conforms* with $\mathcal{M}$ with respect to U, written $sc \approx_U \phi_{\mathsf{MITL}}$, if and only if $\mathcal{M} \models \phi \wedge \forall \omega \in \mathcal{T}_{\mathcal{V}}(U)\colon \omega \models_{\mathsf{MITL}} \phi_{\mathsf{MITL}}$. Moreover, we say that *sc* complies with $\mathcal{M}$ with respect to *MU* if and only if $\mathcal{M} \not\models \phi \wedge \neg\exists \omega \in \mathcal{T}_{\mathcal{V}}(MU)\colon \omega \models_{\mathsf{MITL}} \phi_{\mathsf{MITL}}$. Finally, we say that the controller implementation is $(U, MU, \mathcal{V}, \phi)$*-correct* if and only if $sc \approx_U \phi_{\mathsf{MITL}}$ and the closest *MU* we can find, with $sc \not\approx_{MU} \phi_{\mathsf{MITL}}$ and no overlap with U regarding $\mathcal{V}$, suggests implausible inputs to the DTF.

3.6. Digital twins

We focus on digital twins as a platform for controller deployment and integration testing. Digital twinning is a key Industry 4.0 technology for enabling industrial automation, smart processes, and process autonomy [41,42]. Together with the Internet of Things and machine-to-machine communication, digital twinning enables both the creation of a better data infrastructure and the adoption of smart manufacturing technologies. A digital twin provides greater access to data relating to, and control over, a physical system, and has particular value in the design, implementation, and evaluation of processes and safety controllers. Kritzinger et al. [43] and Tao et al. [44] define a digital twin as:

> "A digital twin is an integrated multi-physics, multi-scale, probabilistic simulation of a complex product and uses the best available physical models, sensor updates, etc., to mirror the life of its corresponding twin."

More specifically, a digital twin is a digital representation of a physical system that operates in parallel[11] with the real system. This concurrency can persist throughout the life-cycle of the physical system. Communication between the physical twin and its digital representation is bilateral, and as a result, both can be mutually informed by real-world or simulated sensor data, requested actions and decisions. As a means for safety verification and analysis, a digital twin presents: (i) A faithful representation of the process domain and state space, (ii) a means to interrogate and collect data that may not be readily available from the physical system (independent of hardware limitations), and (iii) an interface to the physical twin through which real-world responses to new safety procedures can be demonstrated.

4. Modelling for the synthesis of safety controllers

Fig. 3 provides an overview of the proposed approach to the synthesis of safety controllers. The main idea is that the control engineer designs a safety controller on top of an application, in this instance, comprising activities where humans

[11] A digital twin must be able to operate synchronously with the physical system, but asynchronous operation is also permissible.

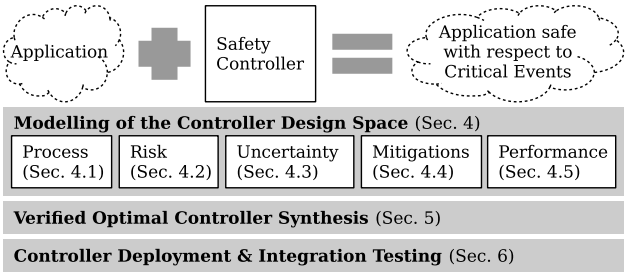


Fig. 3. Stages of the proposed safety controller design method.

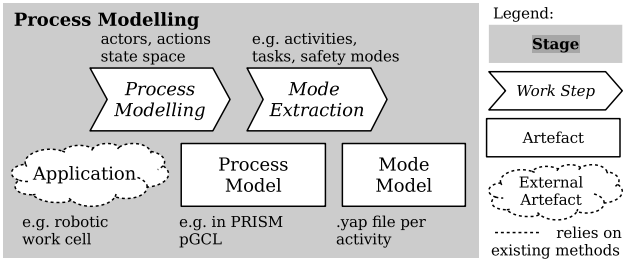


Fig. 4. Work steps and artifacts of the process modelling stage.

and robots collaborate. The intention of the deployed controller is to increase safety with respect to the critical events under consideration.

In the *modelling* stage, the control engineer creates several models to obtain a design space including controller candidates to select from during synthesis. When modelling the *process* of the application, the engineer as a domain expert describes the actions performed by any of the actors in the application. The engineer then performs a risk analysis resulting in a *risk model* that informs the process model with a notion of operational risk. The abstraction chosen for the model and a usual lack of knowledge about process details suggest the integration of *uncertain behaviour* and *performance estimates* into the model. With the risk- and performance-informed stochastic process model, the engineer can specify controller behaviour in form of a *mitigation* model. In the *controller synthesis* stage, an abstract controller is automatically synthesised from the design space according to risk- and performance-based optimality criteria. Finally, in the *deployment and testing* stage, this controller is translated automatically into an executable form. The three stages are supported by the tools YAP, PRISM, EVOCHECKER, and the Digital Twin Framework. These stages are detailed in the following sub-sections and illustrated with several examples from our case study introduced in Section 2.

4.1. Process modelling

For process modelling (Fig. 4), we use pGCL to specify the actions of the process. The process model defines the state space to be manipulated by these actions and includes the actions of the cobot and the environment including human operators. We group actions into *activities* as abstractions of the process. Activities describe certain tasks and facilitate the transition to risk modelling by structuring hazard identification and the creation of a hazard list. The *mode model* resulting from this abstraction step is an abstract LTS (see, e.g., Fig. 2b).

We describe the process, denoted $\mathcal{P}$, as a composition of guarded commands, informally distinguishing *actions* of relevant actors (e.g., a robot arm, a spot welder, an operator) and the safety controller from *events* of passive entities such as a sensor module and shared "manipulables" (e.g., workpiece support). The structure of the guarded commands describing the behaviour in $\mathcal{P}$ follows the pattern defined in Section 3.3. Below, we will use the Greek letter α for custom but unspecified event (or action) labels and the Greek letters γ, υ, and υ_i for custom but unspecified parts of guard expressions, probabilistic update expressions, and multiple assignments in a command. The state space S is built from discrete variables X (cf. Listing 1) capturing the world state (e.g., robot location, workbench status), sensory inputs (e.g., range finder), control outputs (e.g., robot behaviour, notifications), user inputs (e.g., start button), and modes (e.g., activities, safety modes, risk states).

4.1.1. Process execution from a controller's perspective

To account for the interaction of the safety controller with the process, we include a fair cyclic execution scheme into the process model. This scheme emulates the simultaneity of the controller and the process. Execution steps alternate between the controller and a set $\mathcal{A}$ of actors, ensuring in each cycle that each actor can take its turn (Fig. 5). In the running example

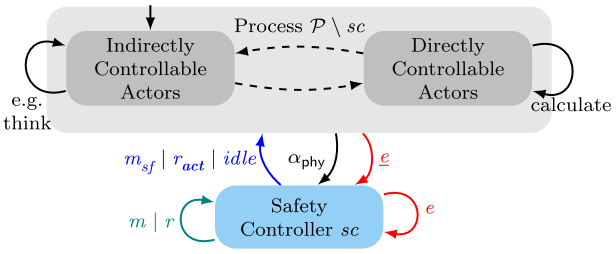


Fig. 5. Process execution from the viewpoint of the *sc* is coordinated by passing a single token *tn* between two groups of actors (dashed arcs) performing actions in the process. After every sequence of logical actions (e.g., think, calculate) ending with a physical action $(\alpha_{\mathsf{phy}}, \underline{e})$ of $\mathcal{P}$, the safety controller *sc* can observe endangerments (e), perform mitigations (m) and resumptions (r), and interact with the process (m_{sf}, r_{act}) or stay *idle*. These events will be explained in more detail in the subsections of Section 4.

below, we assume $\{sc, op\} \subset \mathcal{A}$. Regarding the way the safety controller *sc* can influence the process, we distinguish between directly controllable actors (e.g., cobot, spot welder) and indirectly or not controllable actors (e.g., human operator *op*).

Both kinds of actors can perform logical and physical actions. Logical actions $\alpha_{\mathsf{log}} \in A_{\mathcal{P}}$ are events describing changes of variables in X modelling logical states, for example, data states of the controller or mental states of humans. Physical actions $\alpha_{\mathsf{phy}} \in A_{\mathcal{P}}$ are events describing changes of variables in X modelling physical states, for example, the robot moves between two locations. The separation of these two kinds of variables and actions is to be made by the modeller and can be used to associate the passing of time only with physical actions while logical actions are assumed to take negligible time. This assumption helps to focus on relevant alternations between the *sc* and the process and on safety-critical transitions in $\delta_{\overrightarrow{\mathcal{P}}}$.

Logical and physical actions then follow the two corresponding command patterns:

$$[\alpha_{\mathsf{log}}]\ ok_p \wedge \gamma \longrightarrow \upsilon \qquad \text{(logical action)}$$

$$[\alpha_{\mathsf{phy}}]\ ok_p \wedge \gamma \longrightarrow \upsilon\ +\ \upsilon_i\ \&\ tn' = sc \qquad \text{(physical action)}$$

where ok_p guards the turn of an actor $p \in \mathcal{A}$ and *tn* stores the token passed between the safety controller and the other actors. $tn' = sc$ denotes the safety controller's (sc) turn. ok_p can include a condition for terminating execution when reaching a *final* (i.e., a goal or accident) state, resulting in $ok_p \equiv tn = p \wedge \neg final$. We define *final* to be

$$[\![final]\!]_S = \{s \in S \mid s \in \underline{F} \vee (s \notin \Xi \wedge \text{some application-specific goal condition})\}\ .$$

Note that $tn' = sc$ is part of the update expression of each physical action of any actor. Thus, the controller can most eagerly intervene, or stay idle, whenever one of the actors has completed an action. This scheme is more restrictive than the CSP-style[12] generalised parallel composition of concurrent processes available by default in tools such as Prism [18].

4.1.2. Activity and safety mode modelling

To facilitate the design of a powerful class of safety controllers, we organise actions $A_{\mathcal{P}}$ (e.g., grab workpiece, move robot arm to spot welder) of $\mathcal{P}$'s controllable actors $\mathcal{A}$ using special variables in X. Here, we use one variable per actor to model the engagement of that actor in an activity (e.g., $\mathsf{ract} \in X$) and $\mathsf{safmod} \in X$ for the safety mode of $\mathcal{P}$. These variables group actions and, from a controller perspective, allow high-level process control by filtering enabled actions. Thus, the structure of the guarded commands in $A_{\mathcal{P}}$ is refined according to the pattern

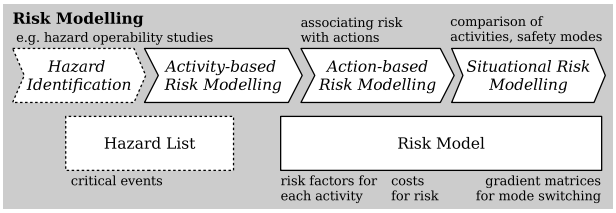
$$[\alpha]\ ok_p \wedge \gamma_{sm} \wedge \gamma_{act} \wedge \gamma \longrightarrow \upsilon\ +\ \upsilon_i\ [\&\, tn' = sc]$$

where γ_{act} guards the enabling of actions in certain activities (e.g., $\gamma_{act} \equiv \mathsf{ract} = exchWrkp$ checks whether the robot is in the "exchange workpiece" activity, Fig. 2b) and γ_{sm} in certain safety modes (e.g., $\gamma_{sm} \equiv \mathsf{safmod} = ssmon$ checks whether the "speed & separation monitoring" mode is activated). We obtain safety- and activity-aware actions by conjoining $\gamma_{sm} \wedge \gamma_{act}$. To allow atomic actor-internal sequences of updates, we leave the update of *tn* optional.

Example 1. The following listing describes part of the enumerations used to define the discrete state space S of $\mathcal{M}$.

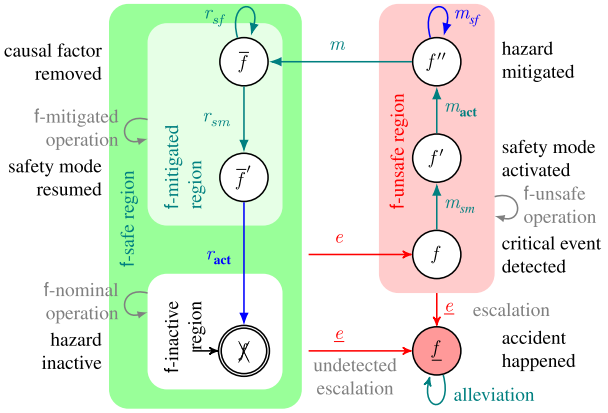
```
1 // spatial locations
2 const int atTable = 0; const int sharedTbl = 1; const int inCell = 2; const int atWeldSpot = 3;
3 // range finder signals
4 const int far = 0; const int near = 1; const int close = 2;
```

[12] Following the synchronous interleaving semantics of Hoare's Communicating Sequential Processes (CSP).



(a) Overview of the risk modelling stage

The states and regions of the LTS in (b) marked with circles and rectangular areas correspond to the refined and generic factor phases. The transitions are labelled with events and actions. Endangerment e (monitored) and accident $\underline{e}$ events; actions $m_\star$ and $r_\star$ of the safety controller with the actions m_{sf} and r_{act} where the controller interacts with the process waiting for a response; in gray, arbitrary process actions not controlled or observed by the safety controller. ▷



(b) Refined notion of a risk factor f.

Fig. 6. Risk modelling (a) based on refined risk factors (b).

```
5  // notification signals
6  const int ok = 0; const int leaveArea = 1; const int resetCtr = 2;
```

The following commands specify two actions, r_moveToTable and r_grabLeftWorkpiece, for the actor robotArm in the activity exchWrkp.

```
1  [r_moveToTable] OK_wc
2    & (safmod=normal|safmod=ssmon|safmod=pflim)
3    & ract=exchWrkp & (rloc != sharedTbl)
4    & (((wps!=right) & reffocc=1) | (wps=left & (reffocc=0)))
5    -> (rloc'=sharedTbl)&(turn'=sc);
6  [r_grabLeftWorkpiece] OK_wc
7    & (safmod=normal|safmod=ssmon|safmod=pflim|safmod=
       hguid)
8    & ract=exchWrkp & rloc=sharedTbl & reffocc=0 & wps=left
9    -> (reffocc'=1)&(wpfin'=0)&(wps'=empty)&(turn'=sc);
```

4.2. Risk modelling

For our synthesis approach to lead to correct and effective controllers, we need an expressive risk model (Fig. 6a). To obtain such a model, we translate the hazard list into a set of risk factors (Section 4.2.1). For this, we transcribe safety analysis results into factor LTSs (Section 3.4). Then, we define a risk profile for each of the process actions (Section 4.2.1).

For each hazard in the hazard list, the *risk profile of an action* describes the risk that a performance of the action results in an accident related to this hazard. The final step of risk modelling consists of capturing risk-related situational change (i.e., a change of activity or safety mode) in the process with a *risk gradient* (Section 4.2.2). For this, we associate a numerical measure with each activity transition (i.e., each situational change), that describes the change in overall risk. We proceed analogously with the definition of safety modes and the corresponding risk assessment.

4.2.1. Refined risk factors

Shown in Fig. 6b, we develop and use a refinement of the generic factor LTS described in Section 3.4. As before, for a factor $\mathsf{f} \in F$, we refer to the states of $\mathcal{M}$ where f is inactive as the *f-inactive region* ($[\![\not{f}]\!]_S$) or, by convention, as $\not{f}$. The states of $\mathcal{M}$, where f has occurred and the causal factors of f have not yet been sufficiently removed, are subsumed by the *f-unsafe region* ($[\![\{f, f', f''\}]\!]_S$). This region includes three phases: critical event e detected (f), safety mode sm activated (f'), and hazard mitigated (f''). The f-*mitigated region* ($[\![\{\bar{f}, \bar{f}'\}]\!]_S$) refers to states deviating from $\not{f}$ but with sufficiently many causal factors of f removed. This region includes the phase $\bar{f}$, reached when causal factors of f have been removed, and $\bar{f}'$, reached after deactivating a safety function and resuming from the current safety mode. The phases $\bar{f}$, $\bar{f}'$, and $\not{f}$ together constitute the f-*safe region* ($[\![\{\not{f}, \bar{f}, \bar{f}'\}]\!]_S$) of $\mathcal{M}$.

Note that f'' is still in the f-unsafe region because causal factors may not have been removed yet. Their removal may require interaction with the environment whose result sc is waiting for. Upon detection of their removal sc performs m to confirm the entrance into the f-mitigated region. Although f'' can be f-safe we declare it to be f-unsafe as long as the causal factor is still there. Resumption to nominal operation without causal factor removal could end up in a hazardous state leading sc to enter a mitigation loop and prevent progress.

Overall, we obtain the new phase set $Ph_\mathsf{f} = \{\not{f}, f, f', f'', \bar{f}, \bar{f}', \underline{f}\}$ replacing the LTS definition for a factor in Section 3.4. In particular, the definition of the F-unsafe region Ξ changes to

$$[\![\Xi]\!]_S = \bigcup_{\mathsf{f} \in F} [\![\{f, f', f''\}]\!]_S \subseteq S\,.$$

By lifting regions from factors to factor sets, we also obtain the F-inactive region 0 (see, e.g., Fig. 7)

$$[\![0]\!]_S = \bigcap_{\mathsf{f} \in F} [\![\not\mathsf{f}]\!]_S \subseteq S \,.$$

When a critical event e is detected by the *sc* in the f-safe region, the *sc* switches f to f. We distinguish e's ground truth predicate $\chi = inv(f)$ (i.e., the cause including the hazard and other causal factors) from its detector (or monitoring) predicate ζ (i.e., the sensor) where $\chi \Leftrightarrow \zeta$ in case of perfect sensing. *Critical event detection* follows the pattern

$$[e]\; ok_{sc} \wedge \zeta \wedge rel(act, \mathsf{f}) \wedge \neg(f \vee \underline{f}) \longrightarrow f$$

where $ok_{sc} \equiv tn = sc \wedge on_{sc} \wedge \neg final$, with a switch on_{sc} to turn the controller on and off, and $rel(act, \mathsf{f})$ is a relevance indicator determining whether to react on ζ in an activity $act \in Act$ or, more generally, in a particular situation. While factor dependencies such as requiresOcc (Section 3.4) put risk states in $R(F)$ into relation, at the pGCL level, they can be used to derive parts of ζ with the potential to reduce $\delta_{\mathcal{P}}$ to relevant controller interventions.

With $\neg(f \vee \underline{f})$, we ignore re-occurrences while f is active or after an accident. *Idling* of the safety controller (Fig. 5) is captured by

$$[idle]\; ok_{sc} \wedge \neg\chi \wedge (\not\mathsf{f} \vee \bar{f}) \longrightarrow tn' = next(p)$$

where $next(p)$ implements the execution scheme (Section 4.1.1, Fig. 5), that is, the handover from *sc* to the process and from actor $p \in \mathcal{A} \setminus \{sc\}$ to the next actor of $\mathcal{P}$. The other actions in Fig. 6b will be explained in Section 4.4.

Activity-based risk. Factor LTSs guide the formalisation of hazards, the associated causes, and mishaps and the events in the causal chain (e.g., a mishap event leads to a mishap state). This way, factors support the design of mitigations to reduce accidents, and alleviations to reduce consequences. Hence, all critical events related to an activity should be translated into a factor set. The composition of factor LTSs for an activity can be visualised as a *risk graph* (e.g., Fig. 7).

Action-based risk. We define an action multi-reward structure $rw^{\rho}_{\text{action}} \colon S \times A_{\mathcal{P}} \times F \to \mathbb{Q}_{\geq 0}$ over $\mathcal{M}$ to measure overall and factor-specific risk ρ in $\mathcal{P}$. Action rewards are guarded, requiring f being active. For example, if $rw^{\rho}_{\text{action}}(s, \alpha, \mathsf{f}) > 0$ then f is active in state s.

