Tuning permissiveness of active safety monitors for autonomous systems

Lola Masson[1], Jérémie Guiochet[12], Hélène Waeselynck[1], Kalou Cabrera[1], Sofia Cassel[3], and Martin Törngren[3]

[1] LAAS-CNRS, CNRS, Toulouse, France, `firstname.lastname@laas.fr`
[2] Université de Toulouse, UPS, Toulouse, France
[3] KTH, Stockholm, Sweden, `sofia.cassel@md.kth.se, martin@md.kth.se`

Abstract. Robots and autonomous systems have become a part of our everyday life, therefore guaranteeing their safety is crucial. Among the possible ways to do so, monitoring is widely used, but few methods exist to systematically generate safety rules to implement such monitors. Particularly, building safety monitors that do not constrain excessively the system's ability to perform its tasks is necessary as those systems operate with few human interventions. We propose in this paper a method to take into account the system's desired tasks in the specification of strategies for monitors and apply it to a case study. We show that we allow more strategies to be found and we facilitate the reasoning about the trade-off between safety and availability.

1 Introduction

Autonomous systems are becoming an increasing part of our daily lives: medical robots, self-driving cars, and industrial robots are good examples. It is critical to be able to guarantee the safety of such systems, since they operate independently in the vicinity of humans. One way to guarantee safety of autonomous systems is to attempt to specify them completely and reason about any dangerous behavior before deployment. This, however, requires that they behave predictably in any situation, which is not true as the systems interact with other humans or autonomous systems, in unstructured environments. In these cases, *monitoring* is an attractive option for guaranteeing safety. It consists in delegating the safety aspect to an independent component. It facilitates the task since a detailed specification of all possible behaviors is not necessary. A coarse granularity is sufficient to distinguish between safe and unsafe behavior. A *safety monitor* watches the system in operation, and intervenes as soon as potentially dangerous behavior is detected, typically by inhibiting certain actions, or by triggering corrective actions.

Such an approach, that only studies unsafe behaviors, may however restrict the system to the point where it cannot function in a meaningful way, i.e. availability suffers. For example, a robot having to transport objects might be kept standing still, or prohibited from dropping or picking up anything. The system might be safe, but useless for its purposes.

In this paper, we address the problem of monitoring autonomous systems while avoiding too conservative restrictions on behavior. This refers to the classical trade-off between safety and availability (readiness for usage). The system's availability depends on what we call the monitor's *permissiveness*, i.e. its ability to let the system reach functional states. We specify permissiveness by identifying the behaviors that are essential to the system's function. The specification is introduced as an extension to SMOF [15], a Safety Monitoring Framework we developed for the synthesis of safety strategies. It aims to facilitate the tuning of the monitor's permissiveness and the reasoning about how safety and availability interrelate in the context of a particular system.

The paper is organised as follows: in Section 2, we give an overview of related work; we detail the background of this work in Section 3; in Section 4, we present how we express functionality requirements for the synthesis of safety strategies. We apply this method to an example in Section 5. We conclude in Section 6.

2 Related work

Safety monitoring is a common mechanism used as part of fault-tolerance approaches [4] usually implemented as an independent mechanism that forces the system to stay in a safe state. This principle has been used in robotics under different names such as *safety manager* [18], *autonomous safety system* [19], *guardian agent* [7], or *emergency layer* [9].

In all these works, the specification of the safety strategies (i.e. the rules that the monitor follows to trigger safety interventions) is essentially done ad hoc: the user would manually choose what is the best (combination of) action(s) to trigger to avoid a risk, and when to do so. Other authors provide methods to identify safety invariants either from a hazard analysis [21] or from execution traces [11]. Some specify safety strategies in a DSL (Domain Specific Language) in order to generate code [3], [10]. But none of them offers a complete approach to identify invariants from hazards and formally derive the safety strategies. In contrast, our previous work [14], [15] provides a complete safety rule identification process, starting from a hazard analysis using the HAZOP-UML [8] technique and using formal verification techniques to synthesize the strategies.

Safety monitoring is related to runtime verification and property enforcement. Runtime verification [12][5] checks for properties (e.g., in temporal logic) by typically adding code into the controller software. Property enforcement [6] extends runtime verification with the ability to modify the execution of the controller, in order to ensure the property. These techniques consider a richer set of property classes than safety ones, and, most importantly, can be tightly coupled to the system. It makes the underlying mechanisms quite different from the external safety monitors considered in this paper which have to rely on limited observation and intervention means. Though, the *transparency* property mentioned for example in [13] and [16] for the specification of runtime monitors is close to our permissiveness property: the runtime monitor should not modify already correct behaviors.

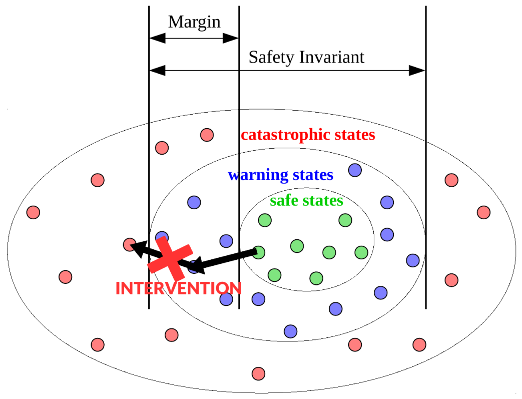

Fig. 1. System state space from the perspective of the monitor

3 Baseline and concepts

Our method to synthesize safety strategies for monitors is presented in [15]: SMOF (Safety Monitoring Framework). We briefly explain the SMOF principles below and introduce the motivation for the extension proposed in this paper.

3.1 Safety invariants, margins and states

As a first step of the process, one identifies a list of hazards that may occur during the system's operation, using the model-based hazard analysis HAZOP-UML [8]. From the list of hazards, one extracts those that can be treated by the monitor. Those hazards are reformulated as *safety invariants* such that each hazard is represented by the violation of an invariant. A safety invariant is a logic formula over a set of observable variables, derived from sensor values. The formalization of hazards as invariants may reveal the need for additional observation means, like in [17] where the original system design was revised to add sensors.

A combination of observation values defines a system state, as perceived by the monitor. If one of the safety invariants is violated, the system enters a *catastrophic state* that is assumed irreversible. Each safety invariant partitions the state space into catastrophic and non-catastrophic states as represented in Fig. 1. The non-catastrophic states can in turn be partitioned into *safe* and *warning states*, in such a way that any path from a safe state to a catastrophic one traverses a warning state. The warning states correspond to safety margins on the values of observations.

The monitor has means to prevent the evolution of the system towards the catastrophic states: these means are a set of safety *interventions* (mostly based on actuators) made available to it. An intervention is modeled by its effect (constraints that cut some transitions) and preconditions (constraint on the state in which it can be applied). Interventions are applied in warning states in order to cut all the existing transitions to the catastrophic states, as shown in

Fig. 1 by the red cross. The association of interventions to warning states constitutes a *safety strategy*. For example, let us assume that the invariant involves a predicate $v < V_{max}$ (the velocity should always be lower than V_{max}). In order to prevent evolution towards V_{max}, the strategy will associate one or several intervention(s) to warning states corresponding to a velocity higher than the threshold $V_{max} - margin$. The determination of the size of the margin involves a worst-case analysis, accounting for the dynamics of the physical system, as well as for the detection and reaction time of the monitor after the threshold crossing.

3.2 Safety and permissiveness properties

The safety strategy must fulfill two types of properties: *safety* and *permissiveness* properties. Both properties are expressed using CTL (Computation Tree Logic) which is well suited for reachability properties. *Safety* is defined as the non reachability of the catastrophic states. *Permissiveness* properties are intended to ensure that the strategy still permits functionality of the system, or, in other words maintains its availability. This is necessary to avoid safe strategies that would constrain the system's behavior to the point where it becomes useless (e.g., always engaging brakes to forbid any movement). SMOF adopts the view that the monitored system will be able to achieve its tasks if it can freely reach a wide range of states (e.g., it can reach states with a non zero velocity). Accordingly, permissiveness is generically formulated in terms of state reachability requirements: every non-catastrophic state must remain reachable from every other non-catastrophic state. We call it *universal permissiveness*. The safety strategy may cut some of the paths between pairs of states, but not all of the paths. In CTL, this is expressed as: $\mathsf{AG}(\mathsf{EF}(\mathsf{nc\_state}))$, for each non-catastrophic state. Indeed, EF specifies that the state of interest is reachable from the initial state, and AG extends this to the reachability from every state. The user can also use the *simple permissiveness* which merely requires the reachability from the initial state: $\mathsf{EF}(\mathsf{nc\_state}))$. It is much weaker than the universal permissiveness as it allows some of the monitor's interventions to be irreversible: after reaching a warning state in which the monitor intervenes, the system may be confined into a subset of states for the rest of the execution. For example, an emergency stop can permanently affect the ability of the system to reach states with a non zero velocity.

3.3 SMOF tooling

The SMOF tool support [2] includes the synthesis algorithm and a modelling template to ease the formalization of the different elements of the model: the behavior model with a partition into safe, warning and catastrophic states; the available interventions modeled by their effect on observable state variables; the safety and permissiveness properties. The template offers predefined modules, as well as auto-completion facilities. For example, the tool automatically identifies the set of warning states (having a transition to a catastrophic state). Also, the permissiveness properties are automatically generated based on the identification of non-catastrophic states. Finally, SMOF provides a synthesis tool based on the model-checker NuSMV [1]. For this reason the NuSMV language is used for the formalization and we will use the `typewriter` font to refer to it in the rest of the paper. The SMOF synthesis tool relies on a branch and bound algorithm that associates interventions to warning states and checks some criteria to evaluate if the branch should be cut or explored. It returns a set of both safe and permissive strategies for the given invariant to enforce (see [15] for details).