4.2.2. Situational risk

The decision space of the safety controller for mitigations and resumptions includes choices from sets of safety modes (SM) and activities (Act) to switch to from a particular safety mode and activity. To simplify the design space in $\mathcal{M}$, we resolve this choice during the generation of the pGCL program for *sc* (Section 5.1) using a *categorical risk gradient*

$$\nabla\rho = [\partial\rho/(x_1 \to x_1'), \ldots, \partial\rho/(x_n \to x_n')]^T$$

with $x_i \in \{act_p | p \in \mathcal{A}\} \cup \{sm\} \subset X$ for $i \in 1..n$. For categorical variables $x \in X$ (e.g., using enumeration types), we allow the convention $\partial y/(x \to x')$ denoting the change of y when x changes its value from x to x'.

We implement $\nabla\rho$ with two skew-diagonal matrices of type $\mathbb{Q}^{|SM|\times|SM|}$ and $\mathbb{Q}^{|Act|\times|Act|}$, assuming that these matrices can be derived from safety analysis based on the following justification. For example, assume that, between the activities $act, act' \in Act$, an actor varies in physical movement, force, and speed. If act involves more or wider movement, higher force application, or higher speed than in act', then a change from act to act' will likely reduce risk because of a reduced collision probability. We model risk reduction as positive changes to ρ and, hence, require $\partial\rho/(act \to act') \geq 0$. Similarly, assume that the safety modes $sm, sm' \in SM$ vary $\mathcal{P}$'s behaviour by relaxing or restricting the range of physical movement, force, and speed of the enabled actions in $A_{\mathcal{P}}$. If sm relaxes these parameters more than sm', then a change from sm to sm' will likely reduce risk. Again, $\partial\rho/(sm \to sm') \geq 0$. As we model endangerments as negative changes to ρ, skew-diagonality of the matrices provides that $\partial\rho/(sm' \to sm) \leq 0$.

The matrices for $\nabla\rho$ are assumed to be provided by the modeller and can be (manually) specified as part of a YAP model. Note that $\nabla\rho$ is only used for the generation of the pGCL program for *sc* (Section 5.1), prior to optimal policy synthesis for $\mathcal{M}$ based on S and $\delta_{\mathcal{P}}$ (Section 5.2). Hence, $\nabla\rho$ is an approximation of risk expressed only in terms of $S|_{\{act_p | p \in \mathcal{A}\} \cup \{sm\}}$.

Example 2. With $F = \{\mathsf{HC}, \mathsf{HS}, \mathsf{HRW}\}$ from Table 1, we obtain three instances of Fig. 6b. The risk graph in Fig. 7 provides a concise visual abstraction of $R(F)$ for our case study by only showing the combinations of regions indicated in Fig. 6b. Let us focus on HC and assume to be in the risk state $HS \in R(F)$. When an operator approaches an active spot welder, an event e^{HC} is detected ($HCdet$), activating HC by a transition to the state $HSHC$ where the predicate HC holds. Note that $HSHC \in [\![HC]\!]_R \subset R(F)$. The safety controller will then start with performing mitigations to reach phase $\overline{HC}$ (or state $HS\overline{HC}$). The handling of HC includes the option $HCSrmstIdleVis$ of switching to a safety-rated monitored stop mode, issuing a visual operator notification, and waiting for the operator's response. From $HS\overline{HC}$, the controller can continue with resumptions ($HCres$, e.g., switching from safety-rated monitored stop back to normal) to finally return to phase $\not HC$ (or state HS) where both HC and $\overline{HC}$ are false. Not shown in Fig. 7: Implemented in the *sc*, endangerments (e.g., reentry

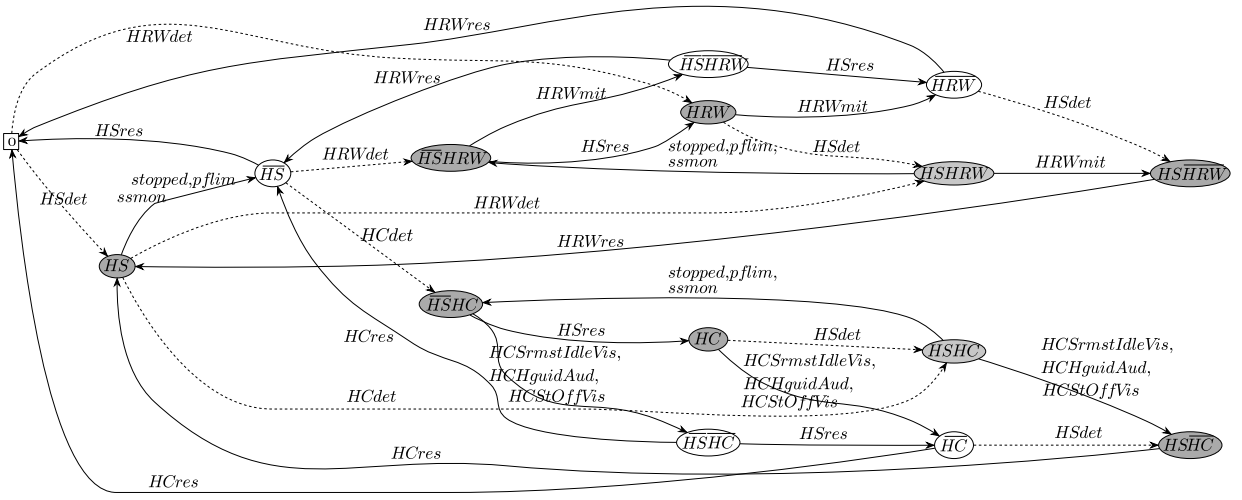


Fig. 7. Risk graph for Example 2 from factors HC, HRW, and HS, with 15 risk states, serving as a specification to be refined by a synthesised controller. Dependencies "HC prevents HRW" and "HRW prevents HC" encode the assumption of one operator in the work cell. Detectors *HSdet* or *HRWdet* (dottet arcs) instantiate endangerments, and *stopped*, *pflim*, *ssmon*, and *HSres* exemplify mitigations and resumptions (solid arcs). The more darkly shaded a risk state, the more dangerous it is, qualitatively.

of the operator) may reactivate HC from phase $\overline{HC}$ or $\overline{HC}'$. Furthermore, an accident $\underline{e}^{\mathsf{HC}}$ with the spot welder or robot arm, leading to phase $\underline{HC}$, can occur, for example, because of a faulty range finder responsible for the detection of e^{HC} or too slow mitigations m^{HC}_{sm} and m^{HC}_{act}. In $\underline{HC}$, alleviations (e.g., flexible robot surfaces, protective goggles) can reduce certain consequences. HC is encoded in a YAP model as shown below.

```
1 HC desc "(H)uman (C)lose to active spot welder and cobot working"
2   requiresOcc (HS) // imposes relationship between cause/guard conditions of HS and HC, e.g. not HS => not HC
3   mitPreventsMit (HS)
4   guard "hSM_PERM & hACT_WELDING & hloc=atWeldSpot"
```

As a factor dependency, we identify HC requiresOcc (HS), expressing the assumption that the factor HS must have occurred prior to the activation of HC. With this dependency, the controller will not detect *HS* (*HCdet*) in the F-inactive region 0 (cf. Fig. 7). Using the guard directive, we specify the cause χ, the ground truth predicate for the activation of HC.

The risk profile for the robot arm actions and the factors HC and HS is encoded in the YAP model in an intuitive and compact manner.

```
1                          guard  risk_HC  risk_HS;
2  // actor: robotArm
3  r_moveToTable:          ""     "5"      "9";
4  r_grabLeftWorkpiece:    ""     "0"      "3";
5  r_placeWorkpieceRight:  ""     "0"      "3";
6  r_moveToWelder:         ""     "9"      "5";
```

Based on the activity automaton in Fig. 2b, we encode $\nabla\rho$ in our YAP model with the two distance matrices act and safmod.

```
1  distances act {
2    off:       0;
3    idle:      1 0;
4    exchWrkp:  3 2 0;
5    welding:   5 4 2 0;
6  }
7
8
9  distances safmod {
10   normal:  0;
11   hguid:   −2 0;
12   ssmon:   −1 1   0;
13   pflim:   −2 0   −1 0;
14   srmst:   −3 −1 −2 −1
               0;
15   stopped: −4 −2 −3 −2
                 −1 0;
16 }
```

A key feature of our approach is that the synthesised safety controllers are required to be refinements of their associated risk structure. The risk graph in Fig. 7 represents a specification of the controller design space. Furthermore, engineers may want to use risk graphs for the analysis or debugging of risk structures.

4.3. Modelling stochastic adversarial phenomena

In this stage (Fig. 8), we integrate adversarial stochastic phenomena into the process model. Probabilistic choice in $\mathcal{M}$ can be used to model various phenomena, such as accidents, human error, and sensor failure.

Mishaps. In the refined factor model (Fig. 6b), a mishap $\underline{e}$ leading to phase $\underline{f}$ is always possible but assumed to happen much more likely in the f-unsafe region than in the f-safe region. Each physical action α can be associated with an *accident-prone physical action* $\underline{e}_\alpha$ describing how $\underline{e}$ can happen. α and $\underline{e}_\alpha$ can be specified by following the two command patterns:

$$[\alpha]\, ok_p \wedge \gamma \wedge \neg\chi \longrightarrow \upsilon \,\&\, tn' = sc$$

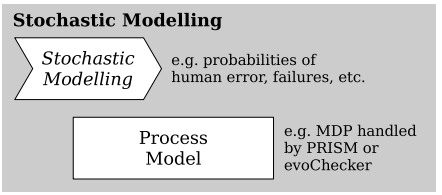


Fig. 8. The stochastic modelling stage.

$$[\underline{e_\alpha}]\, ok_p \wedge \gamma \wedge \chi \wedge \neg \underline{f} \longrightarrow pr_{\underline{f}} \colon \underline{f} \,\&\, \upsilon_{acc} \,\&\, tn' = sc \;+\; (1 - pr_{\underline{f}}) \colon \upsilon_{nom} \,\&\, tn' = sc$$

with the probability $pr_{\underline{f}}$ of a mishap if the cause χ of the critical event has occurred, independent of whether or not f's activation (e) was detected. υ_{acc} and υ_{nom} can include specific updates if an accident occurs respectively if it does not. $pr_{\underline{f}}$ can be inferred from observations, experiments, or accident statistics. We keep using the Greek letters γ and υ for action-specific preconditions respectively updates.

Human error. A human error model can be informed by hierarchical task analysis [45]. We introduce a particular class of human errors into $\mathcal{M}$ using the pair of probabilistic commands

$$[\alpha_{\mathsf{log}}]\, ok_{op} \wedge \neg dn \wedge \gamma \longrightarrow (sp?1 : pr_{he}) \colon dn' = true \;+\; (sp?0 : 1 - pr_{he}) \colon tn' = sc$$
$$[\alpha_{\mathsf{phy}}]\, ok_{op} \wedge dn \wedge \gamma \longrightarrow \upsilon \,\&\, dn' = false \,\&\, tn' = sc$$

with a predicate *sp* specifying when action α_{phy} is safe or permitted to be performed, the probability pr_{he} of an operator to commit a specific error when this action is not safe or not permitted, and a deontic flag *dn* controlling whether the logical action α_{log} is to be realised by the physical action α_{phy} with a potentially dangerous update υ.

Sensor failure. Informed by a fault tree or failure mode effects analysis, one can consider sensor and actuator faults in a way similar to human error. To model *fault behaviour*, we employ the pattern

$$[\alpha]\, ok_p \wedge \gamma \longrightarrow (1 - pr_s) \colon \upsilon_{corr} \,\&\, tn' = sc \;+\; pr_s \colon \upsilon_{fail} \,\&\, tn' = sc$$

with a probability pr_s of a sensor failing to detect a specific event implied by γ, an update υ_{corr} modelling the correct behaviour of the sensor, and an update υ_{fail} modelling its failure behaviour. See Example 3.

Example 3. We model the accident that, with a 20% chance, $\underline{HC}$ follows *HC*' (i.e., HC remains undetected because of a sensor fault) or *HC* (i.e., the safety controller is not reacting timely). For the encoding of HC, YAP's input language supports the specification of the actions with the mishap $\underline{HC}$ as a bad outcome if *HC* is undetected or not mitigated timely, the probability of $\underline{HC}$ under these conditions, and the severity of the expected consequences from $\underline{HC}$. Furthermore, we model the human error that, with a 10% chance, the operator enters the cell, knowing that the robotArm and the spotWelder are active. Finally, we specify as a sensor failure that the range finder as the detector of e^{HC} fails in 5% of the cases when the operator enters the cell.

The explained command patterns can be combined and provide an MDP semantics for risk factors, offering degrees of modelling freedom not further discussed here. For practical examples, see also [16].

4.4. Mitigation modelling

For mitigation modelling (Fig. 9), we complete the factor LTSs introduced in Section 4.2.1 with mitigation actions. The ability of stochastic models, such as MDPs, to express nondeterminism supports the modelling of alternative *mitigation options*. To extend the controller design space, we can thus specify several such options for each factor. Differences in the quality of these options (e.g., expected nuisance and effort) can be quantified using reward structures.

The capabilities of actors in $\mathcal{P}$ determine the controllability of critical events. To restrict the controller design space, we allow three kinds of actions: *action filters* (i.e., safety modes, cf. Section 1), *activity changes* (e.g., change from welding to off), and *safety functions* (e.g., interacting with the operator through warnings). These mitigations are mirrored by corresponding resumptions. We continue with our discussion of how mitigation and resumption actions, according to the refined factor LTS in Fig. 6b, are translated into non-probabilistic pGCL commands of the form summarised by Formula (7).

$$[\underbrace{\text{controller action}}_{\text{event}}]\ \underbrace{\text{process state} \wedge \text{risk state}}_{\text{guard}} \rightarrow \underbrace{\text{safety mode \,\&\, activity switch, safety function}}_{\text{update}}\ . \tag{7}$$

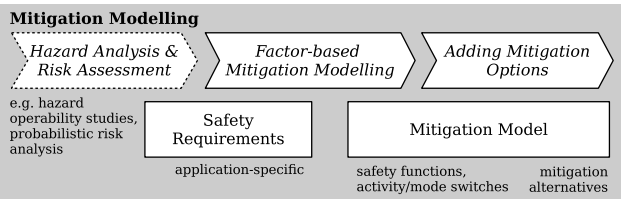


Fig. 9. Overview of the mitigation modelling stage.

Let $\mathsf{f} \in F$ be the risk factor under consideration for the rest of this section. Given the ground truth predicate χ and the corresponding detector predicate ζ for f (Section 4.2.1), we assume to have identified a causal factor κ such that $\neg\kappa \Rightarrow \neg\chi \wedge \neg\zeta$ (i.e., an absent causal factor eliminates the cause), $\chi \Rightarrow \kappa$ (i.e., κ is indeed a causal factor), and $\zeta \Rightarrow \kappa$ (i.e., the detection of the cause implies the presence of the causal factor). Factor dependencies, such as requiresOcc in Example 2, can be used to automatically derive part of ζ (cf. YAP, [16]).

Let $\mathcal{T}_{Act} \subseteq Act^{|\mathcal{A}|}$ be the activity tuples, that is, reasonable combinations of activities the actors in $\mathcal{A} \setminus \{sc\}$ can be involved at a particular point in time. Let a *mitigation option* $(\mathbf{act}_{tgt}, sm_{tgt}, sf) \in \mathcal{T}_{Act} \times SM \times SF$ for f with a target activity[13] $\mathbf{act}_{tgt}$, a target safety mode sm_{tgt}, and sf picked from a set SF of safety functions. For each combination $(sm_{cur}, \mathbf{act}_{cur}) \in SM \times \mathcal{T}_{Act}$ possible in a state $s \in S$, the controller provides a *safety mode switch*

$$[m_{sm}]\ ok_{sc} \wedge f \wedge sm_{cur} \longrightarrow sm_{new}\ \&\ f'$$

and an *activity tuple switch*

$$[m_{\boldsymbol{act}}]\ ok_{sc} \wedge f' \wedge \mathbf{act}_{cur} \longrightarrow \mathbf{act}_{new}\ \&\ f''$$

where $\langle x\rangle_{new}$ with $\langle x\rangle ::= sm \mid \mathbf{act}$ is determined according to the scheme

$$\langle x\rangle_{new} = \begin{cases} \langle x\rangle_{tgt}, & \text{if } \partial\rho/(\langle x\rangle_{cur} \rightarrow \langle x\rangle_{tgt}) \geq 0 \\ \langle x\rangle_{cur}, & \text{otherwise} \end{cases} . \tag{8}$$

We use the non-strict order $\geq$ because a switch to a desired target with the same risk should be allowed. Then, the controller activates a *safety function* sf (modelled both as a propositional formula and an update) through the commands

$$[m_{sf}]\ ok_{sc} \wedge f'' \wedge \kappa \wedge \neg sf \longrightarrow sf\ \&\ tn' = next(p)$$
$$[m_{sf}]\ ok_{sc} \wedge f'' \wedge \kappa \wedge sf \longrightarrow tn' = next(p) \quad .$$

Reaching phase f'' constitutes the first set of logical controller actions. The performance of these actions is followed by an interaction with the process through passing the turn to the next actor p with $tn' = next(p)$. If this interaction results in the elimination of κ, the controller finalises the mitigation stage with the command

$$[m]\ ok_{sc} \wedge f'' \wedge \neg\kappa \longrightarrow \bar{f}\,.$$

The controller then moves to the resumption stage, continuing with the *deactivation of the safety function* (sf^{-1}) through

$$[r_{sf}]\ ok_{sc} \wedge \bar{f} \wedge \neg\kappa \wedge sf \longrightarrow sf^{-1}$$

and the *resumption from the current to a more progressive safety mode* from any risk state $s_\rho \in R(F)$ by

$$[r_{sm}]\ ok_{sc} \wedge \bar{f} \wedge \neg\zeta \wedge \neg\kappa \wedge s_\rho \wedge sm_{cur} \wedge \neg sf \longrightarrow sm_{new}\ \&\ \bar{f}'\ .$$

Beyond its basic enabling condition $(ok_{sc} \wedge \bar{f})$, the controller checks whether both the cause of the critical event and, particularly, the causal factor subject of mitigation have been removed $(\neg\zeta \wedge \neg\kappa)$.

Analogously, the *resumption of the current activity to a more productive activity* from any risk state $s_\rho \in R(F)$ is accomplished with

$$[r_a]\ ok_{sc} \wedge \bar{f}' \wedge \neg\zeta \wedge \neg\kappa \wedge s_\rho \wedge \mathbf{act}_{cur} \wedge \neg sf \longrightarrow \mathbf{act}_{new}\ \&\ \not{f}\ \&\ tn' = next(p)\ .$$

Given a *resumption option* $(\mathbf{act}_{tgt}, sm_{tgt}) \in \mathcal{T}_{Act} \times SM$ and the set $am(s_\rho) \subseteq F$ of factors active or mitigated in risk state s_ρ, the reaction of the controller follows the scheme

[13] Our tooling currently only supports a single activity as the target, hence, the tuple $\mathbf{act}_{tgt}$ contains the same element in each position.

$$(\mathbf{act}_{new}, sm_{new}) = \arg_{(\mathrm{f}.act_{tgt}, \mathrm{f}.sm_{tgt})} \max_{\mathrm{f} \in am(s_\rho)} \left\{ \nabla\rho \mid \nabla\rho < [\mathbf{act}_{tgt}, sm_{tgt}]^T \right\}. \tag{9}$$

Formula (9) determines the most permissive (arg max) yet allowed ($\nabla\rho < [\mathbf{act}_{tgt}, sm_{tgt}]^T$) combination of activity and safety mode ($\mathrm{f}.\mathbf{act}_{tgt}, \mathrm{f}.sm_{tgt}$) to switch to among all factors ($\mathrm{f} \in am(s_\rho)$) activated or mitigated in s_ρ. ($\mathrm{f}.\mathbf{act}_{tgt}, \mathrm{f}.sm_{tgt}$) is the combination with the maximum acceptable risk gradient ($\nabla\rho < [\mathbf{act}_{tgt}, sm_{tgt}]^T$) as seen from ($\mathbf{act}_{cur}, sm_{cur}$) and s_ρ according to $\nabla\rho$.

Let the set $A(\mathrm{f})$ be the repertoire of process events and controller actions for a factor f following the patterns explained above. In order for the controller to be able to safely deal with a factor set F, we assume that each of the controller actions is idempotent. Note how phase indicators (e.g., $\bar{f}$) ensure that the controller is performing in the right context (e.g., if $s_\rho \in [\![\bar{f}]\!]_R \subset R(F)$). The discussion of alleviations is out of scope of this synthesis approach. Example 4 below illustrates the description of the non-probabilistic controller commands defined above in YAP.

Example 4. Continuing with Listing 2, we specify non-probabilistic actions in the YAP model as shown below for the factor HC with three mitigation and two resumption options.

```
1   detectedBy (.HCdet)
2   mitigatedBy (.HCHguidAud,.HCStOffVis,.HCSrmstIdleVis)
3   resumedBy (.HCres,.HCres2) ...;
4
5  mode HCdet desc "light barrier"
6   guard "hSM_PERM & hACT_WELDING & rngDet=close";
7  mode HCStOffVis desc "emergency stop / cobot+welder turned off / visual notif."
8   cf "hST_HOinSGA" // causal factor
9   update "(notif'=leaveArea)" // safety function
10  target (act=off, safmod=stopped); // target activity / safety mode
11 mode HCSrmstIdleVis desc "safety−rated mon. stop / cobot+welder idle / visual notif."
12  cf "hST_HOinSGA" update "(notif=leaveArea)" target (act=idle, safmod=srmst);
13 mode HCStOffAud desc "emergency stop / cobot+welder turned off / audio−vis. notif."
14  cf "hST_HOinSGA" update "(notif=leaveArea)" target (act=off, safmod=stopped);
15 mode HCHguidAud desc "hand−guided welding / moderate vis. notif."
16  cf "hST_HOinSGA" update "(notif=leaveArea)" target (safmod=hguid);
17 mode HCres desc "exchange workpiece and start over"
18  cf "hST_HOinSGA"
19  update "(notif'=ok)" // deactivate safety function
20  target (act=exchWrkp, safmod=normal);
21 mode HCres2 desc "continue welding if workpiece still unfinished"
22  cf "hST_HOinSGA" update "(notif=ok)" target (act=welding, safmod=normal);
```

The directive detectedBy defines the sensor predicate ζ, stating that "the human operator is in the safeguarded area" (hST_HOinSGA). The factor attribute mitigatedBy associates HC with three mitigation options, and the attribute resumedBy with two resumption options. For example, in the action HCStOffVis, (i) update models a safety function, issuing a notification to the operator to leave the safeguarded area, and (ii) target switches the manufacturing cell to the activity off and to the safety mode stopped,[a] all triggered by the range finder (rngDet=close). The guard and detectedBy attributes are translated into a pair of predicates for $\mathcal{M}$, RCE_HC (χ) describing world states, and CE_HC (ζ) signifying states monitored by the range finder.

```
1  // HC:monitor "(H)uman (C)lose to active spot welder and cobot working"
2  formula CE_HC = (hSM_PERM & hACT_WELDING & rngDet=close) & (HSp=act | HSp=mit1 | HSp=mit2 | HSp=mit | HSp=res) & (HRWp!=
       act);
3  formula RCE_HC = (hSM_PERM & hACT_WELDING & hloc=atWeldSpot) & (HRWp!=act);
```

[a] In a design variant discussed in [12], we allow mitigations to synchronise with the robotArm and spotWelder on an event stop.

4.5. Performance modelling

In analogy to the rewards for mitigation actions, in this stage (Fig. 10), we quantify performance (e.g., effort, productivity) for all actions of $\mathcal{P} \setminus sc$. As a result, optimal policy synthesis from $\mathcal{M}$ is based on several reward structures quantifying the performance of both the safety controller and the process.

For controller performance, we distinguish mitigation and resumption options by manually estimated quantities such as *disruption* of the manufacturing process, *nuisance* of the controller to the operator, and resources (e.g., *effort*, *time*) consumed by the controller. We formalise these quantities as action rewards rw^q_{action} with $q \in \{\text{disr}, \text{nuis}, \text{eff}, \text{time}\}$. As shown in Example 5, rewards can depend on parameters other than state variables. Analogously, concerning process performance, we

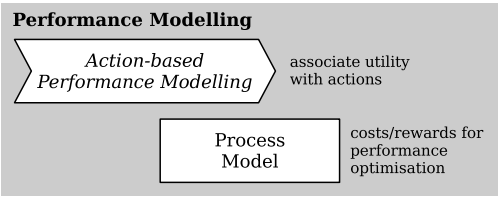


Fig. 10. Procedure for performance modelling.

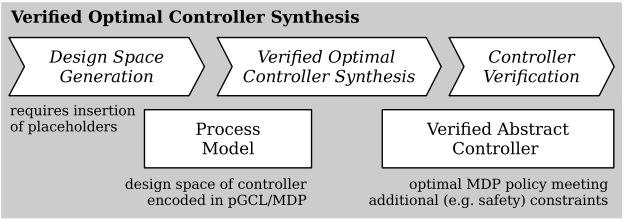


Fig. 11. Overview of the controller synthesis stage.

associate with each process action a *productivity* measure depending on the safety mode, using an action reward structure $rw^{prod}_{\text{action}}$.

The separate consideration of process and *sc* performance allows one to separately quantify and optimise environment behaviour and safety control. For example, one can maximise productivity of the cobot while minimising disruption by the *sc*. Furthermore, one can model worst-case assumptions about operator behaviour by maximising the use of the *sc* (i.e., its risk-reduction potential) while maximising in $\mathcal{M}$ the probability of finishing the task.

Example 5. In a YAP model, mitigation and resumption options can be associated with action rewards:

```
1 mode HCStOffVis ...
2   disruption=9 nuisance="alarmIntensity1 * 5"
3   effort=5.5; // wear/tear intensive, maintenance effort, energy
4 mode HCSrmstIdleVis ...
5   disruption=7.5 nuisance="alarmIntensity1 * 6" effort=6.5;
6 mode HCStOffAud ...
7   disruption=10 nuisance="alarmIntensity1 * 9" effort=5;
```

Note how nuisance depends on the parameter alarmIntensity1 modelling the loudness or brightness of an alarm sound or lamp that can be varied in the search for an optimal controller. Rewards for process actions can be specified in a concise manner in a YAP model.

```
1                          guard_prod      prod;
2 // actor: robotArm
3 r_moveToTable:           ""              "h";
4 r_grabLeftWorkpiece:     ""              "m";
5 r_placeWorkpieceRight:   ""              "h";
6 r_moveToWelder:          ""              "h";
```

5. Verified synthesis of safety controllers

In this stage, we integrate the risk and mitigation models with the process model, resulting in a risk-informed reward-enhanced stochastic model that includes the *controller design space* in the process decision space (Fig. 11). The action sets for the cobot, the operator, and the controller are now combined in $A_{\mathcal{P}}$. With this integrated model $\mathcal{P}$, we perform a constrained policy synthesis to select an optimal yet abstract safety controller from the design space. We use constraints to encode the *safety requirements* and *optimisation problems* to facilitate this selection. For this to work, we express safety requirements as PRCTL properties and verify them using a stochastic model checker. We accomplish controller synthesis in **two settings**.

The MDP setting. We perform optimal policy synthesis from an MDP (using PRISM) where the design space is encoded as non-deterministic choice (e.g., among mitigation options). This approach has already been discussed in [12].

The parametric DTMC (pDTMC) setting. We perform an evolutionary search (using EvoChecker) of a set of DTMCs. This set defines the design space by fixing the parameters of a pDTMC. This pDTMC can be obtained from the original MDP by replacing non-deterministic choice with random choice and by introducing the corresponding parameters. In both cases, optimal policy synthesis produces a DTMC containing an abstract discrete-event safety controller. The pDTMC-based approach avoids the split into two verification stages as previously required in [12].

The synthesis follows a two-staged search through the controller design space: The first stage focuses on the generation of the guarded commands according to Section 4.4. The gradient $\nabla\rho$ (Section 4.2.2) resolves the calculation of risk-minimal control updates for these commands. Reward structures are generated for risk and performance quantification (Section 4.5) in the second stage. Then, a stochastic model checker performs verified policy synthesis for $\mathcal{M}$. In [12], we use $\mathcal{M}$ as an MDP for policy synthesis with Prism [18] and extract a controller from the resulting DTMC representing the policy.

Here, we enhance this approach. The flexible CSP-style concurrency of the pGCL modules used in [12] is replaced in $\mathcal{P}$ by a more restrictive alternation (Section 4.1.1) of the process actors and the safety controller, thereby resembling a closed-loop control scheme. The avoidance of interleaving and synchronous events for modelling real-time phenomena (e.g., sensor faults) results in an MDP that does not contain states where process actors and the controller compete in non-deterministic choices. The only choices left are actor-internal choices including the choice to idle and pass the token. As a consequence, we obtain *fairness* for the controller and a simplification of $\mathcal{M}$. Additionally, we interpret $\mathcal{M}$ as a pDTMC rather than an MDP. Choice in the safety controller among mitigation and resumption options is explicitly controlled through decision parameters.

This scheme results in the removal of non-deterministic choice from the controller and a randomisation of residual choice in $\mathcal{P}$. Importantly, choice in the environment now remains in each policy as uniformly distributed probabilistic choice, resulting in controllers able to deal with a variety of environments. We further improve the way how accidents can happen in $\mathcal{M}$. For fine-granular risk and performance quantification, we allow additional design space parameters (e.g., alarm intensity) to be used. We use EvoChecker [17] for the search of optimal controllers in the set of DTMCs defined by these parameters. The modelling, analysis, and pre-processing required for our approach in [12] and its enhancement described here are supported by the Yap tool [16].

5.1. Design space and reward structure generation

The design space is created by instantiating the command patterns of the generic factor LTS (Fig. 6b) described in Section 4.4. Algorithm 1 exemplifies the command generation for three of these patterns. The complete algorithm is implemented in Yap and produces a concrete pGCL program for *sc* included in $\mathcal{P}$ as the controller design space.

For example, safety mode mitigations m_{sm} (Lines 1-4) are generated for each safety mode $sm \in SM$, factor $\mathrm{f} \in F$, and mitigation option $ms \in AS_{\mathrm{f}}$ (Line 2). $A(ph) \subset A(\mathrm{f})$ identifies the set of mitigation options for a factor phase $ph \in Ph_{\mathrm{f}}$ (Line 2). The function CompCmd (Line 3) composes guard and update expressions and integrates these into controller commands compliant with the patterns in Section 4.4. The functions GradUpdM (Line 3) and GradUpdR (Line 15) implement Formula (8) respectively Formula (9) for controlling the updates of safety modes and activities. We omit the corresponding generation functions here. $\mathcal{T}_{Act}$ in Line 7 assures the generation of mitigations for relevant combinations of activities the actors in $\mathcal{A} \setminus \{sc\}$ can be involved at a particular time. Lines 4, 9, and 16 indicate the generated action set $A_{\mathcal{P}}$. The listing in Example 6 shows a fragment of the design space for our running example.

Algorithm 1 Controller design space generation.

```
1: function GenSafetyModeMitigations(F, SM))
2:     for all sm ∈ SM, f ∈ F, ms ∈ A(f) do
3:         m_sm ← CompCmd(f ∧ sm, GradUpdM(sm, ms))          ▷ create safety mode mitigation
4:         A_P ← A_P ∪ {m_sm}                                ▷ add mitigation command
5:     end for
6: end function
7:
8: function GenActivityMitigations(F, T_Act)
9:     for all act ∈ T_Act, f ∈ F, ms ∈ A(f') do
10:        m_act ← CompCmd(f' ∧ act, GradUpdM(act, ms))      ▷ create command
11:        A_P ← A_P ∪ {m_act}                               ▷ add mitigation command
12:    end for
13: end function
14: ...
15:
16: function GenModeResumptions(F, SM)
17:    for all s_ρ ∈ R(F), sm ∈ SM, f ∈ am(s_ρ), ms ∈ A(f̄) do
18:        (ζ, κ, sf) ← obtained from factor model for f
19:        r_sm ← CompCmd(f̄ ∧ ¬ζ ∧ ¬κ ∧ s_ρ ∧ sm ∧ ¬sf, GradUpdR(s_ρ, f, ms))   ▷ create safety mode resumption
20:        A_P ← A_P ∪ {r_sm}                                ▷ add command
21:    end for
22: end function
```

Example 6. pGCL fragment generated for the factor HC:

```
// Endangerments (monitor, extension point: sensor and monitoring errors)
[si_HCact] OK_S & !(HCp=act | HCp=mit1 | HCp=mit2 | HCp=mis) & wact=idle & ract=exchWrkp & CE_HC −> (HCp'=act); ...
// Change of safety modes
[si_HCSrmstIdleVissafmod] OK_S & HCp=act & dpHCmit=HCHCSrmstIdleVis & safmod=normal −> (safmod'=srmst)&(HCp'=mit1); ...
// Frame switches
[s_HChalt] OK_S & HCp=mit1 & dpHCmit=HCHCSrmstIdleVis & wact=welding & ract=exchWrkp −> (wact'=idle)&(HCp'=mit2); ...
// Execution of safety functions
[si_HCSrmstIdleVisfun] OK_S & HCp=mit2 & dpHCmit=HCHCSrmstIdleVis & hST_HOinSGA & !(notif=leaveArea)
−> (notif'=leaveArea)&(token'=mod(token+1,ag))&(turn'=token+1); ...
// For entering the mitigated phase
[si_HCmit] OK_S & HCp=mit2 & dpHCmit=HCHCSrmstIdleVis & !(hST_HOinSGA) −> (HCp'=mit); ...
// Switching off safety functions
[si_HCres2fun] OK_S & HCp=mit & dpHCres=HCHCres2 & !(hST_HOinSGA) & (notif=leaveArea | notif=leaveArea | notif=leaveArea) −> (
    notif'=ok); ...
// Meta−policy for resuming to a less restrictive safety mode
[si_HCres2safmod] OK_S & HCp=mit & dpHCres=HCHCres2 & !CE_HC & !hST_HOinSGA & (HSp=mit|HSp=res) & (HRWp=mit|HRWp=res
    ) & safmod=normal & notif=ok −> (safmod'=pflim)&(HCp'=res); ...
// Resuming actor's activities
[s_HCresume2] OK_S & HCp=res & dpHCres=HCHCres2 & !CE_HC & !(hST_HOinSGA) & (HSp=mit|HSp=res) & (HRWp=mit|HRWp=res)
    & wact=welding & ract=exchWrkp −> (wact'=welding) & (ract'=welding) & (token'=mod(token+1,ag)) & (turn'=token+1) & (HCp'=inact);
    ...
```

Listing 7 shows a reward structure fragment generated from the YAP model in Sections 4.2.1, 4.2.1, and 4.5.

Example 7. The listing below shows two reward structure fragments, one for risk from an active HC and one for nuisance.

```
// Risk of HC−mishap when performing nominal action ...
rewards "risk_HC"
[rw_leaveWelder] !CYCLEEND & (RCE_HC | CE_HC) &
    safmod=pflim : 0.6 ∗ 5 ∗ 9.0;
[h_exitPlant] !CYCLEEND & (RCE_HC | CE_HC) & safmod=
    hguid : 0.4 ∗ 2 ∗ 9.0;
[r_moveToTable] !CYCLEEND & (RCE_HC | CE_HC) & safmod
    =hguid : 0.4 ∗ 5 ∗ 9.0;
...
endrewards
```

```
...
// Nuisance (e.g. to the human operator; per mitigation option)
rewards "nuisance"
[si_HCStOffVisfun] REWGUARD_HC : alarmIntensity1 ∗ 5 / 1;
[si_HRWmit2fun] REWGUARD_HRW : alarmIntensity2 ∗ 4 / 1;
[si_pflimfun] REWGUARD_HS : alarmIntensity1 ∗ 8 / 1;
...
endrewards
```

5.2. Verified optimal synthesis

The pGCL action system $\mathcal{P}$ consists of the process (i.e., cobot, welding machine, and operator) and the safety controller generated according to Section 5.1. The policy set $\Pi_{\mathcal{M}}$ includes the controller design space. $\mathcal{P}$ is expanded into an MDP by a probabilistic model checker such as PRISM or it is used as a pDTMC by EVOCHECKER relying on DTMC model checking. Choice in $\Pi_{\mathcal{M}}$ stems from commands (e.g., mitigations, resumptions) simultaneously enabled in a state $s \in S$, yielding multiple policies for s and from commands enabled in multiple states, giving rise to a policy for each ordering in which these commands can be chosen.

Controller solutions selected from the design space are subjected to two kinds of requirements. The first are optimisation objectives, such as minimal energy consumption. The second are probabilistic and reward-based constraints for safety (i.e., unlikely reachability of bad states, e.g., accidents below probability threshold; accumulated risk below reward threshold) and response (e.g., timely controller response exceeds probability threshold). Both kinds of requirements are expressed in PRCTL (Section 3.2).

Optimisation objectives. An optimal policy $\pi^\star \in \Pi_{\mathcal{M}}$, including the controller decisions, can be selected based on, for example, minimum nuisance or maximum productivity. For that, $\mathcal{M}$ includes action rewards to quantify controller and process performance (Section 4.5), risk reduction *pot*ential (in the **MDP setting**), and risk based on factors, modes, and activities as explained in Section 4.2.

Constraints. Constraints are of the form $\phi_{wf} \wedge \phi_{cor}$. ϕ_{wf} captures well-formedness, including properties for the verification of, for example, hazard occurrence and freedom from pre-*final* deadlocks, and properties for the falsification of, for example,

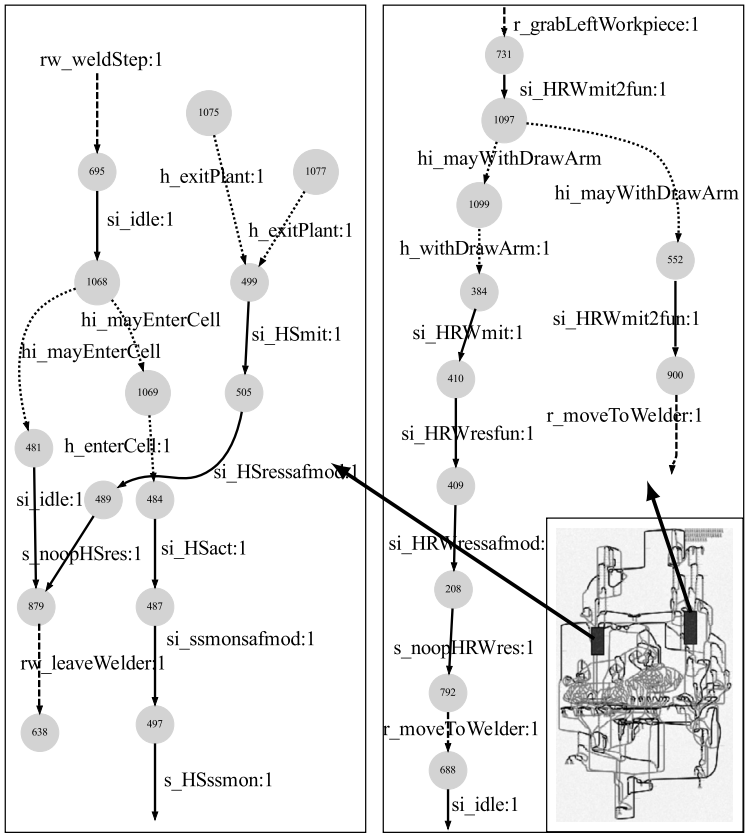


Fig. 12. Visualisation of an optimal policy $\pi^\star$ (lower right box) refining the risk graph in Fig. 7. The controller decisions (solid arcs), the spot welder and cobot actions (dashed arcs), probabilistic choices of the operator (dotted arcs). The two zoomed fragments highlight that mitigation of HS or HRW can deal with a variety of environments, for example, adversarial human decisions.

that final states must not be initial states.[14] Hazard occurrence fosters our focus on adversarial environments. Establishing $\mathcal{M} \models \phi_{wf}$ can help one to simplify model debugging, decrease model size, remove deadlocking states, and reduce vacuity of verification results.