The formalization and strategy synthesis is done for each invariant separately. Then a last step is to merge the models and to check for the consistency of the strategies selected for the different invariants.

The SMOF method and tool have been applied to real examples of robots: an industrial co-worker in a manufacturing setting [15], and a maintenance robot in airfield [17]. Examples and tutorials can be found online [2].

3.4 On tuning permissiveness properties

This paper revisits the notion of permissiveness, in order to address some limitations of the generic definition adopted in SMOF. By default, SMOF requires the universal permissiveness, which is a very stringent requirement. As a result, the synthesis algorithm prunes any strategy that would cut all paths to a non-catastrophic state, even though this specific state may be useless for the accomplishment of the functions of the system. To give an example, let us consider a classical invariant stating that the system velocity should never reach a maximal absolute value V_{max}. The synthesis would reject any strategy preventing reachability of warning states with values close to V_{max}. But the cruise velocity of the system, used to accomplish its functions, is typically much lower than V_{max} and $V_{max} - margin$. Requiring the universal reachability of the warning states is useless in this case, since getting close to V_{max} is not a nominal behavior. The system operation could well accommodate a safety strategy that forbids evolution to close-to-catastrophic velocity values.

Suppose now that we do not find any solution respecting universal permissiveness. The user can choose to require the simple permissiveness. The requirements would be dramatically weakened. We would accept strategies that permanently affect the reachability of *any* non-catastrophic state, e.g., not only the close-to-catastrophic velocity values but also the moderate ones.

From what precedes, it may seem that we could simply modify the generic definition of permissiveness to require universal reachability of safe states only, excluding warning states. However, this would not work for all systems, as demonstrated by the maintenance robot studied in [17]. For this robot, some warning states *do* correspond to a nominal behavior and are essential to the accomplishment of the maintenance mission. More precisely, the robot is intended to control the intensity of lights along the airport runways. The light measurement task is done by moving very close to the lights, which, from the perspective of the anticollision invariant, corresponds to a warning state. Any safety strategy removing reachability of a close-to-catastrophic distance to the lights would defeat the very purpose of the robot.

Actually, there is no *generic* definition of permissiveness that would provide the best trade-off with respect to the system functions. We would need to incorporate some *application-specific* information to tune the permissiveness requirements to the needs of the system. This paper proposes a way to introduce such custom permissiveness properties into SMOF, allowing more strategies to be found and facilitating the elicitation of trade-offs in cases where some functionalities must be restricted due to safety concerns.

4 Defining custom permissiveness properties

The custom permissiveness properties are introduced by a dedicated state model, focusing on the identification of the states that are essential to the system functionalities (Section 4.1). The essential/not essential view is different from the safe/warning/catastrophic one we have in the safety state model. We thus need to bind together the functionality and safety models to reflect the fact that they represent two orthogonal decompositions of the same system state space (Section 4.2). Once this is done, custom permissiveness properties can be used as a replacement for the generic ones: the synthesis tool will search for safe strategies ensuring universal permissiveness with respect a subset of non-catastrophic states, the ones identified as essential. In case ignoring the non-essential states does not suffice to allow a strategy to be found, the user may consider whether restricting one of the functionalities is feasible (Section 4.3). The whole approach does not essentially change the principles of SMOF, which should facilitate its integration into the existing toolchain (Section 4.4).

4.1 A formal model for the permissiveness

We consider that a functionality is defined by a goal or objective that the system was designed to achieve. For example, if the system is designed to pick up objects and transport them, two of its functionalities could be "move from A to B" and "pick up an object". Some of the functionalities are not related to any of the monitored variables, and therefore do not need to be considered.

To be used in the synthesis, we must model the permissiveness properties associated to the identified functionalities. While generic permissiveness properties apply to all non-catastrophic states, we choose to express the custom ones as the reachability of a subset of states, the ones that are essential to the system's functionalities.

The state model for the functionalities is defined as a set of variables partitioned in classes of values of interest. For instance, let us consider the functionality `f`, which requires observable variable `v` (e.g., velocity, or position of a tool) to reach a given value $\mathtt{V_{req}}$ (e.g., cruise velocity, vertical position) with some tolerance δ. The domain of the variable `v` would be partitioned into three classes:

$\texttt{0}$ corresponding to values lower than $\texttt{V}_{\texttt{req}} - \delta$; $\texttt{1}$ to values in $[\texttt{V}_{\texttt{req}} - \delta, \texttt{V}_{\texttt{req}} + \delta]$; $\texttt{2}$ to values greater than $\texttt{V}_{\texttt{req}} + \delta$.

Let $\texttt{v}_{\texttt{f}}\ :\ \{\texttt{0}, \texttt{1}, \texttt{2}\}$ be the abstract variable encoding the partition from the point of view of the functionality. The SMOF template provides a predefined module to express the continuity constraints on the evolution of the variables values. For example, the velocity cannot jump from `0` to `2` without traversing the nominal range of values. Generally speaking, the modeling can reuse the syntactic facilities offered for the safety model, also defined in terms of observable variables, classes of values of interest and evolution constraints.

The purpose of the functionality model is to allow the user to conveniently specify sets of states that are essential to a given functionality. A set of states is introduced by a predicate `req` over a variable or a set of variables. In the above example, the user would specify that the functionality requires $\texttt{v}_{\texttt{f}} = \texttt{1}$. Each functionality may introduce several requirements, i.e., several essential sets of states. For instance, a "move" functionality could have two separate requirements, one for cruise motion and one for slow motion.

The list of `req` predicates can then be extracted to produce permissiveness properties of the form: $\mathsf{AG}(\mathsf{EF}(\mathsf{req}))$. We choose to reuse the same template as the one for universal permissiveness. However, we now require the reachability of sets of states, rather than the reachability of every individual state. For example, we have no reachability requirement on states satisfying $\texttt{v}_{\texttt{f}} = \texttt{2}$, and may accommodate strategies discarding some of the states $\texttt{v}_{\texttt{f}} = \texttt{1}$ provided that at least one of them remains reachable.

The functionalities that would require another type of template are not considered yet, but so far the expressivity of this template has been sufficient to model the considered functionalities.

4.2 Binding together invariants and permissiveness

The permissiveness and the safety properties are defined using two different state models. Some of the abstract variables used in those state models represent the same physical observation, or dependent ones. To connect the invariants and functionalities models, we have to bind their variables. Two types of bindings can be used: physical dependencies (e.g., speed and acceleration), or the use of the same observation with two different partitions.

In the first case, we specify the constraints on transitions (using the NuSMV keyword `TRANS`) or on states (`INVAR`). For example, for observations of speed and accelerations, we would write $\texttt{TRANS}\ \texttt{next(acc)} = \texttt{0} \rightarrow \texttt{next(speed)} = \texttt{speed}$. The NuSMV primitives `next(acc)` or `next(speed)` specify the value of `acc` or `speed` after transitioning, i.e., if the acceleration is null, the speed cannot change.

In the second case, we need to introduce a "glue" variable to bind the different partitions. This variable will be partitioned in as many intervals as needed. The different intervals will be bound with a specification on the states. For example, let us assume we have an invariant and a functionality using a velocity variable, and the partition used for the invariant is $\mathsf{v}_{\mathsf{inv}} = \{\texttt{0}, \texttt{1}, \texttt{2}\}$ where

0 : $\mathsf{stationary\ or\ slow}$, 1 : medium and 2 : high, and the one used for the functionality is $\mathsf{v_f} = \{0, 1\}$ where 0 : $\mathsf{stationary\ or\ slow}$ and 1 : $\mathsf{medium\ or\ high}$. We introduce a continuous "glue" variable partitioned as $\mathsf{v_{glue}} = \{0, 1, 2\}$. The binding through the "glue" variable is specified as follows:
INVAR $\mathtt{v_{glue}} = 0 \leftrightarrow \mathtt{v_{inv}} = 0$ & $\mathtt{v_f} = 0$;
INVAR $\mathtt{v_{glue}} = 1 \leftrightarrow \mathtt{v_{inv}} = 1$ & $\mathtt{v_f} = 1$;
INVAR $\mathtt{v_{glue}} = 2 \leftrightarrow \mathtt{v_{inv}} = 2$ & $\mathtt{v_f} = 1$.

Note that those two binding approaches, by adding constraints or glue variables, are also used in the standard SMOF process when merging models of different safety invariants.

4.3 Restricting functionalities

Custom permissiveness is weaker than SMOF's generic permissiveness, since we get rid of non essential reachability requirements. As a result, the strategy synthesis tool may return relevant strategies that would have been discarded with the generic version.

Still, it is possible that the custom requirements do not suffice, and that no strategy is found by the tool. We want to make it possible that the user restricts the functionalities, i.e., further weakens permissiveness. This may change the system's objectives, or increase the time or effort it takes to achieve the objectives. Functionalities can be restricted in several ways. We consider three of them in this paper. An important point is that the corresponding trade-offs are made explicit in the model of functionalities.