ϕ_{cor} specifies safety-carrying correctness and can include progress, safety, liveness, and reliability properties, such as informally specified in Table 1. Let *goal* be the set of final states where the process under consideration has been successfully finished without accident, formally, $[\![goal]\!]_S = \{s \in S \mid s \in final \wedge \text{all tasks finished} \wedge s \notin \underline{F}\}$. When verifying *reach-avoid* properties we can check, for example, that in the process without the safety controller $(\mathcal{P} \setminus sc)$ the probability of avoiding the non-accident F-unsafe region Ξ (Section 3.4) until reaching the *goal* region (i.e., the end of the manufacturing cycle) stays below a probability bound pr, formally,

$$\mathcal{M}[\mathcal{P} \setminus sc] \models \mathbf{P}_{<pr}[\neg \Xi \, \mathbf{U} \, goal] \, .$$

Furthermore, we can check that in the process with the safety controller $(\mathcal{P})$ the probability of avoiding the accident region $\underline{F}$ until reaching the end of the manufacturing cycle stays below a probability bound pr, formally,

$$\mathcal{M}[\mathcal{P}] \models \mathbf{P}_{<pr}[\neg \underline{F} \, \mathbf{U} \, goal] \, .$$

We might also want to check that the probability of failure on demand of the controller or the probability of a mishap from any hazard is below a threshold. With

$$\mathbf{AG}(f \rightarrow \mathbf{P}_{>pr}[\mathbf{F}\bar{f}]) \wedge \mathbf{AG}(\bar{f} \rightarrow \mathbf{P}_{>pr}[\mathbf{F}f]) \, ,$$

[14] Well-formedness refers to properties (see, e.g., Table 3) to be checked to reduce basic modelling flaws and errors prior to verifying correctness properties specific to the application. Non-wellformed models can be difficult or impossible to verify or even deviate from reality.

Table 3
Optimisation problems to be solved over $\Pi_{\mathcal{M}}$ and properties to be checked of every $\pi \in \Pi_{\mathcal{M}}$.

Property†	Description
Single-objective optimisation problems	
$\mathbf{R}^{\text{eff}}_{\max=?}[\mathbf{C}]$	Assuming an adversarial environment, select π that maximally utilises (eff) the safety controller.
$\mathbf{R}^{\text{nuis}}_{\min=?}[\mathbf{C}^{<t}]$	Select π that minimises nuisance (nuis) up to time t.
Two-objective optimisation problems with reward-based constraints	
$\mathbf{R}^{\text{prod}}_{\max=?}[\mathbf{C}],\ \mathbf{R}^{\text{sev}}_{\leq b_s}[\mathbf{C}] \wedge \mathbf{R}^{\rho}_{\leq b_\rho}[\mathbf{C}]$	Select *sc* that maximises productivity, constrained by risk bound b_ρ and expected severity b_s.
$\mathbf{R}^{\text{prod}}_{\max=?}[\mathbf{C}],\ \mathbf{R}^{\text{sev}}_{\leq b_s}[\mathbf{C}]$	Select *sc* that maximises productivity, constrained by expected severity b_s of injuries.
Well-formedness constraints in ϕ_{wf}	
$\mathbf{E}\,\mathbf{F}\,goal$	$\mathcal{P}$ can finish the production cycle in a state labelled with *goal*.
$\mathbf{E}\,\mathbf{F}(f \wedge \neg final)$	Hazard f can occur during a production cycle.
$\mathbf{E}\,\mathbf{F}(deadlock \wedge \neg final)$	$\mathcal{P}$ can deadlock early, that is, without reaching *final*. (to be falsified)
$\mathbf{A}\,\mathbf{F}\,f$	Hazard f is inevitable. (to be falsified)
$\forall s \in S:\ \neg final \vee \neg init$	Some initial states are also final states. (to be falsified)
Correctness constraints in ϕ_{cor}	
$\mathbf{A}\mathbf{G}(\zeta \rightarrow \mathbf{A}\mathbf{F}^{<t}\,f)$	The controller detects χ for factor f within t steps.
$\mathbf{A}\mathbf{G}(\zeta \wedge \not\zeta \rightarrow \mathbf{A}(\zeta\,\mathbf{U}f))$	The controller timely responds to the observation of χ.‡
$\mathbf{A}\mathbf{G}(f \rightarrow \mathbf{P}_{>pr}[\mathbf{F}\bar{f}]) \wedge \mathbf{A}\mathbf{G}(\bar{f} \rightarrow \mathbf{P}_{>pr}[\mathbf{F}\not\bar{f}])$	The controller lively handles f, with phase transitions succeeding with a probability higher than *pr*.
$\mathbf{A}\mathbf{G}(f \rightarrow \mathbf{P}_{>pr}[\mathbf{F}\,goal])$	After hazard f, the controller resumes $\mathcal{P}$ so it can finish its cycle with a probability higher than *pr*.
$\mathbf{P}_{>pr}[\mathbf{G}\neg\underline{F}]$	Mishap freedom is more likely than *pr*.
$\mathbf{S}_{<pr}\,\underline{F}$	The steady-state probability of any $\underline{f}$ is below *pr*.

† *deadlock*: state with no commands enabled, *final*: end of manufacturing cycle (*goal*) or accident, *goal*: end of manufacturing cycle, *init*: initial state of a manufacturing cycle, $\underline{F}$: mishap state, *pr*: probability bound, prod: productivity, eff: controller effectiveness, sev: severity, ρ: risk. ‡We adopt the universality pattern of Dwyer et al. [46] using **U** instead of W because of the required response; note that we have to use the sensor predicate ζ rather than the ground truth predicate χ.

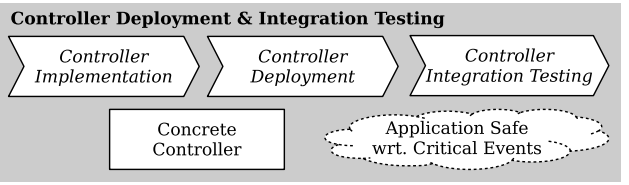


Fig. 13. Overview of the controller deployment and testing stage.

we constrain the search for solutions $\pi \in \Pi_{\mathcal{M}}$ to controllers whose mitigation paths from critical events are complete with at least probability *pr* (cf. Table 3). In the **MDP setting** (as explained in Section 5), our model allows the evaluation of freedom from accidents in $\mathcal{M}$ with

$$\mathbf{P}_{\neg\underline{F}} \equiv \otimes_{s\in\Xi}\, \mathbf{P}^{s}_{\min=?}[\neg\underline{F}\ \mathbf{W}\ \not\underline{F}] \tag{10}$$

where $\otimes \in \{\min, \text{mean}, \max\}$. For Ξ, Formula (10) requires the controller to minimise the probability of mishaps until the F-safe region (i.e., $S \setminus (\Xi \cup \underline{F})$) is reached. $\mathbf{P}_{\neg\underline{F}}$ aggregates min, the arithmetic mean μ, and max probabilities over Ξ. In the **pDTMC setting**, we use the plain quantification operator **P**. Table 3 contains further examples of properties in ϕ_{wf} and ϕ_{cor} to be verified or falsified of $\mathcal{M}$.

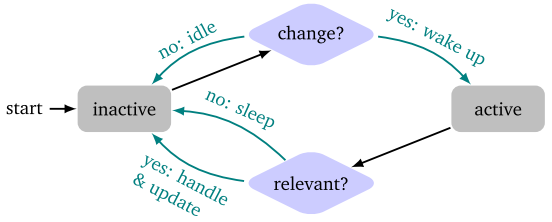
Verified synthesis. Based on the action rewards from the Sections 4.2.1, 4.2.1, and 4.2.2, the model checker investigates all choice resolutions and parameter valuations for $\mathcal{M}$ that fulfil well-formedness and further PRCTL constraints. The checker then identifies policies that fulfil the given constraints and are (Pareto-)optimal with respect to the optimisation objectives. The existence of an optimal strategy depends on the existence of a strategy in $\Pi_{\mathcal{M}}$ that fulfils the PRCTL constraints. Note that, by Definition 2, all policies considered for $\mathcal{M}$ are of the same size but may vary in their distribution of choice among the involved actors. The overall result of this step is a verified and optimal abstract controller extracted from the selected policy $\pi^\star$. An example of such a policy is visualised in Fig. 12.

6. Controller deployment and integration testing

Given the execution semantics of a target platform (e.g., the DTF), the selected controller can now be translated into a concrete executable form (Fig. 13) to be deployed, tested, demonstrated, and eventually used on this platform.

6.1. Controller implementation

The abstract controller, *sc*, is part of the calculated policy $\pi^\star$, a DTMC with state space $S_{sc} \subseteq S$. Following Section 3.4, S_{sc} is contained in the combination of the risk space $R(F)$, generated by YAP from the factor set F, with the process



(a) Execution scheme of the concrete safety controller in the Digital Twin Framework

```
1:  procedure SafetyController(in S|_Mon s_mon, out S|_Ctr s_ctr)
2:      (s, s_ρ, s'_ρ) ← Init(S|_Mon, R(F))   ▷ initialise controller and risk state
3:      while true do
4:          if (s, s_ρ) ≠ (s_mon, s'_ρ) then          ▷ wake up on event/change
5:              (s, s_ρ) ← (s_mon, s'_ρ)              ▷ capture monitored process state
6:              if s ∈ S_sc|_Mon then                 ▷ is the event relevant?
7:                  s_ctr ← HandleEvent(s, s_ρ)       ▷ issue control command
8:                  s'_ρ ← UpdateRiskState(s, s_ρ)    ▷ stay active if changed
9:              end if
10:         end if
11:     end while
12: end procedure
```

(b) Pseudo code of the implemented control algorithm

Fig. 14. Implementation of the safety controller, execution scheme (a) and control algorithm (b).

state space $S|_{X\setminus F}$. Coherent with Definition 2, $\pi^\star$ is a set $\delta_{\pi^\star} \subseteq \delta_{\mathcal{P}}$ of tuples $(s, \alpha, pr, s') \in S \times A_{\mathcal{P}} \times (0,1] \times S$ of the probabilistic transition function. However, the controller as a deterministic part of $\pi^\star$ then corresponds to a set of transitions $\delta_{sc}^{\rightarrow} \subseteq S_{sc} \times S_{sc}$ that, at the concrete level, again comprises guarded commands following Formula (7).

We use $\delta_{\mathcal{P}}$, obtained from a model checker (e.g., Prism) by expansion of the pGCL program $\mathcal{P}$ from the initial state s_0, to translate controller decisions in $\pi^\star$ into concrete actions. Using the four-relation structure of controller models in Parnas and Madey [47], this step involves (i) the translation of abstract states into guard conditions mapping concrete states into abstract states, and (ii) the translation of the abstract updates into low-level procedures generating control inputs to the process.

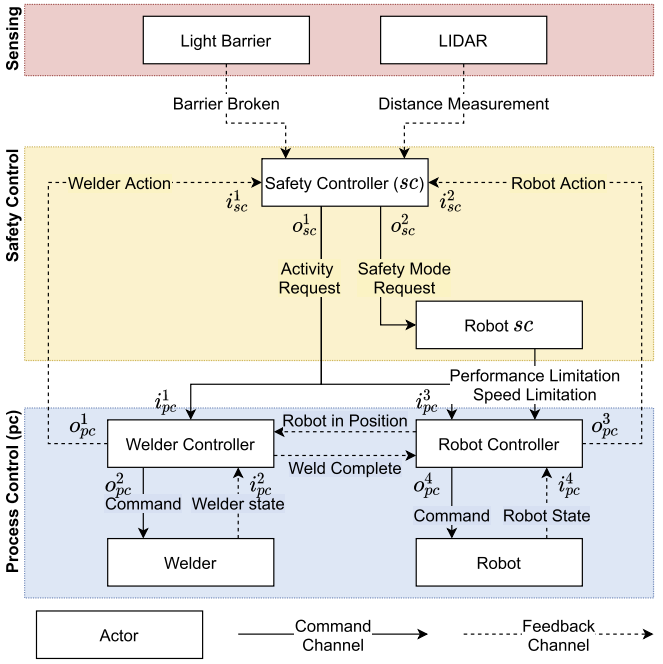
Fig. 14a shows the activation scheme of the concrete controller when deployed on an execution platform. When *inactive*, the controller continuously checks the process state and *wakes up* to be *active* if the state has *changed* from the last observation, and it *idles* otherwise. When *active*, the controller handles risk if the change was *relevant*, and it goes back to *sleep* otherwise.

Fig. 14b describes the algorithm of the discrete-event SafetyController. The controller receives updates in the monitorable state space $S_{sc}|_{Mon}$ being a projection of S_{sc} onto the monitored variables $Mon \subseteq X$ and can influence the controllable variables, resulting in a projection of S_{sc} onto the controllable variables $Ctr \subseteq X$ (Line 1). After initialisation of the monitored variables and the control (or risk) state (Line 2), the controller remains inactive (Line 3), unless a change in *Mon* is detected (Line 4). According to the scheme in Fig. 5, through Line 4, the controller is aware of each atomic update in *Mon*. Any update is recorded (Line 5) and checked for relevance (Line 6), that is, whether s covered by S_{sc} or whether the controller can stay idle. If relevant, the controller handles the event (Line 7) according to the commands generated by Line 1 and updates the control state (Line 8).

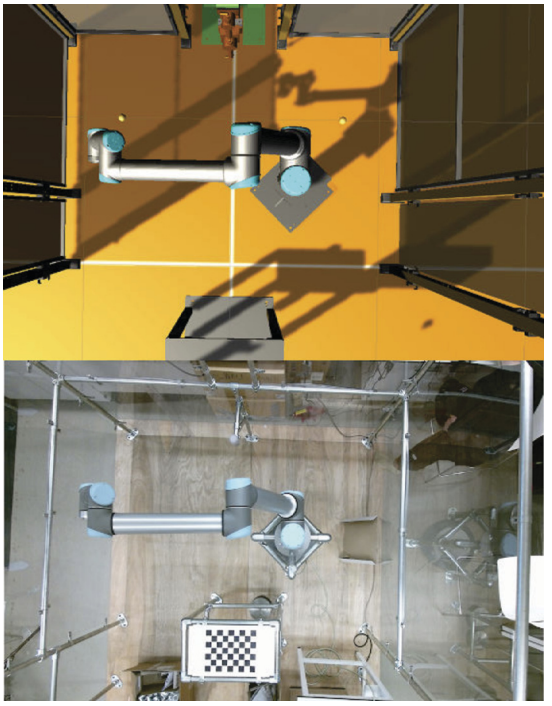
Example 8. In our case study, the check whether $s \in S_{sc}|_{Mon}$ in Line 6 of the Algorithm in Fig. 14b is implemented as a switch statement iterating over all relevant critical events derived from $\mathcal{M}$. The function HandleEvent is responsible for issuing control inputs, such as switching into a power & force limitation mode and notifying the operator to leave the work cell if HC is activated. The function UpdateRiskState is responsible for managing and remembering the risk state internal to the controller (e.g., the phase in the mitigation of HC). In the supplemental material[a] for this work, we provide modelling and code examples[b] and a video[c] of the controller in action.

[a] See https://github.com/douthwja01/CSI-artifacts/tree/master/controller-synthesis.
[b] See the `hrc2` example of the Yap package (https://github.com/ytzemih/yap).
[c] See https://youtu.be/cm-XkZ_aitQ.

Expected overhead. The detection and handling overhead is the time elapsed in every cycle of the **while** loop in the Algorithm shown in Fig. 14b. Let $d\colon A_{\mathcal{P}} \to \mathbb{Q}_{\geq 0}$ be the processing time required for an action, for example, the calculation of the detection of HC in e^{HC}. If implemented as part of a sequential cell controller, Line 6 requires a time slot of length $\Sigma_{\mathsf{f}\in F} d(e^{\mathsf{f}})$ in each control cycle. If Line 6 is monitored simultaneously in dedicated safety controller hardware, the slowest detection rate for F is $1/\max_{\mathsf{f}\in F} d(e^{\mathsf{f}})$. The overhead for handling f in Line 7 can be estimated from Fig. 6b and may range from $d^{\mathsf{f}}_{\min} = d(m^{\mathsf{f}}_{sm}) + d(m^{\mathsf{f}}_{\boldsymbol{act}}) + d(m^{\mathsf{f}}) + d(r^{\mathsf{f}}_{sm}) + d(r^{\mathsf{f}}_{\boldsymbol{act}})$ to $d^{\mathsf{f}}_{\max} = d^{\mathsf{f}}_{\min} + x \cdot d(m^{\mathsf{f}}_{sf}) + d(r^{\mathsf{f}}_{sf})$ with a repetition factor $x\colon \mathbb{N}$. The overhead of the implementation in the Algorithm shown in Fig. 14b can be obtained by recording timed state sequences from an execution platform (e.g., the DTF). In order for the controller to interact with such a platform, the Yap model is extended by an interface specification in addition to the model fragments discussed in the previous sections. This interface is discussed in more detail in the following section.



(a) Communication structure in the DTF. In accordance with our case study, a safety mode is requested by the safety controller (middle block) to apply to the welder and robot digital twins (bottom block). In response, the nominal process for both systems is overridden. For example, in response to a detected hazard the controller may issue a "safety stop", preventing the process continuing.



(b) The DTF applied to our case study. Requests issued to the robot within the DTF are enacted by the physical twin in the real-world system. The robot, LIDAR, and light barrier provide feedback to the environment.

Fig. 15. Communication structure in the DTF (a) and correspondence of digital and real twins (b).

6.2. Controller deployment

Section 3.6 introduces the notion of actors within a collaborative manufacturing setting involving a cobot and an operator (see Fig. 15b). We are able to reconstruct this setting in the form of a digital twin using our Digital Twin Framework. The DTF is a toolchain developed in C# and visualised in Unity3D that provides the kinematic, communication, and data infrastructure necessary to deploy digital twins on real world systems.[15] Using the DTF APIs for the MATLAB® robotics toolbox and the Robot Operating System framework,[16] the safety controller's actions are exchanged with the physical platform in real-time where a response is demonstrated.

In the DTF, we represent a scenario for the cobot setting as an aggregation of actions and decisions made by each actor. Actors may then be identified as (i) *digital twins*—actors with a distinct collection of models, state-machines and behaviours that emulate the capabilities of the physical system—and (ii) *abstract* actors without physical embodiment that may provide a service, communicate with or control other actors. The term *environment* will be used to describe this complete set of actors. Consistent with the Sections 3 and 4, in this environment, we separate the concern of safety from the nominal process (Fig. 15a). Thus, there is a *process controller* responsible for the application and a safety controller responsible for safety, able to interact with the process controller.

Process representation. Let $\mathcal{A}$ be a set of actors. Examining the process from a network perspective allows us to model a communication node as an actor $n \in \mathcal{A}$, similar to the Robot Operating System framework. Following the terminology by Broy [48], the actor n is able to communicate with other nodes through an interface $\mathrm{IF}(n) = (I_n, O_n)$. Here, $I_n = [i_n^1, i_n^2, ..., i_n^j]$ and $O_n = [o_n^1, o_n^2, ..., o_n^k]$ denote n's input (subscription) and output (publication) ports. This decentralised structure allows us to represent a process controller $pc \in \mathcal{A}$ as an abstract actor with the interface $\mathrm{IF}(pc) = (I_{pc}, O_{pc})$. pc is modelled as a state machine that listens to outputs on O_n and issues commands to some actors via some ports in O_{pc}. The interfaces and their ports are then connected via command and feedback channels.

Each sensor present in the process is similarly introduced as a digital twin actor $dt \in \mathcal{A}$. Here, data originating from the sensing capabilities of dt are broadcast to assigned ports O_{dt} in order to inform the network n of changes to the

[15] See https://github.com/douthwja01/CSI-artifacts/JSS/.
[16] See https://www.ros.org.

physical environment. Manipulators and machinery are modelled as digital twins of the physical equipment, with behaviours informed by the actuation constraints of the physical system. In response to commands issued by *pc*, the robot digital twin is able to interact with the workpiece, operator or tendered machine. This communication is then forwarded and expressed by the physical twin as seen in Fig. 15b.

Safety controller. A safety controller $sc \in \mathcal{A}$ is introduced to the DTF as a singleton node declared as an abstract actor. *sc* has the interface $IF(sc) = (I_{sc}, O_{sc})$ where $I_{sc}, O_{sc} \subseteq X$ and communicates on $O_{sc} = [o^1_{sc}, o^2_{sc}, ..., o^l_{sc}]$ with a family of process actors $\{p_i\}_{i\in 1..k} \subset \mathcal{A}$. The nominal procedure of *pc* is governed by a local hierarchical state machine responding to process updates received over I_{pc}. To allow the *sc* to intervene in *pc*'s procedure, $I_{sc} = [i^1_{sc}, i^2_{sc}, ..., i^l_{sc}]$ is implemented as a state machine (e.g., "Robot *sc*" in Fig. 15a). This state machine allows *sc* to enact changes to *pc*'s safety mode(s) in response to requests made over ports in O_{sc}.