First, one of the req predicates can be weakened to correspond to a larger set of system states. The permissiveness property becomes $\mathsf{AG}(\mathsf{EF}(\mathsf{req}'))$, with $\mathsf{req} => \mathsf{req}'$. For example, the "move" functionality can be restricted to "move slowly", where req' means that $\mathtt{V}$ only needs to reach a velocity $\mathtt{V_{req'}}$ lower than the initially specified cruise one.

Second, it is possible to replace the universal permissiveness property by the simple one, $\mathsf{EF}(\mathsf{req})$. This weak form of permissiveness was already offered by the generic version of SMOF, but it applied to all individual states. With the custom list of req predicates, the user can decide for which of these predicates simple permissiveness would be acceptable. For example, the "move" functionality could become impossible after some monitor's intervention has been triggered, but other functionalities like manipulating objects would have to be preserved.

The third way is to simply remove the functionality from the requirements. For example a "manipulation while moving" functionality is no longer required. Here, the corresponding CTL property is simply deleted, and the synthesis run again without it. This ultimate restriction step can show the user that the intended functionality is not compatible with the required level of safety. This information can be propagated back to the hazard analysis step and used to revise the design of the system or its operation rules. Again, not all of the requirements may need a restriction. The functionalities that are not restricted are guaranteed to remain fully available despite the monitor's intervention.

4.4 Integration in SMOF tooling

Integrating management of custom permissiveness into SMOF is possible without any major change of the core toolset. Only the front-end would need to be updated. The template would include an optional functionality modeling part with syntactic sugar to introduce the list of `req` predicates. The auto-completion facilities would allow for the generation of the CTL properties from the `req` predicates and the desired level of permissiveness (universal, simple) for each of them. The core modeling approach and strategy synthesis algorithm would remain unchanged.

The SMOF front-end has not been updated yet. Still, we can use the current version of SMOF for the experimentation, as presented in the next Section. We manually entered the CTL properties (rather than just the `req` predicates) and intercepted a generated command script to instruct the synthesis tool to use the custom properties, not the generic ones.

5 Application to an example

To explain the approach and study its usefulness we will consider a robotic system composed of a mobile platform and an articulated arm (see Fig. 2). It is an industrial co-worker in a manufacturing setting, sharing its workspace with human workers. Its purpose is to pick up objects using its arm and to transport them. To do so, the arm can rotate and a gripper at its end can open and close to hold objects. Some areas of the workspace are prohibited to the robot. The hazard analysis has been performed on this robot in a previous project [20] and a list of 13 safety invariants has been identified. Among them, 7 could be handled by the monitor [15]. Several ways to model those invariants have been explored, considering various interventions and observation means. We defined custom permissiveness properties for each of the models and detail here the three invariants that give different results compared to generic permissiveness:

· SI_1: the arm must not be extended when the platform moves with a speed higher than s_{max};
· SI_2: a gripped box must not be tilted more than α_0;
· SI_3: the robot must not enter a prohibited zone.

The models for the three mentioned invariants can be found online at [2]. For all of them, the synthesis took less than 0.5 seconds to compute on an Intel Core i5-3437U CPU @ 1.90GHz x 4 with 16 GB of memory.

5.1 SI_1: the arm must not be extended when the platform moves over a certain speed

Modeling We consider the invariant SI_1*: the arm must not be extended when the platform moves with a speed higher than* s_{max}. The available observations are s_{inv}, the speed of the platform; and a_{inv}, the position of the arm. Note that the variables names are extended with the index `inv` to specify that they are

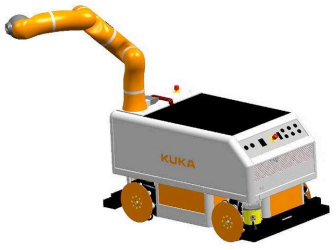
Fig. 2. Manipulator robot from Kuka

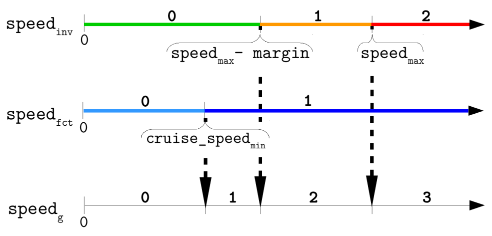

Fig. 3. Partitioning of the s_g variable

Speed of the platform	**Real speed interval**	**Discrete variable**
Low	$s_{inv} < s_{max} - m$	$\mathtt{s_{inv}} = 0$
Within the margin	$s_{max} - m \leq s_{inv} < s_{max}$	$\mathtt{s_{inv}} = 1$
Higher than the maximum allowed value	$s_{inv} \geq s_{max}$	$\mathtt{s_{inv}} = 2$