Example 9. As part of the case study (Section 2), we developed an environment (Fig. 15b top half) to evaluate the synthesised safety controller. This environment includes a distributed system composed of multiple digital twins, with each physical actor (e.g., robot, spot welder, operator) and component (e.g., sensors) in the real-world process assigned its own individual twin. The process under consideration is a work-piece exchange and welding task, overseen by a process controller that issues and responds to tasks assigned to each actor. The safety controller observes feedback from the process controller, a workbench light barrier and a LIDAR positioned within the cell. Fig. 15a shows the communication of safety mode and activity change requests to all actors within the environment on occurrence of a critical event, following the event handling scheme in Fig. 14a.

6.3. Controller verification by testing

For testing the implementation of *sc* in the DTF (Section 3.5), we translate requirements ϕ given as PRCTL properties (Table 3) into corresponding MITL properties ϕ_{MITL}. Particularly,

$$\mathbf{G}(\zeta \wedge \not f \rightarrow \zeta\, \mathbf{U}_{[0,t]} f) \tag{11}$$

requires *sc* to detect the critical event, that is, to get active whenever the sensor predicate ζ holds true within t time units. Furthermore,

$$\big((\mathbf{F}\, goal) \rightarrow \big(\mathbf{G}(f \rightarrow \mathbf{F}\bar{f}) \wedge \mathbf{G}(\bar{f} \rightarrow \mathbf{F}\not f)\big)\big) \wedge \mathbf{G}\neg \underline{f} \tag{12}$$

states for a completed process ($\mathbf{F}\, goal$) that whenever the controller detects f (Table 1) it moves the DTF to a state where f is mitigated ($f \rightarrow \mathbf{F}\bar{f}$) and from there, given the environment (e.g., operator) eventually reacts, it returns the process to a state where f is inactive and operation is resumed as far as possible ($\bar{f} \rightarrow \mathbf{F}\not f$). Even if the environment does not react, an accident never occurs ($\mathbf{G}\neg \underline{f}$).

We derive U (e.g., Fig. 18g) by using the model checker to obtain a witness for $\mathbf{F}\, goal$ and extract the corresponding sequence of inputs to the DTF from that witness. To define U as a witness for $\mathbf{F}\, goal$ implies that U represents a shortest successful (accident-free) path in $Paths(\mathcal{M})$. As such U models closely what would happen in the real work cell. In the DTF, we use a combination of (iii) and (iv) from Section 3.5 to derive a set of *timed paths in vicinity* $\mathcal{V}_{\mathcal{M}}$ *of* U as $\mathcal{T}_{\mathcal{V}}(U) = \{\omega \in TPaths(MCobot) \mid \mathcal{V}_{MCobot}(\omega, U)\} \subseteq TPaths(MCobot)$ as follows. To indirectly define $\mathcal{V}_{\mathcal{M}}$, we extract from U two independent sequential processes (e.g., operator *op*, robot/welder *rw*), parameterise one (e.g., *op*), and perform its subsequent events (i.e., transitions) in *MCobot* at different times, leading to different interleavings (iii) and different timed paths (iv). We record observations of *MCobot* with time stamps, resulting in the set $\mathcal{T}_{\mathcal{V}}(U)$ of stimulated and recorded finite timed paths of *MCobot*. As a result, the vicinity $\mathcal{V}$ is a set of (randomly generated) configurations of the DTF (e.g., considering the operator being far, near, and close to the spot welder, and entering the cell at different times), more specifically, a subset of relevant interleavings of the behaviours of the chosen processes (e.g., *op*, *rw*) obtained from changing certain waiting times of one process (e.g., *op*). We statistically explore $\mathcal{T}_{\mathcal{V}}(U)$ by playing in a sample to the DTF, record the path ω, and verify $\omega \models_{\mathsf{MITL}} \phi_{\mathsf{MITL}}$ using the MITL path checker PyMTL [49]. The *MU* is then derived by interleaving an unusual process (e.g., a second operator) with U. In the context of practical certification, one would need to perform this step for all or at least a representative subset of witnesses for $\mathbf{F}\, goal$.

Data extraction. To allow communications between actors to be recorded and analysed, an additional abstract actor is introduced as a network *snooper*. This actor subscribes to updates from a family of actors $\{p_i\}_{i\in 1..n} \subset \mathcal{A}$ and exposes their communications externally as a series of data sequences. As the scenario is executed, the generated sequences are marked with a reference time and origin of each data packet. Communications between the safety controller, process controller and other actors are then recovered for testing, debugging, and evaluation of the safety controller.

The safety controller can be exported as a C# extension to our DTF and deployed within a digital twin of the case study in Section 2. At the point of analysis, each data sequence is composed of data packets with an assigned timestamp and entity identifier. Each sequence is exported from the database following the scenario execution. This allows the data to be presented as a ledger of all monitored ports, indicating where inter-actor communication occurred, actor states have

changed, and the controller guards have been triggered. From each such data sequence, we can then extract a timed path for testing.

7. Evaluation

In this section, we pose three research questions about safety, utility, and scalability, describe the evaluation methodology for each of these questions, and discuss the results of our evaluation.

7.1. Research questions and methodology

Based on the questions raised in Section 1, we investigate key aspects of our approach by asking three research questions (RQs).

RQ1 (Safety) What is the likelihood of accident-free operation under the control of a synthesised safety controller?
RQ2 (Utility) Does the safety controller reduce the number of hard stops of the robot due to hazards, compared to a basic controller that switches off the system whenever the operator intrudes the work cell?
RQ3 (Scalability) How well can the proposed approach deal with multiple hazards and mitigation and resumption options?

Methodology for RQ1 (safety). We answer **RQ1** in two stages, first based on our modelling approach (**RQ1a**) and, second, supported by our deployment and testing approach (**RQ1b**).

Methodology for RQ1a. We evaluate freedom from accidents according to Formula (10), leading to probability triples comprising min, the arithmetic mean μ, and max.

In the **MDP setting**, we use PRISM to select $\pi^\star \in \Pi_{\mathcal{M}}$ according to the three two-objective optimisation problems

$$\mathbf{R}^{\text{pot}}_{\max=?}[\mathbf{C}] \text{ and } \mathbf{P}_{\max=?}[\mathbf{F}\, goal], \tag{a}$$
$$\mathbf{R}^{\text{prod}}_{\max=?}[\mathbf{C}] \text{ and } \mathbf{P}_{\max=?}[\mathbf{F}\, goal], \text{ and} \tag{b}$$
$$\mathbf{R}^{\text{eff}}_{\max=?}[\mathbf{C}] \text{ and } \mathbf{R}^{\text{nuis}}_{\max=?}[\mathbf{C}]\,. \tag{c}$$

In the spirit of negative testing, Formula (a) aims at maximising the use (i.e., the risk-reduction *pot*ential) of the safety controller while maximising the probability of reaching the *goal* region (i.e., finishing two tasks, a workpiece and carrying through cell maintenance). This problem approximates worst-case behaviour of the operator and other actors but does not take into account further optimisation parameters defined for mitigations and resumptions. As opposed to that, Formula (b) fosters the maximisation of *prod*uctivity, any combination of decisions allowing the finalisation of tasks is preferred, hence, transitions leading to accidents or the use of the controller are equally neglected. While Formula (c) also forces the environment to trigger the controller, these policies represent the best controller usage in terms of *nuis*ance and *eff*ort. Because of restrictions in the use of $\mathbf{R}_{\min}$ for MDPs, we maximise costs interpreting positive values as negative (e.g., the higher the nuisance the better). We then investigate the Pareto front of optimal policies synthesised from the Formulas (a), (b), and (c). For policies with less than 1000 states, we inspect the corresponding policy graphs (e.g., whether there is a path from *initial* to *goal* or whether paths from unsafe states reachable from *initial* avoid deadlocks). Finally, we evaluate accident freedom according to Formula (10), except that we use $\mathbf{P}_{=?}$ for DTMCs instead of $\mathbf{P}_{\min=?}$.[17]

In the **pDTMC setting**, we use EVOCHECKER to identify a Pareto set of optimal policies $\Pi^\star_{\mathcal{M}} \subseteq \Pi_{\mathcal{M}}$ according to the constrained three-objective optimisation problem:

$$\text{maximise } \mathbf{R}^{\text{prod}}_{=?}[\mathbf{C}^{\leq t}] / \left(\mathbf{R}^{\text{disr}}_{=?}[\mathbf{C}^{\leq t}] + \mathbf{R}^{\text{eff}}_{=?}[\mathbf{C}^{\leq t}]\right) \text{ and} \qquad \text{(Productivity)}$$
$$\text{minimise } \mathbf{R}^{\text{nuis}}_{=?}[\mathbf{C}^{\leq t}] \text{ and} \qquad \text{(Nuisance)}$$
$$\text{minimise } \Sigma_{\mathsf{f}\in F}\left(s_{\mathsf{f}} \cdot \mathbf{R}^{\rho,\mathsf{f}}_{=?}[\mathbf{C}^{\leq t}]\right) \text{ such that} \qquad \text{(Risk)}$$
$$\forall \pi^\star \in \Pi^\star_{\mathcal{M}} : \pi^\star \models \neg\, \mathbf{E}[\mathbf{F}(deadlock \wedge \neg final)] \tag{13}$$

over a time period $[0, t]$ and with factor-specific scaling values s_{f}. This problem contains a probabilistic constraint ruling out early deadlocks (Formula (13)), and three objectives for maximising relative productivity (i.e., the ratio of productivity over *disruption* of the operator and effort spent), minimising nuisance (now interpreted[18] as: the higher the nuisance the worse), and minimising risk from a factor set F using the multi-reward structure from Section 4.2.1. We assess the resulting Pareto front, choose a solution $\pi^\star \in \Pi^\star_{\mathcal{M}}$, and use YAP to refine $\pi^\star$ into a concrete controller.

Methodology for RQ1b. In Section 5.2, we described the synthesis of a correct abstract controller *sc* and, in Section 6.1, its translation into a concrete controller interfacing with the DTF explained in Section 6.2. Then, how do we assure the transfer

[17] To keep manual workload under control, if the model checker (here, PRISM) lists several adversaries, we apply the experiment only to the first listed.
[18] For this interpretation, we had to reassign the corresponding action rewards in a revised model.

of the results on freedom from accidents of the abstract controller (**RQ1a**) to the concrete one in the DTF? For this, we follow the framework in Section 6.3 and deploy and test *sc* in the DTF for compliance with $\mathcal{M}$ in a use case and assess its behaviour in a misuse case.

The DTF is used in a simulation capacity to test the integration of the proposed safety controller into its operational environment. The simulation provides for (i) automated testing and (ii) a safe environment for evaluation, whilst representing a faithful one-to-one reconstruction of the work cell. Physical, digital, or mixed environments expose the same interface for integration with the DTF; the same controller can be used in either context without modification. As such, integration with the physical work cell is not evaluated, in part due to limited access to the lab replica.[19]

We construct $\mathcal{T}_{\mathcal{V}}(U)$ (Section 3.5) for the use case (U) as depicted in Fig. 18g and we derive one misuse case (MU), both to activate all the factor phases of HC, HS, and HRW described in Table 1. The scenarios are:

U: During operation, the operator reaches across the workbench and walks to the spot welder.
MU: During operation, the operator reaches across the workbench while another operator walks to the spot welder.

Let $F = \{\mathsf{HC}, \mathsf{HRW}, \mathsf{HS}, \mathsf{HW}\}$ for the remainder this and the next section. We perform tests for the use case as outlined in Fig. 18g. The operator first heads to the workbench, breaking the light barrier (HRW), before heading inside the cell (HS) close to the spot welder (HC). The observed path during each test can be split for the independent test of each risk factor. Considering the path as a whole provides for the test of hazard mitigation and the absence of impact on further actions.

The operator waits at given positions while the cobot and spot welder proceed through their scheduled activities. Variations in the time spent by the operator in different states result in different interleavings of the operator and robot, and such variations in turn might be ground for the activation of hazards in the system. The configuration for each test is a vector of 4 values corresponding to 4 wait operations of the operator: entering the workbench, at the workbench, entering the work cell, and at the spot welder. To bound the time taken by each test, the values are picked such that the operator completes its actions in less than 20 s. This is ample time for the robot to perform its own actions, and the operator is able to disrupt the different tasks in the process either by walking to the spot welder or reaching across the workbench. We rely on the Dirichlet-Rescale algorithm [50] for generating vectors such that the values of the vector sum to a given total, and the distribution of vectors in the constrained space is uniform.

We adopt situation-based coverage criteria, proposed by Alexander et al. [51], to assess the performance of our test campaign, and whether we have achieved a satisfactory level of testing. A situation is a state or transition having been observed at a particular time. With the set $Sit^{\rho} = Ph_F \setminus \{\underline{HC}, \underline{HRW}, \underline{HS}, \underline{HW}\}$, we ensure that the non-accident region of $R(F)$ (all phases of $\mathsf{f} \in F$ except for the mishap $\underline{f}$, as defined in Fig. 6b) has been encountered. For Sit^{ρ}, we thus obtain the correspondences $[\![Sit^{\rho}]\!]_{R(F)} = R(F) \setminus [\![\underline{F}]\!]_{R(F)}$ and $[\![Sit^{\rho}]\!]_S = S \setminus [\![\underline{F}]\!]_S$ for $R(F)$ and S respectively.

We define further situations in terms of the state of the spot welder (captured by variable $wact \in X$), the robot activity ($ract \in X$), and the robot location ($rloc \in X$), against the interferences caused by the operator ($hloc \in X$) in the course of U, either by reaching across the workbench ($([hloc \mapsto _, \dots], [hloc \mapsto sharedTbl, \dots]) \in \delta_{\mathcal{P}}^{\rightarrow}$) or entering the cell ($([hloc \mapsto _, \dots], [hloc \mapsto inCell, \dots]) \in \delta_{\mathcal{P}}^{\rightarrow}$). Recall, the situation (HRW) where the operator reaches for the workbench while the robot is at the workbench is described by the formula $hloc = sharedTbl \wedge rloc = sharedTbl$.

Let $Y \subseteq X$ and $x \in X$ and

$$[\![Sit^{Y}]\!]_S \text{ be a minimal element in } \left\{S' \subseteq S \mid \forall x \in Y, v \in \mathsf{type}\, x\, \exists s \in S' \colon s(x) = v\right\} \text{ and}$$
$$[\![Sit^{\rightarrow x}]\!]_{\delta_{\mathcal{P}}} \text{ be a minimal element in } \{\delta_{\mathcal{P}}^{\rightarrow'} \subseteq \delta_{\mathcal{P}}^{\rightarrow} \mid \forall v \in \mathsf{type}\, x\, \exists (s, s') \in \delta_{\mathcal{P}}^{\rightarrow'} \colon s'(x) = v\}\,.$$

Sit^{Y} denotes a smallest set of states covering $\bigcup_{y \in Y} \mathsf{type}\, y$, that is all possible values of all variables in Y. Likewise, $Sit^{\rightarrow x}$ is a smallest set of transitions reaching all possible values of $\mathsf{type}\, x$. Then, we construct the set of situations

$$[\![Sit^{Act}]\!]_{S,\delta_{\mathcal{P}}} = [\![Sit^{\{wact,ract,rloc\}}]\!]_S \times [\![Sit^{\rightarrow hloc}]\!]_{\delta_{\mathcal{P}}}$$

resulting from interferences from the operator at each step in U. The set of situations *to cover* is thus defined as,

$$[\![Sit]\!]_{S,\delta_{\mathcal{P}}} = [\![Sit^{Act}]\!]_{S,\delta_{\mathcal{P}}} \cup [\![Sit^{\rho}]\!]_S\,.$$

We process the path $\omega \in TPaths(DT)$ produced during each run to identify the encountered situations in Sit. Full coverage (i.e., the construction of $\mathcal{T}_{\mathcal{V}}(U)$) is achieved when all such situations have been observed, that is, the interference caused by the operator has been observed against all actions and positions of the robot and spot welder. This ensures that causal factors have been encountered, mitigated, and the system could resume normal operation.

Methodology for RQ2 (utility). We argue that the synthesised safety controller is better than a state-of-the-art controller that only has a stop mode and performs no automatic resumption. For this argument, we compare the range of controllers from

[19] At the time of conducting our experiments, COVID-19 prevents access to the Sheffield Robotics lab and physical components of the case study. For this reason, we do not use the network n, digital twin actors dt, and their interfaces $\mathrm{IF}(dt)$ in our example.

Table 4
Results of the experiment for **RQ1a** (accident-free operation) and **RQ3** (scalability) in the **MDP setting**.

Risk Model[†]				**MDP**[†]			**(a) max-ASC**[†]			**(b) max-prod**			**(c) opt-ASC**		
F	mr/c	$\lvert R(F)\rvert$	d_Y [ms]	$\mathbf{P}_{\neg\underline{F}}$ $\lfloor\mu\rceil$	$\lvert\Xi\rvert$	$\lvert S\rvert/\lvert\delta_{\mathcal{P}}^{\rightarrow}\rvert$	$\mathbf{P}_{\neg\underline{F}}$ $\lfloor\mu\rceil$	$\lvert\Xi\rvert$	d_P [s]	$\mathbf{P}_{\neg\underline{F}}$ $\lfloor\mu\rceil$	$\lvert\Xi\rvert$	d_P [s]	$\mathbf{P}_{\neg\underline{F}}$ $\lfloor\mu\rceil$	$\lvert\Xi\rvert$	d_P [s]
HC	5/0	3	40	[.9,.9,.9]	14	322/1031	[1,1,1]	3	.02	[1,1,1]	1	.02	[1,1,1]	6	.15
+HS	9/2	5	52	[.92,.96,.98]	256	930/3483	[.07,.66,1]	11	.77	[0,.88,1]	8	.82	[.95,.98,1]	18	.9
+WS	11/3	8	44	[.93,.97,1]	288	1088/3865	[0,.29,1]	17	2.1	[0,.8,1]	5	2	[1,1,1]	24	1.5
+HRW	13/7	16	65	[.93,.97,1]	981	7675/33322	[1,1,1]	17	9.7	[1,1,1]	11	9.4	[1,1,1]	15	13.3
+HW	15/8	36	76	[.93,.97,1]	2296	21281/98694	[1,1,1]	15	42.9	[0,.71,1]	7	41.4	[1,1,1]	15	46.6
+RT	15/9	50	87	[.93,.97,1]	2864	21965/100133	[1,1,1]	13	48.2	[1,1,1]	9	46.4	[1,1,1]	15	53.8
+RC	15/15	122	162	[.93,.99,1]	12079	21670/102263	[0,.94,1]	35	38	[0,.72,1]	22	36.6	[1,1,1]	36	51.1

[†] F…factor set; mr/c…no. of mitigations+resumptions/constraints; $\lvert R(F)\rvert$…cardinality of the relevant subset of $R(F)$ (Section 3.4); d_Y…YAP's processing time; $\mathbf{P}_{\neg\underline{F}}$…probability of conditional freedom from accidents; Ξ…set of F-unsafe states; $\lvert S\rvert/\lvert\delta_{\mathcal{P}}^{\rightarrow}\rvert$…number of states/transitions of the MDP ($\lvert\delta_{\mathcal{P}}^{\rightarrow}\rvert$ equals the size of the policies in $\Pi_{\mathcal{M}}$); Formulas (a), (b), and (c)…optimisation problems; d_P…PRISM's processing time

Table 5
Comparison of minimum, average (μ), and maximum freedom from accidents in $\mathcal{P}$ and $\mathcal{P} \setminus sc$ according to Formula (10).

Process …	min	μ	max
…with a safety controller ($\mathcal{P}$)	63%	97%	100%
…without a safety controller ($\mathcal{P} \setminus sc$)	63%	87%	100%

the design space with those controllers that always perform a safety stop upon detecting a critical event. Based on that, we assess the increase in productivity and fluency of collaboration, and the decrease of mean-time to finishing a process cycle. This argument underpins our contribution to the problem statement in Section 1.

Methodology for RQ3 (scalability). We prepare and analyse multiple increments of the risk model, each adding one critical event, mitigation and resumption options, and constraints to the risk model. We record the resulting model sizes and analysis times.

7.2. Results

In the following, we present the results of our evaluation separately for each research question.

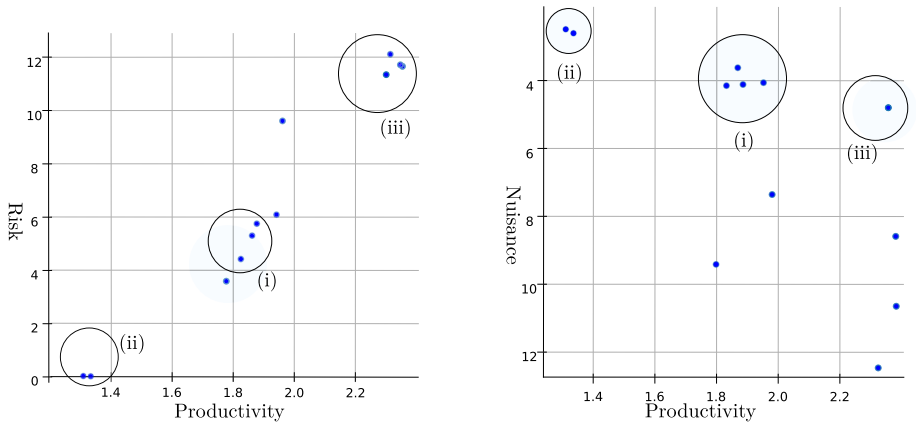
7.2.1. Results for RQ1a: safety in the model

We consider as inputs a risk model and a process model of the work cell, with a single initial state of these models where all actors are in the activity off and no critical event has occurred. The risk model is given in YAP's input language and the behavioural model is given in PRISM's flavour of pGCL, instrumented with YAP template placeholders. We consider the two settings explained in Section 5. In the **MDP setting**, we use YAP 0.5.1 for generating the design space (Section 5.1) embedded into a process model without alternating execution semantics and with controller synthesis directly from an MDP using PRISM 4.5. In the improved **pDTMC setting**, we use YAP 0.7.1 for generating the design space embedded into $\mathcal{M}[\mathcal{P}]$ with the alternating execution semantics (Section 4.1.1), an improved accident model (Section 4.3), and with controller synthesis from a pDTMC using EVOCHECKER and PRISM 4.5. For **RQ1a**, we used GNU/Linux 5.4.19 and 5.8.0 (x86, 64 bit), and an Intel® Core i7-8665U CPU with up to 8 Threads of up to 4.8 GHz, and 16 GiB RAM.

The results for the original **MDP setting** are displayed in Table 4, which shows the data collected based on seven increasingly more complex risk models. For example, row 3 (highlighted) uses a risk model with $F = \{\mathsf{HC}, \mathsf{HS}, \mathsf{WS}\}$. This model contains 11 mitigation and resumption actions in $A_{\mathcal{P}}$ and $\lvert R(F)\rvert = 8$ (instead of 27) because of being subjected to 3 constraints (WS requiresNOf (1 | HC, HS), HC requires HS, HC mitDeniesMit HS). It took YAP $d_Y = 44$ ms to process the model. In the MDP, the (min, mean, max) probability triples are denoted by $\lfloor\mu\rceil$. The result $\lfloor\mu\rceil = [.93, .97, 1]$ for a policy denotes 93-100% conditional freedom from accidents ($\mathbf{P}_{\neg\underline{F}}$) based on $\lvert\Xi\rvert = 288$ hazardous states among the $\lvert S\rvert = 1088$ states. The solution of Problem (a) results in a poor $\mathbf{P}_{\neg\underline{F}}$ with a 71% (100%-29%) accident probability on average among the 1088 states, even though the policy only crosses $\lvert\Xi\rvert = 17$ unsafe states. The reason for this poor performance of that model is that it introduces the mishap WS and does not contain a handler for HRW, such that corresponding accidents remain easily reachable. It took PRISM $d_P = 2.1$ s to process the model. In contrast, the solution of Problems (b) and (c) leads to controllers with a much improved $\mathbf{P}_{\neg\underline{F}}$.

Overall, the desirable $\lfloor\mu\rceil = [1, 1, 1]$ (denoting 100%) is most often achieved with Formula (c) due to the fact that simultaneity of decisions of the environment and the safety controller in the same state is avoided by focusing on rewards only specified for controller actions. Such rewards model the fact that a controller is usually much faster than an operator. Formulas (a) and (b) show poorer freedom from accidents because *prod*uctivity rewards given to the environment compete with rewards given to the safety controller to exploit its risk reduction *pot*ential.

In the improved **pDTMC setting**, we focus on a single risk model with the three factors HS, HC, and HRW. We calculate $\lfloor\mu\rceil$ for Formula (10) for a process without a safety controller ($\mathcal{P} \setminus sc$) and one with a controller ($\mathcal{P}$). Table 5 shows an



(a) Projection: risk and productivity

(b) Projection: nuisance and productivity

Fig. 16. Two projections of a 3D Pareto front for the optimisation objective in Formula (13) for the time period up to $t = 30$.

increase in average freedom from accidents from 87% to around 97% when using a safety controller. Starting from any state in the process shows that there are very safe states with a probability of accidents of down to 0% (max column) and rather dangerous states with a probability of up to 37% (min column). Overall, going from 13% down 3% average accident probability (across the three risk factors) means that the controller in this particular process leads to a reduction of accidents by 77%.

The 3D Pareto front in Fig. 16 indicates that the lower the risk of a controller (Conjunct (Risk) in Formula (13)) the lower the productivity (Conjunct (Productivity)) due to the interventions of the controller (cf. Fig. 16a). The Figs. 16a and 16b show (i) four controllers in the design space with low nuisance (Conjunct (Nuisance)) and medium productivity and risk, and (ii) two minimal-risk controllers with low nuisance but low productivity. (iii) Controllers that minimise nuisance and maximise productivity tend to do this at the cost of high risk. If the minimal-risk controllers under (ii) are not satisfactory, a more detailed trade-off regarding the controllers in (i) or a repetition of the analysis with different design parameters or an altered process and controller model will have to be made.

7.2.2. Results for RQ1b: safety in the digital twin

We performed 100 tests for the use case shown in Fig. 18g. This proved sufficient to achieve full coverage of (i) the mitigation phases of all considered risk factors and (ii) the operator interference types across spot welder and cobot states. All recorded timed state sequences ω verify $\omega \models_{\mathsf{MITL}} \phi_{\mathsf{MITL}}$, that is, all paths satisfy the selected properties translated from Table 3. Risk factors were correctly detected (with $t = 0.25$ ms in Property (11)) and mitigated by the synthesised safety controller in all observed situations in the DTF. Following mitigation, the process was always observed as returning to a nominal state, where it completes as expected and all risk factors are inactive (Property (12)). All results for **RQ1b** were produced under the Windows 10 Home Edition operating system, build 19042.985, on an Intel® Core i5-8250U 4-core CPU at 1.8 GHz, with 8 GiB RAM. The whole test suite, including path checking with PyMTL, took less than two hours to complete. No time compression was used, the DTF ran simulations in real-time.

Fig. 18 illustrates the output of the DTF for the use case (Fig. 18g). Each image captures the state of the system as the operator begins a wait operation. The use case highlights the activation of all monitored risk factors (see Table 1) with the operator and cobot making concurrent use of the workbench, and the operator in the vicinity of the spot welder as an operation is about to start.

Following the approach in Section 6.3, our instance of *MU* relies on two operators simultaneously following the behaviours outlined by U. The tested model assumes a single operator interacts with the system, such that an operator must leave the workbench before entering the work cell. As such, the safety controller correctly mitigates the risk due to both cobot and operator reaching for the workbench. However, as the second operator enters the cell the related risk factors are neither identified as active nor mitigated. Fig. 17 highlights the situation, with the second operator in close proximity to an active spot welder.

The work cell is safe considering a single operator following the use case, that is entering the work cell or the workbench during operation. However, the risk and process models would need to be revised to account for multiple operators interacting with the system (e.g., by removing the factor dependencies between HRW and HS, see Section 4.2.1). The case with two operators would normally be picked up in an extended risk assessment following Table 1, but it was not relevant to this particular process. This case is out of scope to keep the case study simple.

7.2.3. Results for RQ2: utility

In the **MDP approach**, Table 6 highlights several aspects. First, the fact that none of the controller configurations achieves the productivity level (64.19) of the process without a controller. Such a process reaches a theoretical productivity maximum

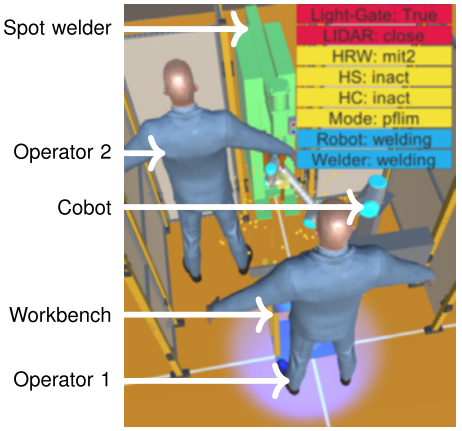
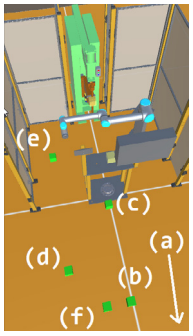
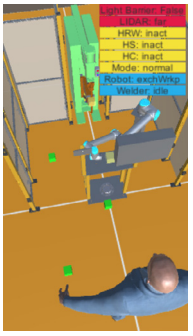


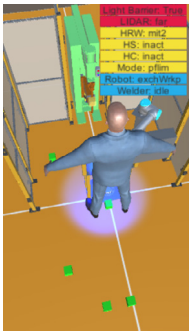
Fig. 17. Result for the misuse case (*MU*): operator 2 is not notified to leave the cell, and the spot welder remains active while operator 2 is in close proximity.



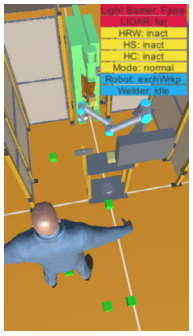
(a) System configuration for testing, with the operator's path highlighted by numbered waypoints.



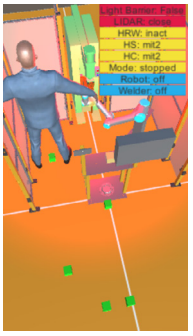
(b) Initial state. The operator waits in front the workbench while the cobot moves to retrieve a workpiece.



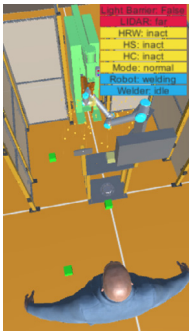
(c) The *sc* issues mitigations as the operator and cobot simultaneously reach for the workbench (*HRW*).



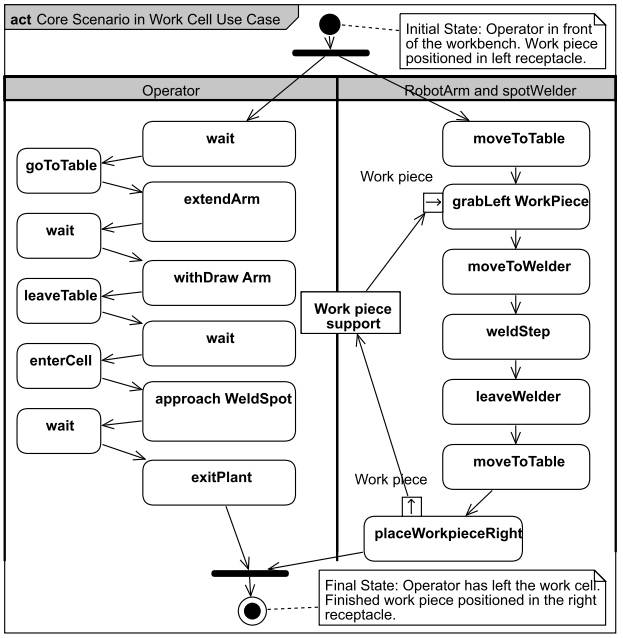
(d) The operator leaves the workbench while the cobot moves to the spot welder. The risk factor has been removed.



(e) The controller issues a stop in response to the operator approaching the spot welder during operation (*HS* and *HC*).



(f) Welding resumes after the operator leaves the work cell. The notification to leave the cell is deactivated.



(g) Use case describing operator, cobot and spot welder (inter)actions. In the top right corner of each image on the left, the state of the different actors of the system is highlighted as the use case progresses, that is, the sensors (light barrier and LIDAR), the safety controller (*HRW*, *HS*, *HC*, and the safety *mode*), and the activities (of the cobot and the spot welder). A visual notification instructing the operator to leave the handover table or welding area are issued in the form of blue and red lights respectively.

Fig. 18. Illustration of different states (18a)-(18f) of the use case described in (18g).

Table 6
Comparison of a process without a safety controller with processes including controllers using different safety mode and activity switching policies. Safety functions such as warnings are not taken into account in this comparison.

Process	Configuration†			Productivity‡	Risk‡
	HC	HRW	HS	[units]	[units]
$\mathcal{P}$ (with safety controller)	smrst / idle	stop	stop	37.06	0
	stop / off	stop	stop	37.06	0
	hguid / –	stop	stop	37.06	0
	smrst / idle	pflim	stop	44.75	9
	stop / off	pflim	stop	44.75	9
	hguid / –	pflim	stop	44.75	9
	smrst / idle	stop	pflim	46.26	49
	stop / off	stop	pflim	46.26	49
	hguid / –	stop	pflim	46.26	49
	smrst / idle	stop	ssmon	48.15	59
	stop / off	stop	ssmon	48.15	59
	hguid / –	stop	ssmon	48.15	59
	smrst / idle	pflim	pflim	49.90	58
	stop / off	pflim	pflim	49.90	58
	hguid / –	pflim	pflim	49.90	58
	smrst / idle	pflim	ssmon	51.42	68
	stop / off	pflim	ssmon	51.42	68
	hguid / –	pflim	ssmon	51.42	68
$\mathcal{P} \setminus sc$ (without safety controller)	–	–	–	64.19	291

†Configuration of mitigation options (safety mode and activity switches), *smrst*: safety-rated monitored stop, *stop*: power shutdown with user-initiated work cell reset, *hguid*: hand-guided operation, *pflim*: power & force limitation, *ssmon*: speed & separation monitoring, *idle*: currently not performing work steps, *off*: work cell not in operation.

‡According to $\mathbf{R}^{prod}_{\max=?}[\mathbf{C}^{\leq T}]$ and Property (Risk) with $T = 50$.

assuming no safety requirements, which is for obvious reasons not desirable. Perhaps unsurprisingly, safety comes at the cost of productivity. However, more importantly, among all configurations, the one switching to "stopped", whenever e^{HS}, e^{HC}, or e^{HRW} occurs, is the one performing the poorest (37.06, highlighted in grey). As expected, controllers implementing the pflim and ssmon safety modes in their policy are among the highest performing variants (51.42). With the best safety controller, the process achieves about 40% more productivity than with the worst controller and is only 25% less productive than the maximum productivity achievable in the process.

Moreover, the most risky configurations (68, highlighted in grey) are around 77% less risky than not using a safety controller. However, the most difficult trade-off is to be made between very low risk controllers (≤ 9) and moderate risk controllers (≥ 49) where the gain in productivity is comparatively little. Finally, with this model instance, the addition of the factor HC does neither influence risk nor productivity.

Regarding the expected overhead (Section 6.1), we used the C# profiling tool integrated in the development environment for the DTF. We measured the execution time of control cycles in the DTF across 1000 invocations of the controller. Our observations cover both nominal activity, and the occurrence of risk factors. The longest observed execution time $d^{\mathsf{HS,HC,HRW}}_{\max}$, its high watermark, is around 40 ms. The high watermark occurs upon activation of a risk factor, and accounts for safety mode and activity switch, and the required mitigation actions. In the absence of a mitigation requirement, the average execution time $d^{\mathsf{HS,HC,HRW}}$ is less than 1 ms. Execution times are expected to be lower in the physical twin. Profiling was performed on the same environment as for **RQ1b**, that is a 1.8 GHz i5 CPU with 8 GiB of RAM running Windows 10.

7.2.4. Results for RQ3: scalability

We answer this question in Table 4 by using six increments (the maximum for $|F| = 7$ in Table 1) applied to a basic risk model with $|F| = 1$. We only consider the **MDP setting** because applying model increments in the **pDTMC setting** does not provide additional insights into the scalability aspect of our method.

For demonstration of YAP's capabilities, the incident RT and the accident RC are included in the risk model without handler commands. However, these factors add further constraints on $R(F)$ to be dealt with by the safety controller. Hence, *mr* stays at 15 actions and *c* rises to 15 constraints. In model 7 (last line of Table 4), $R(F)$ (122 risk states) and $[\![\Xi]\!]_S$ (12079 states) differ by two orders of magnitude. Risk states offer a higher level of abstraction for risk assessment. The derivation of properties that focus on relevant regions of the MDP state space from a risk structure can ease the state explosion problem in explicit model checking. For example, the constraints HRW prevents HC and HC prevents HRW express that the combined occurrence of HC and HRW is considered infeasible or irrelevant by the safety engineer. Hence, checking properties of the corresponding region of the MDP state space can be abandoned.

7.3. Threats to validity

Internal validity. Too strong environmental assumptions reduce the scope of the safety guarantee. To limit the need for game-theoretic reasoning about $\mathcal{M}$, we reduce non-deterministic choice for the environment (i.e., operator, cobot, spot welder).

The more deterministic such choice, the closer is the gap between the policy set $\Pi_{\mathcal{M}}$ and the controller design space. Any decisions left to the environment will make a verified policy $\pi^\star$ safe relative to $\pi^\star$'s environmental decisions. These decisions form the assumption of the controller's safety guarantee.

Occupational health and safety assumes trained operators not to act maliciously, suggesting "friendly environments" with realistic human errors. To increase the priority of the controller in the **MDP-based approach**, we express realistic worst-case assumptions, for example, by minimising factor- and activity-based risk and maximising the risk reduction potential. The **pDTMC-based approach** weakens such assumptions by the fact that all outcomes of environmental decisions remain in the policy as uniformly distributed probabilistic choices. While, in the **MDP setting**, Prism's result from MDP policy synthesis is a DTMC containing worst-case environmental choices, in the pDTMC setting, evoChecker's result (a solution from fixing the parameters of the pDTMC) is a DTMC containing all environmental choices.[20] Hence, the controller extracted from $\pi^\star$ will contain an optimal choice for all states reachable from environmental decisions. Keeping environmental decisions is important, but uniform distributions can influence or bias the optimisation to a certain extent. We have not been able to investigate this further in our case study.

For quantitative analysis in evoChecker (Section 5.2), all non-deterministic decisions in the original MDP are kept in the DTMC as uniformly distributed probabilistic choices. These decisions include decisions by the environment (e.g., operator and cobot) but also decisions by the controller not captured by decision parameters, that is, decisions about the interleaving and finite repetition of idempotent mitigation actions. The alternative of a reduction of the original MDP (allowing non-deterministic choice) to a pDTMC with all non-deterministic controller choices replaced by decision parameters is non-trivial for the considered case study and was, hence, not pursued.

Decisions of the environment lead to a controller with a weaker assumption about its environment, that is, a controller that can be used across a wider range of scenarios. Action order decisions by the controller are resolved during the translation into executables: the first enumerated option is chosen for each decision. Obtaining appropriate trade-offs between productivity and safety is challenging. Hence, in the future we may switch to game-theoretic approaches which may allow for a more nuanced view of the environment and hence allow for even greater productivity.

Incorrect application of tools. First, for co-safe LTL reward properties (e.g., $\mathbf{R}^q[\mathbf{F}\,\phi]$), the expected reward is defined to be infinite if the probability of satisfying ϕ is less than 1. In our methodology, we avoid side-effects due to this issue by only using cumulative ($\mathbf{R}^q[\mathbf{C}^{\leq t}]$) and total ($\mathbf{R}^q[\mathbf{C}]$) rewards, where this problem does not appear. Hence, our methodology should not be affected by this issue. Second, evoChecker relaxes constraints if the constrained solution space is empty, leading to solutions under relaxed constraints. Hence, the design space needs to be redefined in order for evoChecker to select from a non-empty set of solutions obeying safety-related hard constraints.