Position of the arm	**Discrete variable**
Not extended beyond the platform	$\mathtt{a_{inv}} = 0$
Extended beyond the platform	$\mathtt{a_{inv}} = 1$

Table 1. Partitioning of the variables $\mathtt{s_{inv}}$ and $\mathtt{a_{inv}}$

Speed of the platform	**Real speed interval**	**Discrete variable**
Null or lower than the minimum cruise speed	$s_{fct} < c\_s_{min}$	$\mathtt{s_{fct}} = 0$
At the minimum cruise speed or higher	$s_{fct} \geq c\_s_{min}$	$\mathtt{s_{fct}} = 1$

Position of the arm	**Discrete variable**
Folded	$\mathtt{a_{fct}} = 0$
Extended beyond the platform	$\mathtt{a_{fct}} = 1$

Table 2. Partitioning of the variables $\mathtt{s_{fct}}$ and $\mathtt{a_{fct}}$

the variables used for the invariant model. The observations are partitioned as detailed in Tab. 1. Considering the discrete representation of the variables, the catastrophic state can be expressed as $\mathtt{cata}:\ \mathtt{s_{inv}} = 2\ \&\ \mathtt{a_{inv}} = 1$ (high speed with extended arm).

To express the relevant permissiveness properties, we identify in the specifications what functionalities are related to the invariant. Let us consider the variables involved in SI_1. The $\mathtt{s_{inv}}$ variable is an observation of the speed of the mobile platform, in absolute value. The system is supposed to move around the workplace to carry objects, i.e., the speed must be allowed to reach a minimal cruise speed value $\mathtt{c\_s_{min}}$, from any state. To model this functionality we introduce the $\mathtt{s_{fct}}$ variable, which will be partitioned as showed in Tab. 2. Note that the variables names are extended with the index $\mathtt{fct}$ to specify that they are the variables used for the functionalities model. This property can be expressed following the template: $\mathsf{cruise\ motion}\ :\ \mathsf{AG}(\mathsf{EF}(\mathsf{s_{fct}} = 1))$. The system must also be able to stop or move slowly, thus another functionality is expressed: $\mathsf{slow\ motion}\ :\ \mathsf{AG}(\mathsf{EF}(\mathsf{s_{fct}} = 0))$. Also, the $\mathtt{a_{inv}}$ variable models whether the manipulator arm is extended beyond the platform or not. To handle objects, the arm must be allowed from any state to reach a state where the arm is extended beyond the platform, and a state where the arm is folded. We introduce the variable $\mathtt{a_{fct}}$ which is partitioned as showed in Tab. 2. We have arm extension : $\mathsf{AG}(\mathsf{EF}(\mathsf{a_{fct}} = 1))$ and arm folding : $\mathsf{AG}(\mathsf{EF}(\mathsf{a_{fct}} = 0))$.

The speed value and arm position are observed in both the invariant model ($\mathtt{s_{inv}}$ and $\mathtt{a_{inv}}$) and the functionalities model ($\mathtt{s_{fct}}$ and $\mathtt{a_{fct}}$). We need to make their evolution consistent. To do so, we introduce glue variables, $\mathtt{s_g}$ and $\mathtt{a_g}$.

For the speed, we have two different partitions as presented in Fig. 3, one for $\mathtt{s_{inv}}$ (with discrete values $\{0, 1, 2\}$) and one for $\mathtt{s_{fct}}$ (with discrete values $\{0, 1\}$). The resulting glue variable $\mathtt{s_g}$ will then have four values as presented in Fig. 3. We thus have the formal definition:

INVAR $\mathtt{s_g} = 0 \leftrightarrow \mathtt{s_{inv}} = 0 \;\&\; \mathtt{s_{fct}} = 0$;
INVAR $\mathtt{s_g} = 1 \leftrightarrow \mathtt{s_{inv}} = 0 \;\&\; \mathtt{s_{fct}} = 1$;
INVAR $\mathtt{s_g} = 2 \leftrightarrow \mathtt{s_{inv}} = 1 \;\&\; \mathtt{s_{fct}} = 1$;
INVAR $\mathtt{s_g} = 3 \leftrightarrow \mathtt{s_{inv}} = 2 \;\&\; \mathtt{s_{fct}} = 1$.

For the arm variable, it is much simpler:

INVAR $\mathtt{a_g} = 0 \leftrightarrow \mathtt{a_{inv}} = 0 \;\&\; \mathtt{a_{fct}} = 0$;
INVAR $\mathtt{a_g} = 1 \leftrightarrow \mathtt{a_{inv}} = 1 \;\&\; \mathtt{a_{fct}} = 1$.

Additionally, we have to provide a model of the interventions of the monitor. Let us consider that two interventions are available: the brakes can be triggered and affect the speed, and the extension of the arm can be blocked. Concerning the braking intervention, it can be applied at any time but will only be efficient (i.e. prevent the reachability of $\mathtt{s_{inv}} > \mathtt{s_{max}}$) if it is engaged when the speed threshold $\mathtt{s_{max}} - \mathtt{m}$ has just been crossed. Indeed, the size of the margin is chosen precisely to have time to brake before reaching the undesired value. For the intervention blocking the arm, its effect is to block the extension and it can only be applied if the arm is not already extended (see definitions in Tab. 3).

Name	**Precondition**	**Effect**
Brake	$\mathtt{s_{inv}} = 0$ & $\mathtt{next(s_{inv})} = 1$	$\mathtt{next(s_{inv})} = \mathtt{s_{inv}} - 1$
Block_arm	$\mathtt{a_{inv}} = 0$	$\mathtt{next(a_{inv})} = 0$

Table 3. Interventions definition for SI_1

Name	**Precondition**	**Effect**
Brake	$\mathtt{d} = 2$ & $\mathtt{next(d)} = 1$	$\mathtt{next(v_{inv})} = 0$

Table 4. Interventions definition for SI_2

Results We compare in this section the results obtained without and with the approach through the definition of custom permissiveness. To graphically describe the strategies, we represent the invariant as a state machine. In the first case, we use the generic permissiveness, i.e., the reachability of every non-catastrophic state (the states $\{\mathtt{safe_1}, \mathtt{safe_2}, \mathtt{w_1}, \mathtt{w_2}, \mathtt{w_3}\}$ in Fig. 4), from every other non-catastrophic state. Only one minimal strategy (no useless interventions) is both safe and permissive. It is found by SMOF and represented in Fig. 4.

In the second case, we replace the generic permissiveness with the use of the custom permissiveness properties cruise motion, slow motion, arm folding and arm extension specified before. We only require the reachability of the states $\{\mathtt{safe_1}, \mathtt{safe_2}\}$. After running the synthesis, in addition to the previous strategy we have a strategy only using the braking intervention (see Fig. 5). This can be

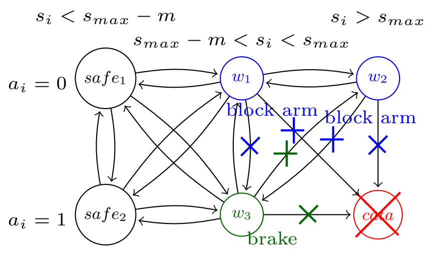

Fig. 4. Single strategy synthesized for the invariant SI_1 with generic permissiveness properties.

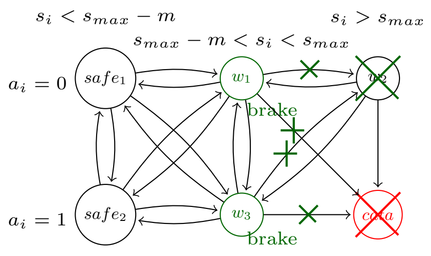

Fig. 5. Additional strategy synthesized for the invariant SI_1 with the custom permissiveness properties.

preferable in some cases, as the use of the arm is then never impacted and even if the monitor triggers the brakes the system can keep manipulating objects. This strategy couldn't be found with the generic permissiveness as it removes the reachability of `w_2`.

The custom permissiveness requirements may allow to synthesize more strategies, like in the previous example, or even to synthesize strategies for problems that had no solution with the generic permissiveness. Indeed, we require the reachability of a reduced set of states, therefore more strategies can be found.

5.2 SI_2: a gripped box must not be tilted more than α_0

For the initial model presented in [15], the monitor could observe the presence of a box (inferred through the position of the robot's arm in the workspace and the position of the gripper), and the angle of rotation of the arm. No strategy was found. Indeed, the monitor only could brake the arm (prevent its rotation) and no control was possible on the gripper. The monitor would thus not be able to prevent the system from grabbing a box with the gripper already tilted more than α_0. We chose to reformulate the invariant as SI_2' *: a gripped box must not be tilted mode than* α_0 *if the robot is outside of the storage area*. The industrial partner indicated that dropping a box (tilt it over α_0) in the storage area is not that dangerous as it would fall from a low height.

As an alternative solution to ensure SI_2, we also explored the effect of an additional intervention: the monitor can lock the gripper (prevent it from closing). But the automated synthesis with the generic permissiveness failed to return any strategy. We now revisit this model by specifying custom permissiveness properties for a manipulation functionality (carrying a box at a low rotation angle). Using the custom permissiveness, the tool successfully synthesizes a strategy. It combines the braking of the arm and the lock of the gripper to maintain the invariant while permitting functionality.