Numerical errors. The quantitative reasoning we rely on in our verification and optimal synthesis is only as exact as the numerical reasoning (e.g., floating-point precision) employed in the tools we use. The safety assessment of numerical errors is a more general problem beyond the scope of our work. Part of this issue is dealt with by adding conservative error margins on the used numbers, see the mitigation of "inadequate quantification" below.

Insufficient testing. Confidence in the use case-based integration testing depends on the range and complexity of (mis)use cases and the parameters used for randomisation. The use case for **RQ1b** is generic enough to cover a wide range of work cell scenarios and the misuse case clarifies the environmental assumptions of the use case.

External validity. Reality gap through inadequate quantification. The lab replica of the work cell matches closely to the one in the company. This allowed us to develop an accurate digital twin and an MDP compliant with the actual work cell that incorporates all relevant parameters, particularly, probabilities. We crafted the MDP such that assessment and optimisation based on probabilities and rewards rely more on qualitative relations between the parameters rather than accurate numbers. It is more important and easier to conservatively assess whether an activity or safety mode is more or less dangerous than another and it is less important and more difficult to provide accurate numbers. The resulting gap between optimality in the model and optimality in the digital twin should hence not alter the demonstrated confidence in the controller. More accurate numbers can be obtained from observations, and after a model update, a repeated synthesis can improve productivity of the safety controller.

Incorrect sensing assumptions. In our case study, the safety controller relies on the detection of an operator (e.g., extremities, body) and a robot (e.g., arm, effector) entering a location, the cell state (e.g., grabber occupied, workbench support filled), and the workpiece location (e.g., in grabber, in support). For $\mathcal{M}$, we assume the tracking system (i.e., range finder and light barrier in the industrial setting, MS Kinect[21] in the lab replica) to map the location of the operator and robot to the areas "in front of the workbench", "on the workbench", "in the workcell", and "close to the spot welder" rather than to a fine-grained occupancy grid. In Fig. 1b, the range finder signals "at welding spot" if the closest detected object is nearer than the close range, and "in cell" if the closest object is nearer than the wide range. Tracking extensions, not discussed here, could include object silhouettes and minimum distances, operator intent, or joint velocities and forces.

Incorrect real-time behaviour. pGCL, as we used it, requires care with the modelling of real-time behaviour, particularly, when actions from several concurrent modules are enabled. To model real-time controller behaviour, we use Prism's synchronous parallel composition to synchronise actions of an operator module with events from a sensor module. In the

[20] Actually, instead of a pDTMC, one could also use a parametric MDP and fix the model parameters to obtain an MDP.

[21] See https://en.wikipedia.org/wiki/Kinect.

original **MDP setting**, we force the priority of controller reactions in $\pi^\star$ by maximising the risk reduction potential (cf. *pot* in Table 3). Individual pGCL actions can operate on local and global variables. However, actions synchronised between different modules cannot alter global variables, because this would lead to conflicts between synchronised actions that want to alter the same variable at the same time. Hence, synchronised actions need to operate on variables local to their modules, leading to further state variables and increasing $\mathcal{M}$'s state space. However, we found synchronisation to be the best solution to model real-time behaviour in the multi-module **MDP setting**. The alternating execution scheme used in the **pDTMC setting**, however, avoids the use of rewards to implement priorities and prevents actors from competing in an unnatural way.

Generalisation to other human-robot collaboration settings. Risk structures and the synthesis approach itself are generic. However, the process model, the activity model, the safety modes and functions, and the integration with the digital twin for testing are clearly focusing on the robotic manufacturing domain. More work is needed to adapt these specific aspects to other application domains.

7.4. Discussion

Tool restrictions and model debugging. State rewards allow a natural modelling of, for example, risk exposure, particularly in continuous-time Markov chains where the time residing in a state is multiplied with the risk associated with that state. However, in Prism 4.5, one is required to use action rewards to solve multi-objective optimisation problems over MDPs. Risk gradients help to overcome a minor restriction in Prism's definition of action rewards.[22] Alternatively, we could have introduced extra states at the cost of increasing $\mathcal{M}$'s state space, undesirable for synthesis.

Multi-objective model checking of MDPs for maximising cumulative rewards requires the elimination of non-zero reward end components (i.e., deadlocks or components with cycles that allow infinite paths and, hence, infinite reward accumulation). Prism provides useful facilities to identify such components. However, their elimination is non-trivial and laborious in larger models and can require intricate model revisions.

We strongly discretise model parameters such as location to further reduce the state space and keep the model small. In the **MDP setting**, we use probabilistic choice in synchronous updates only in one of the participating commands to simplify debugging. We avoid global variables to support synchronisation with complex updates.

Misuse cases for controller testing. Misuse cases help in exploiting deficiencies of a control concept through counterexamples. The latter can be generated from the process model and exercised as negative test cases to aid in debugging the controller. We explored this idea in [52], applying a Prolog-based Golog interpreter that constructs counterexamples from backward depth- and breadth-first search from a given accident state. PRCTL model checkers, such as Prism, also employ forward breadth-first search from an initial state and stopping when reaching an accident state. In both cases, explicit state exploration is used with the usual problem of state space explosion. In any case, an environment model is needed to express accident states. In this work, we prefer a model checker because the encoding of the stochastic process in pGCL appeared to be easier than in a stochastic situation calculus and because of more flexibility of expressing properties to be verified in PRCTL.

8. Related work

Our work builds upon a set of well known theoretic principles on which research on controller design and synthesis [27] for collaborative robots has been carried out. As mentioned in Section 1, we improve our previous work [12,16] by an enhanced process and risk model with a refined notion of risk factors, the use of further tools to extend the synthesis capabilities, a translation of synthesised controllers into executable code, a stage for testing the integration of the controller into a realistic digital twinning environment, and a more comprehensive evaluation. In the following, we compare our results with other results.

Risk-informed controller design and verification. Askarpour et al. [53] discuss controller assurance of a cobot work cell based on a discrete-event formalisation in the linear-time temporal language TRIO. Actions are specified as *pre*/*inv*/*post*-triples for contract-based reasoning with the SAT solver Zot. Violations of the safety invariant *inv* lead to pausing the cell. Fine-grained severity quantification [54] allows a controller to trigger safety measures on exceeding of certain risk thresholds. Additionally, Askarpour et al. [55] present a model of erroneous or non-deterministic operator behaviour to enable designers to refine controller models until erroneous behaviours are mitigated. This approach aims at risk-informed prototyping and exhaustive exploitation of all possible executions to identify constraint violations and remove hazardous situations. Risk thresholds correspond to a refined variant of Property (11). Whilst not the focus of this work, our approach also allows for the quantification of severity in detector predicates (ζ). Their severity model [55,54] inspires future risk factor models. Instead of a priority parameter, which reduces state variables, we use guards to implement action orderings. Our approach is more flexible because it can deal with multiple mitigation options offering more variety in safety responses. Beyond model consistency checks and the search of counterexamples for model repair, our approach yields an executable policy. Our use

[22] Currently, rewards cannot be associated with particular updates, that is, with incoming transitions rather than only states.

of pGCL and PRCTL [32] results in a separation of action modelling and property specification. Although this separation is a non-essential difference, it syntactically supports a more independent working on two typical abstraction levels, required process properties and process implementation.

Verified controller synthesis for cobots. A number of authors have suggested synthesis approaches for controllers in human-robot collaboration. Key features of these approaches are verified optimal synthesis for collaborative plan execution, quantitative verification of plans, code generation for deployment on robot platforms.

For generic robots, Orlandini et al. [56] and Bersani et al. [57] employ synthesis by game solving (i.e., finding winning strategies) over timed game automata (TIGA) supported by the model checker UPPAAL-TIGA. Correctness properties can be formulated as reach-avoid problems (i.e., reach the goal state and avoid unsafe states) specified as $\mathbf{A}[\mathit{safe}\,\mathbf{U}\,\mathit{goal}]$ in timed computation tree logic. From a timeline-based plan description [58], Orlandini et al. [56] generate a TIGA with clock constraints encoding temporal degrees of freedom for performing control actions. From the TIGA, a winning strategy (i.e., a robust plan execution minimising violations of clock constraints) is then synthesised. The TIGA-based stage is continuously performed during operation. The distinction of controllable (i.e., duration known) from uncontrollable actions (i.e., duration unknown) allows one to react to temporally uncertain environmental events by obtaining strategies that can schedule robot actions in the presence of worst-case timing of the environment. In [59], an extension of this approach is applied to controller synthesis for safe human-robot collaboration. For task coordination between humans and robots, Cesta et al. [59] and other works utilise the distinction of uncontrollable and controllable actions. Our reactive execution scheme (Fig. 5) accommodates this feature required to separate the cobot capabilities from the capabilities of its physical environment.

Kshirsagar et al. [60] demonstrate on-line controller synthesis for human-robot handovers obeying timing constraints specified in signal temporal logic. Cobot kinetics need to be given in terms of ordinary differential equations. Being suitable for low-level synthesis in homogeneous action systems, this approach could be integrated into our framework, for example, for controlling a speed and separation monitoring mode switched on during a handover.

MDP policy synthesis has also been used to generate optimal motion plans for mobile robots [61]. The MDPs used in this solution model the physical layout of the space within which a mobile robot can navigate. As such, the problem addressed in this work is complementary to our use of probabilistic models and MDP policy synthesis, as our approach tackles the generation of optimal hazard mitigation actions.

Modelling of controllers for cobots. Domain-specific languages for controller design can support control engineers in encoding their control schemes. Cesta et al. [59] employ the domain and problem definition languages DDL and PDL for encoding the state spaces and action domains a controller can select from. Bersani et al. [57] provide a lean language for robotic controller design. Models in that language typically include a description of the uncontrollable environment. Safety properties can be difficult to formulate, as shown in [57] and, particularly, when dealing with multiple hazards or risk factors. We provide YAP's input language for safety controller design informed by risk models, streamlining the specification of safety properties from multiple hazards.

The approaches by Lahijanian et al. [61] and Kshirsagar et al. [60] focus on *homogeneous action systems*, that is, systems with few and similar types of actions manipulating type-wise simple state spaces (e.g., movement in a Euclidean plane or in a 2D grid). Such systems make it easier to provide complete and tractable synthesis algorithms for realistic models. In contrast, and as suggested by Kress-Gazit et al. [27], our approach focuses on *heterogeneous action systems*, such as human-robot collaboration, which can be characterised by a wide variety of actions over a heterogeneously typed state space. Such actions can differ in their complexity and discrete or continuous nature (e.g., grab a workpiece, move a robot arm, perform a welding action, switch a mode, turn off an alarm sound).

Robust controllers are able to *deal with unstructured environments*, to react to a wide variety of uncontrollable adverse events (e.g., human error). Overall, they guarantee correct behaviour under weak assumptions. Game-based approaches, such as policy synthesis for TIGAs [62,56,57], inherently support such environments. Whilst our MDP-based approach benefits from more efficient algorithms and provides fine-grained methodological guidance, an extension to utilise the greater flexibility of game-based approaches should be part of our next steps.

A distinctive feature of safety controllers is their ability to *control the resumption of normal operation* (i.e., the recovery from a conservative or degraded safe state) in addition to mitigation. While resumption is implicit to many solutions for homogeneous-action reach-avoid problems [56,57], for heterogeneous action systems, resumptions often need to be modelled separately (e.g., switching off a safety mode or function, resuming/restarting a suspended/cancelled task). Risk factors at the core of our approach accommodate primitives for specifying resumptions.

Overall, an advantage of timeline-based approaches [59] is their ability to encode physical domains qualitatively. In pGCL, corresponding domain constraints need to be distributed over action guards, which can be cumbersome. Moreover, an advantage of TIGAs over MDPs is that time is continuous and managed via clocks, leading to more concise models. However, in summary, our use of pGCL enables the concise encoding of the action domain of human-robot collaboration with multiple actors operating over a discrete state space. MDPs represent a natural model of interaction of agents with an uncontrollable and uncertain environment. PRCTL provides a flexible and expressive language for specifying multiple objectives (e.g., minimum risk, maximum performance) and reward constraints (e.g., accepted risk thresholds). We could enhance our approach to probabilistic timed automata in order to further increase model fidelity.

Controller deployment. An important step in many synthesis approaches is the deployment of the resulting controllers in an execution environment. Although for another domain (i.e., climate control in a pig stable), Jessen et al. [62] show how controllers synthesised using UPPAAL-TIGA can be embedded as components of a Simulink model to perform validation by simulation. As part of their evaluation, Bersani et al. [57] demonstrate the deployment of their controllers on the low-cost platform Turtlebot used as a cobot. Orlandini et al. [56] deploy their approach as a module on a GenoM-based robotic software platform with a mapping into the real-time framework BIP. Their flexible approach could be a potential route for future extensions of our work. Currently, YAP provides a generator for C# modules for the DTF (Sections 3.6 and 6.2).

To the best of our knowledge, our method is the first end-to-end approach to synthesising and deploying safety controllers for handling multiple risks from collaborative robots in manufacturing processes. The works discussed above either address parts of the synthesis challenge or implement alternative solutions for cobot (safety) controllers.

9. Conclusion

We introduced an integrated and tool-supported software engineering approach for the verified synthesis of optimal safety controllers from Markov decisions processes, focusing on human-robot collaboration. These software controllers implement regulatory safety goals for such applications. We describe steps for streamlined application modelling and risk-informed controller design and demonstrate our method using a tool chain consisting of YAP [15] for structured risk modelling and pGCL program generation, EVOCHECKER [17] for search-based policy synthesis from pDTMCs, and PRISM [18] for probabilistic model checking and MDP policy synthesis. We show that our approach can be used to incrementally build up multi-hazard models including alternative mitigation and resumption strategies. We also discuss how our approach can simplify explicit model checking when dealing with large state spaces. Our approach improves the state-of-the-art of controller synthesis for collaborative robots, particularly when dealing with multiple risks, mitigation options, activities and safety modes. That way, we contribute to the alignment of lower-level requirements posed by cobot safety standards (e.g., ISO/TS 15066 [20]) with higher-level cobot hazard analysis and risk assessment [4] through a structured risk-informed design approach. Furthermore, we translate the verified controllers into executable code and deploy them in the Digital Twin Framework [19] and, thus, accomplish a smooth transition between formal controller verification and controller integration testing in a realistic environment. Using this framework, we demonstrate the controller's correct and timely response in a representative use case. The verification and test results generated by our approach can contribute evidence to a controller assurance case [63–65]. The proposed method provides a layer of abstraction by integrating the mentioned engineering steps (i.e., risk modelling, controller design), formal modelling techniques (i.e., pGCL, MDP), and tools (i.e., YAP, PRISM, EVOCHECKER).

Future work. For optimal synthesis, the proposed method uses parameters such as upper risk and severity bounds as constraints. We plan to introduce parameters for probabilities, such as sensor failure and human error, into the MDP and to use parametric risk gradients by extending YAP.

It is important to model all relevant behaviours of the environment, or more generally the uncontrollable actions, the safety controller needs to respond to in order to increase freedom from accidents. Hence, in future work, we plan to use stochastic games to more accurately (and less conservatively) model the behaviour of the environment in which this controller operates. We also plan to explore online policy synthesis [66] to allow more variety in environmental decisions (e.g., mitigating hazards due to malicious operators). This corresponds to weakening the assumptions under which the controller can guarantee safety.

The case study can be extended by randomised control decisions with fixed probabilities (e.g., workload), by adding uncertain action outcomes (e.g., welding errors), and by time-dependent randomised choice of mitigation options. To use time in guarded commands, we want to explore clock-based models rather than only using reward structures, as far as the synthesis capabilities allow this.

For motion planning over finite partitions of geometric spaces, Lahijanian et al. [61] describe algorithms for the synthesis of MDP policies that maximise the probability of a given arbitrary PRCTL formula. By assuming that state estimation is precise, the authors avoid the use of partially observable MDPs in their application. To improve our approach regarding the synthesis over homogeneous action systems, their algorithms could be integrated into our MDP synthesis tool chain, for example, in addition to PRISM or underpinning the search-based synthesis in EVOCHECKER.

The modelling of human errors and sensor failures in Section 4.3 provides a template for the modelling of human sabotage or integrity attacks to sensors, actuators, controller code, and other components. It would be interesting to extend our approach to align it with recent results from cyber-physical systems security.

Accident data for the considered industrial application was generally not available. We were also unable[23] to collect data from the lab replica. Thus, we had to make best guesses of probabilities (cf. Section 4.3). However, the frequency of undesired intrusion of operators into the safeguarded area and accident likelihood can be transferred into our case study. The digital twin framework enables a full-fledged digital twin, from which the interaction between the safety controller and

[23] Due to restricted lab access during the COVID-19 pandemic.

Table 7
Table of symbols and concepts (excluding locally used parameters and standard notation).

Notation	Description
Generic MDP Modelling	
$A_{\mathcal{P}}$	action alphabet of process $\mathcal{P}$
A	map of states, factors, or phases to available actions and mitigation options, $\forall s \in S\colon A(s) \subseteq A_{\mathcal{P}}$
AP, ap	set of atomic propositions, atomic proposition
$\delta_{\mathcal{P}}, \delta_{\mathcal{P}}^{\rightarrow}$	probabilistic transition function and transition relation of an MDP generated by program $\mathcal{P}$
L	MDP state labelling function
$\mathcal{M}$	MDP model of $\mathcal{P}$
$\mathcal{M}[\mathcal{P}]$	MDP $\mathcal{M}$ generated by $\mathcal{P}$ from state s_0
$\Pi_{\mathcal{M}}, \Pi^{\star}_{\mathcal{M}}$	policy set for an MDP $\mathcal{M}$, Pareto subset of $\Pi_{\mathcal{M}}$
$\pi, \pi^{\star}$	MDP policy, optimal policy; with $\pi, \pi^{\star} \in \Pi_{\mathcal{M}}$
rw^{q}_{action}	reward structure, map of states, actions (and risk factors) to rewards of quantity q (e.g., risk ρ, nuisance)
S	state space of an MDP
s, s_0	(initial) state in S, valuation of all variables in X
PRCTL Syntax and Semantics	
b, b_{ρ}, b_s	pos. integer/real bound for quantities (e.g., risk, severity)
$\phi, \phi_{\text{path}}, \psi$	PRCTL formulae
ϕ_{cor}, ϕ_{wf}	correctness and well-formedness, expressed in PRCTL
$[\![\phi]\!]_X$	subset of a state space X (e.g., S, $R(F)$) where ϕ holds
$\mathbf{A}, \mathbf{E}$	temporal branching operator
$\mathbf{C}^{[\sim b]}$	operator for [b-bounded] accumulation of rewards
$\mathbf{F}, \mathbf{G}, \mathbf{X}, \mathbf{W}, \mathbf{U}$	temporal path operator
ι	interval over the reals
$\mathbf{P}, \mathbf{S}$	quantification operator for (long-run) probabilities
$\mathbf{R}$	operator for calculating rewards
t	upper time bound, time stamp
Generic pGCL Guarded Command Modelling	
α	event label (e.g., action name) in $A_{\mathcal{P}}$
$\gamma, \gamma_{sm}, \gamma_{act}$	quantifier-free formula, e.g., guarding safety mode or activity
υ	update expression (prob. choice and multiple assignment)
pr	effect probability; in PRCTL formula: probability bound
u	a function of state variables X for calculating updates
$X, X', Y; x$	sets of state variables of $\mathcal{M}$; a (state) variable in X
pGCL Modelling (process, activities, scheduler)	
$\mathcal{A}$	set of actors part of $\mathcal{P}$, some specified in pGCL
Act, act	set of activities, an activity in Act
$next(p)$	pass the token from actor $p \in \mathcal{A}$ to another actor in $\mathcal{P}$
ok_p	predicate guarding the execution of actor $p \in \mathcal{A}$
$\mathcal{P}$	a process/program, a composition of pGCL commands
$\mathcal{P} \setminus \mathcal{Q}$	program $\mathcal{P}$ without program $\mathcal{Q}$
SF, sf	set of safety functions, a safety function in SF
SM, sm	set of safety modes, a safety mode in SM
$\mathcal{T}_{Act}$	set of activity tuples, $\mathcal{T}_{Act} \subseteq Act^{\lvert\mathcal{A}\rvert}$
tn	special variable for managing turns
Model Assessment and Evaluation (risks, accidents)	
d	map of actions to controller processing time
d_P, d_Y	duration of calculations done by PRISM, YAP
$deadlock$	propositional formula denoting deadlocked states
$final$	propositional formula denoting final states
$goal$	goal state, subset of $final$ with all tasks finished
$init$	propositional formula denoting initial states
$\lfloor\mu\rceil$	[min, mean, max]-probability triple resulting from $\mathbf{P}_{\neg\underline{F}}$
$\mathbf{P}_{\neg\underline{F}}$	quantify freedom from accidents $\underline{F}$ in $\mathcal{M}$
ρ	risk, used both as a quantity in rw and in $\nabla\rho$

Notation	Description
Risk Structures (modelling primitives)	
e	endangerment (critical event)
$\underline{e}$	mishap event
f	risk factor
$\overset{\vee}{\wedge}$	inactive phase of f
$\vec{f}, \vec{f}', \vec{f}''$	active phases of f
$\bar{f}, \bar{\vec{f}}$	mitigated phases of f
$\underline{f}$	mishap phase of f
F	set of risk factors
$\underline{F}$	mishap states, $[\![\underline{F}]\!]_{R(F)} \subset R(F)$, $[\![\underline{F}]\!]_S \subset S$
$\overset{\vee}{F}$	F-inactive region, $[\![\overset{\vee}{F}]\!]_{R(F)} \subset R(F)$, $[\![\overset{\vee}{F}]\!]_S \subset S$
m^{f}	mitigation action (f omitted if clear from context)
$m^{\mathsf{f}}_{sm}, m^{\mathsf{f}}_{act}, m^{\mathsf{f}}_{sf}$	mitigation to safety mode, to activity, using safety function
$\nabla\rho$	categorical risk gradient
$Ph_F, Ph_{\mathsf{f}}, ph$	phase set of factor set F, factor f, a phase
$R(F)$	risk space for factor set F
r^{f}	resumption action (f omitted if clear from context)
$r^{\mathsf{f}}_{sf}, r^{\mathsf{f}}_{sm}, r^{\mathsf{f}}_{act}$	resumption from safety function, from mode, to activity
s_{ρ}	risk state in $R(F)$
Risk Structures (MDP semantics)	
χ	state predicate denoting a cause
Ξ	states labelled with at least one risk factor, non-accident F-unsafe region $[\![\Xi]\!]_S \subset S$
Ξ_f	states denoting causes, $[\![\Xi_f]\!]_S \subset S$
Safety Controller pGCL Modelling	
κ	state predicate modelling a detectable causal factor
ζ	state predicate modelling the sensing of a factor
Safety Controller Implementation	
$\delta_{sc}^{\rightarrow}$	controller transition relation, $\delta_{sc}^{\rightarrow} \subseteq \delta_{\mathcal{P}}^{\rightarrow}$
S_{sc}	controller state space, $S_{sc} \subseteq S$ reachable from s_0
Safety Controller Deployment	
dt, n, pc, p_i, sc	actors in $\mathcal{A}$, e.g., digital twin, network, process controller, process actor, safety controller
$\text{IF}(p)$	interface of actor $p \in \mathcal{A}$, $\text{IF}(p) = (I_p, O_p)$
I_p	p's input ports $\{i^1_p, \ldots, i^l_p\}$ receiving messages from $\mathcal{A}$
O_p	p's output ports $\{o^1_p, \ldots, o^l_p\}$ sending messages to $\mathcal{A}$
Safety Controller Testing (Section 6.3)	
$\approx_U$	a conformance relation
ϕ, ϕ_m, ϕ_r	(mitigation, resumption) requirement, expressed in PRCTL
ϕ_{MITL}	ϕ translated into MITL to be checked of a timed path $\omega \in \mathcal{T}_{\mathcal{V}}(U)$
ω	path (state sequence) recorded of $\mathcal{P}$
$MCobot$	an implementation of $\mathcal{P}$
MU	misuse case
Sit^*	situations, set of states or state/transition combinations
$\mathcal{T}_{\mathcal{V}}(U)$	paths in vicinity of U
U	use case
$\mathcal{V}$	a predicate defining vicinity to U (e.g., value ranges)

the real system can be demonstrated. Limited access to the physical cell has required testing in the digital twin to take precedence. Further tests integrating the physical work cell will therefore also be the focus of future work.

CRediT authorship contribution statement

Mario Gleirscher: Conceptualisation (risk structures, controller synthesis, formalisation), data curation, formal analysis, methodology, software/code implementation, supervision, validation, visualisation, original draft preparation, review & edit-

ing. **Radu Calinescu**: Conceptualisation (controller synthesis), funding acquisition, investigation, project administration, visualisation, review & editing. **James Douthwaite**: Data curation, methodology, project administration, resources, software/code implementation, visualisation, original draft preparation, review & editing. **Benjamin Lesage**: Data curation, investigation, software/code implementation, resources, visualisation, original draft preparation. **Colin Paterson**: Data curation, validation, original draft preparation, review & editing. **Jonathan Aitken**: Funding acquisition, software/code implementation, supervision. **Rob Alexander**: Funding acquisition, supervision. **James Law**: Conceptualisation (case study), funding acquisition, project administration, resources, supervision, review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This research has received funding from the Assuring Autonomy International Programme (AAIP grant CSI: Cobot), a partnership between Lloyd's Register Foundation and the University of York, and from the UKRI project EP/V026747/1 'Trustworthy Autonomous Systems Node in Resilience'.

We are grateful for the insights into manufacturing cobots gained from our industrial collaborator and being useful for the integration of the work on verified controller synthesis and situation-based testing at the University of York with the digital twinning research at the University of Sheffield. We would like to thank Simos Gerasimou for his guidance on using EvoChecker and David Parker for his advice in the use of Prism's MDP policy synthesis functionality. We are also grateful to Jan Peleska for discussions and feedback on formal parts of the manuscript and to Andrea Orlandini for further more general feedback on our work.

References

[1] P. Nicolaisen, Occupational safety and industrial robots, in: Yong Bonney (Ed.), Robot Safety, IFS, 1985, pp. 33–48.
[2] R.H. Jones, A Study of Safety and Production Problems and Safety Strategies Associated with Industrial Robot Systems, Ph.D. thesis, Imperial College, 1986.
[3] A.D. Santis, B. Siciliano, A.D. Luca, A. Bicchi, An atlas of physical human–robot interaction, Mech. Mach. Theory 43 (2008) 253–270.
[4] P. Chemweno, L. Pintelon, W. Decre, Orienting safety assurance with outcomes of hazard analysis and risk assessment: a review of the ISO 15066 standard for collaborative robot systems, Saf. Sci. 129 (2020) 104832.
[5] B. Hayes, B. Scassellati, Challenges in shared-environment human-robot collaboration, in: Collab. Manipulation Workshop at HRI, 2013, pp. 1–6.
[6] V. Villani, F. Pini, F. Leali, C. Secchi, Survey on human-robot collaboration in industrial settings: safety, intuitive interfaces and applications, Mechatronics 55 (2018) 248–266.
[7] R. Alami, A. Albu-Schaeffer, A. Bicchi, R. Bischoff, R. Chatila, et al., Safe and dependable physical human-robot interaction in anthropic domains: state of the art and challenges, in: IEEE/RSJ Int. Conf. Intelligent Robots and Systems, 2006, pp. 1–16.
[8] S. Haddadin, A. Albu-Schäffer, G. Hirzinger, Requirements for safe robots: measurements, analysis and new insights, Int. J. Robot. Res. 28 (2009) 1507–1527.
[9] A. Ajoudani, A.M. Zanchettin, S. Ivaldi, A. Albu-Schäffer, K. Kosuge, O. Khatib, Progress and prospects of the human-robot collaboration, Auton. Robots 42 (2017) 957–975.
[10] R.B. Gillespie, J.E. Colgate, M.A. Peshkin, A general framework for cobot control, IEEE Trans. Robot. Autom. 17 (2001) 391–401.
[11] G. Anderson, The Economic Impact of Technology Infrastructure for Advanced Robotics, Economic Analysis Briefs, NIST, 2016, https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=921956.
[12] M. Gleirscher, R. Calinescu, Safety controller synthesis for collaborative robots, in: Engineering of Complex Computer Systems (ICECCS), 25th Int. Conf., Singapore, 2020, pp. 83–92, arXiv:2007.03340.
[13] M. Gleirscher, Run-time risk mitigation in automated vehicles: a model for studying preparatory steps, in: 1st iFM Workshop Formal Verification of Autonomous Vehicles, FVAV, in: EPTCS, vol. 257.8, 2017, pp. 75–90.
[14] M. Gleirscher, R. Calinescu, J. Woodcock, Risk structures: a design algebra for risk-aware machines, Form. Asp. Comput. (2021).
[15] M. Gleirscher, Yap Against Perils: Application Guide and User's Manual, University of Bremen and University of York, 2021, https://yap.gleirscher.de/dl/yap-0.7-manual.pdf.
[16] M. Gleirscher, Yap: tool support for deriving safety controllers from hazard analysis and risk assessments, in: M. Luckuck, M. Farrell (Eds.), Formal Methods for Autonomous Systems (FMAS), 2nd Workshop, in: EPTCS, vol. 329, Open Publishing Association, 2020, pp. 31–47, arXiv:2012.01176.
[17] S. Gerasimou, R. Calinescu, G. Tamburrelli, Synthesis of probabilistic models for quality-of-service software engineering, Autom. Softw. Eng. 25 (2018) 785–831.
[18] M. Kwiatkowska, G. Norman, D. Parker, PRISM 4.0: verification of probabilistic real-time systems, in: 23rd CAV, in: LNCS, vol. 6806, Springer, 2011, pp. 585–591.
[19] J. Douthwaite, B. Lesage, M. Gleirscher, R. Calinescu, J.M. Aitken, R. Alexander, J. Law, A modular digital twinning framework for safety assurance of collaborative robotics, Front. Robot. AI 8 (2021) 402.
[20] ISO/TS 15066, Robots and robotic devices – Collaborative robots, Standard, Robotic Industries Association (RIA), https://www.iso.org/standard/62996.html, 2016.
[21] N. Sugimoto, Safety engineering on industrial robots and their draft standards for safety requirements, in: 7th Int. Symposium on Industrial Robots, 1977, pp. 461–470.
[22] X.V. Wang, Z. Kemény, J. Váncza, L. Wang, Human-robot collaborative assembly in cyber-physical production: classification framework and implementation, CIRP Ann. 66 (2017) 5–8.
[23] L. Kaiser, A. Schlotzhauer, M. Brandstötter, Safety-related risks and opportunities of key design-aspects for industrial human-robot collaboration, in: LNCS, Springer, 2018, pp. 95–104.

[24] B. Matthias, S. Kock, H. Jerregard, M. Kallman, I. Lundberg, Safety of collaborative industrial robots: certification possibilities for a collaborative assembly robot concept, in: IEEE Int. Symposium on Assembly and Manufacturing (ISAM), 2011, pp. 1–6.
[25] J.A. Marvel, J. Falco, I. Marstio, Characterizing task-based human-robot collaboration safety in manufacturing, IEEE Trans. Syst. Man Cybern. Syst. 45 (2015) 260–275.
[26] A. Avizienis, J.-C. Laprie, B. Randell, C. Landwehr, Basic concepts and taxonomy of dependable and secure computing, IEEE Trans. Dependable Secure Comput. 1 (2004) 11–33.
[27] H. Kress-Gazit, M. Lahijanian, V. Raman, Synthesis for robots: guarantees and feedback for robot behavior, Annu. Rev. Control Robot. Auton. Syst. 1 (2018) 211–236.
[28] ISO 10218, Robots and Robotic Devices – Safety Requirements for Industrial Robots, Standard, Robotic Industries Association (RIA), 2011, https://www.iso.org/standard/51330.html.
[29] E. Helms, R.D. Schraft, M. Hagele, rob@work: Robot assistant in industrial environments, in: 11th IEEE Int. Workshop on Robot and Human Interactive Communication, 2002, pp. 399–404.
[30] J. Heinzmann, A. Zelinsky, Quantitative safety guarantees for physical human-robot interaction, Int. J. Robot. Res. 22 (2003) 479–504.
[31] P. Long, C. Chevallereau, D. Chablat, A. Girin, An industrial security system for human-robot coexistence, Ind. Robot 45 (2018) 220–226.
[32] M. Kwiatkowska, G. Norman, D. Parker, Stochastic model checking, in: M. Bernardo, J. Hillston (Eds.), Formal Methods for the Design of Comp., Comm. and Soft. Sys.: Performance Evaluation (SFM), in: LNCS, vol. 4486, Springer, 2007, pp. 220–270.
[33] V. Forejt, M. Kwiatkowska, G. Norman, D. Parker, Automated verification techniques for probabilistic systems, in: M. Bernardo, V. Issarny (Eds.), Formal Methods for Eternal Networked Soft. Sys., in: LNCS, vol. 6659, 2011, pp. 53–113, tutorial.
[34] C. Baier, J.-P. Katoen, Principles of Model Checking, MIT Press, 2008.
[35] C. Dehnert, S. Junges, J.-P. Katoen, M. Volk, A storm is coming: a modern probabilistic model checker, in: R. Majumdar, V. Kunčak (Eds.), Computer Aided Verification, Springer, 2017, pp. 592–600.
[36] D. Bjørner, Domain engineering, in: Formal Methods: State of the Art and New Directions, Springer, 2009, pp. 1–41.
[37] S. Gerasimou, J. Camara Moreno, R. Calinescu, N. Alasmari, F. Alhwikem, X. Fang, Evolutionary-guided synthesis of verified Pareto-optimal MDP policies, in: 36th IEEE/ACM International Conference on Automated Software Engineering, 2021, pp. 1–12.
[38] N.G. Leveson, Engineering a Safer World: Systems Thinking Applied to Safety, Engineering Systems, MIT Press, 2012.
[39] N.G. Leveson, Safeware: System Safety and Computers, Addison-Wesley, 1995.
[40] R. Alur, T. Feder, T.A. Henzinger, The benefits of relaxing punctuality, J. ACM 43 (1996) 116–146.
[41] E. Negri, L. Fumagalli, M. Macchi, A review of the roles of digital twin in CPS-based production systems, Procedia Manuf. 11 (2017) 939–948.
[42] A. Bolton, L. Butler, I. Dabson, M. Enzer, M. Evans, T. Fenemore, F. Harradence, The Gemini Principles, Technical Report, Centre for Digital Built Britain, University of Cambridge, Cambridge, UK, 2018, https://www.cdbb.cam.ac.uk/system/files/documents/TheGeminiPrinciples.pdf.
[43] W. Kritzinger, M. Karner, G. Traar, J. Henjes, W. Sihn, Digital twin in manufacturing: a categorical literature review and classification, IFAC 51 (2018) 1016–1022.
[44] F. Tao, J. Cheng, Q. Qi, M. Zhang, H. Zhang, F. Sui, Digital twin-driven product design, manufacturing and service with big data, Int. J. Adv. Manuf. Technol. 94 (2018) 3563–3576.
[45] N.A. Stanton, Hierarchical task analysis: developments, applications, and extensions, Appl. Ergon. 37 (2006) 55–79.
[46] M.B. Dwyer, G.S. Avrunin, J.C. Corbett, Patterns in property specifications for finite-state verification, in: ICSE, 1999, pp. 411–420.
[47] D. Parnas, J. Madey, Functional documentation for computer systems, Sci. Comput. Program. 25 (1995) 41–61.
[48] M. Broy, A logical basis for component-oriented software and systems engineering, Comput. J. 53 (2010) 1758–1782.
[49] M. Vazquez-Chanlatte, mvcisback/py-metric-temporal-logic: v0.1.1, https://doi.org/10.5281/zenodo.2548862, 2019.
[50] D. Griffin, I. Bate, R.I. Davis, Generating utilization vectors for the systematic evaluation of schedulability tests, in: IEEE Real-Time Systems Symposium, RTSS 2020, Houston, Texas, USA, IEEE, 2020, pp. 76–88, https://www-users.cs.york.ac.uk/~robdavis/papers/DRSRTSS2020.pdf.
[51] R. Alexander, H. Hawkins, A. Rae, Situation coverage – a coverage criterion for testing autonomous robots, volume Report number YCS-2015-496, Department of Computer Science, University of York, 2015.
[52] M. Gleirscher, Hazard-based selection of test cases, in: Automation of Software Test (AST), 6th ICSE Workshop, 2011, pp. 64–70.
[53] M. Askarpour, D. Mandrioli, M. Rossi, F. Vicentini SAFER-HRC, Safety analysis through formal vERification in human-robot collaboration, in: LNCS, Springer, 2016, pp. 283–295.
[54] F. Vicentini, M. Askarpour, M.G. Rossi, D. Mandrioli, Safety assessment of collaborative robotics through automated formal verification, IEEE Trans. Robot. 36 (2020) 42–61, Conference Name: IEEE Transactions on Robotics.
[55] M. Askarpour, D. Mandrioli, M. Rossi, F. Vicentini, Formal model of human erroneous behavior for safety analysis in collaborative robotics, Robot. Comput.-Integr. Manuf. 57 (2019) 465–476.
[56] A. Orlandini, M. Suriano, A. Cesta, A. Finzi, Controller synthesis for safety critical planning, in: Tools with Artificial Intelligence (ICTAI), 25th Int. Conf., IEEE, 2013, pp. 1–8.
[57] M.M. Bersani, M. Soldo, C. Menghi, P. Pelliccione, M. Rossi, PuRSUE - from specification of robotic environments to synthesis of controllers, Form. Asp. Comput. 32 (2020) 187–227.
[58] A. Cesta, S. Fratini, The timeline representation framework as a planning and scheduling software development environment, in: Proc. of 27th Workshop of the UK Planning and Scheduling SIG, 2008, pp. 1–8, https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.432.1860&rep=rep1&type=pdf.
[59] A. Cesta, A. Orlandini, G. Bernardi, A. Umbrico, Towards a planning-based framework for symbiotic human-robot collaboration, in: Emerging Technologies and Factory Automation (ETFA), 21st Int. Conf., IEEE, 2016, pp. 1–8.
[60] A. Kshirsagar, H. Kress-Gazit, G. Hoffman, Specifying and synthesizing human-robot handovers, in: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, Macau, China, 2019, pp. 5930–5936, https://ieeexplore.ieee.org/document/8967709/.
[61] M. Lahijanian, S.B. Andersson, C. Belta, Temporal logic motion planning and control with probabilistic satisfaction guarantees, IEEE Trans. Robot. 28 (2012) 396–409, Conference Name: IEEE Transactions on Robotics.
[62] J.J. Jessen, J.I. Rasmussen, K.G. Larsen, A. David, Guided controller synthesis for climate controller using UPPAAL tiga, in: J.-F. Raskin, P.S. Thiagarajan (Eds.), Formal Modeling and Analysis of Timed Systems, vol. 4763, Springer, Berlin, Heidelberg, 2007, pp. 227–240.
[63] M. Gleirscher, S. Foster, Y. Nemouchi, Evolution of formal model-based assurance cases for autonomous robots, in: 17th Int. Conf. SEFM, in: LNCS, vol. 11724, Springer, 2019, pp. 87–104.
[64] S. Foster, M. Gleirscher, R. Calinescu, Towards deductive verification of control algorithms for autonomous marine vehicles, in: Engineering of Complex Computer Systems (ICECCS), 25th Int. Conf., Singapore, 2020, pp. 113–118, arXiv:2006.09233.
[65] R. Calinescu, D. Weyns, S. Gerasimou, M.U. Iftikhar, I. Habli, T. Kelly, Engineering trustworthy self-adaptive software with dynamic assurance cases, IEEE Trans. Softw. Eng. 44 (2018) 1039–1069.
[66] R. Calinescu, M. Autili, J. Cámara, A. Di Marco, S. Gerasimou, P. Inverardi, A. Perucci, N. Jansen, J.-P. Katoen, M. Kwiatkowska, et al., Synthesis and verification of self-aware computing systems, in: Self-Aware Computing Systems, Springer, 2017, pp. 337–373.