5.3 SI_3: the robot must not enter a prohibited zone

Modeling The considered invariant is: *SI_3: the robot must not enter a prohibited zone.* The observation used is d, the distance to the prohibited zone. The distance variable is partitioned according to the concept of margin: $\mathtt{d}: \{0, 1, 2\}$, 0 representing the robot into the prohibited zone, 1 the robot close to the prohibited zone and 2 the robot far from the prohibited zone. According to this partition, the catastrophic state can be expressed as $\mathtt{cata}: \mathtt{d} = 0$.

The only available intervention here is the braking intervention, which stops the robot completely. To model this intervention, we introduce a velocity variable $\mathtt{v_{inv}}$, partitioned as follows: $\mathtt{v_{inv}}: \{0, 1\}$ where 0 represents the robot stopped and 1 the robot moving. A dependency between the distance and the velocity variables is specified as $\mathtt{TRANS\ next(v_{inv})} = 0 \rightarrow \mathtt{next(d)} = \mathtt{d}$, i.e., the distance cannot change if the robot does not move. The braking intervention is only effective under the precondition that the distance threshold to the prohibited zone has just been crossed, and affects the velocity variable. This intervention is modeled as shown in Tab. 4.

In this case, for the functionalities we just need to specify that the robot needs to reach a state where it is moving, and a state where it is stopped. We model the custom permissiveness with $\mathsf{move} : \mathsf{AG}(\mathsf{EF}(\mathsf{v_{fct}} = 1))$ and $\mathsf{stop} : \mathsf{AG}(\mathsf{EF}(\mathsf{v_{fct}} = 0))$ where $\mathtt{v_{fct}}$ represents the robot moving or stopped. This variable is directly bound to the $\mathtt{v_{inv}}$ variable with a glue variable $\mathtt{v_g}$ as:
$\mathtt{INVAR}\ \mathtt{v_g} = 0 \leftrightarrow \mathtt{v_{inv}} = 0\ \&\ \mathtt{v_{fct}} = 0;$
$\mathtt{INVAR}\ \mathtt{v_g} = 1 \leftrightarrow \mathtt{v_{inv}} = 1\ \&\ \mathtt{v_{fct}} = 1.$

Results The synthesis of strategies with the braking intervention and the move and stop functionalities does not give any result. Applying the brakes until the system is stopped violates the permissiveness property associated to the move functionality. The system is stopped close to the prohibited zone and cannot ever move again, i.e., the monitor's intervention is irreversible. In term of automata, we reached a deadlock state. In order to guarantee safety and synthesize a strategy, we need to either change the interventions or accept a restriction of the functionality as described in Section 4.3. In our case, we do not have any other intervention, we thus need to restrict the functionality.

We can express the restriction as follows: we accept the intervention of the monitor to be irreversible, but we still want the functionality to be fully available before the intervention of the monitor. We have the following resulting permissiveness property: $\mathsf{restricted\ move} : \mathsf{EF}(\mathsf{v_{fct}} = 1)$. The property for stop remains unchanged. With the restricted permissiveness property, the synthesis generates one strategy which is modeled in Fig. 6.

As we can see, the w state is a deadlock: the system cannot reach any other state from this state. It means that if the robot ever gets too close to a prohibited zone, it will be stopped by the monitor and an intervention of the operator will be needed to continue the mission. The safety is guaranteed by this strategy. Specifying the restriction of functionalities highlights the impact of the monitor

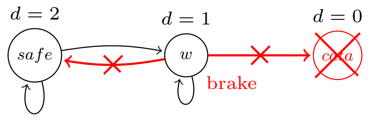

Fig. 6. Strategy synthesized for the invariant SI_3 with restricted functionality.

intervention on the system ability to function, which was not possible with the use of generic simple permissiveness.

6 Conclusion and perspectives

In this paper, we have described an approach to specify safety monitors for robots, using and extending the SMOF monitoring framework. We overcome an overly stringent definition of the monitor's permissiveness in proposing a custom definition of permissiveness according to the system's functionalities (the behaviors necessary to fulfill its purposes). The custom permissiveness properties are expressed following a simple template. We require the reachability of a reduced set of states, therefore, more strategies can be synthesized. In the studied example, the proposed solution provided a new strategy only requiring the use of one intervention instead of two. Also, a problem which had no solutions with a generic definition of permissiveness properties had one with custom properties.

Whenever it is not possible to synthesize a safety strategy, we propose an iterative design strategy: we give three ways to adapt functionalities by weakening the permissiveness properties following a template. In these situations, some strategies can often still be found with slight and traceable changes of the functionalities. The impact of the monitor on the robot's operation can thus be qualified and reasoned about.

Integrating the definition and use of custom permissiveness properties is now possible with the existing SMOF tooling with a small change on the front-end. The synthesis algorithm remains unchanged. In future work we wish to adapt the template so that the user does not have to use the CTL logic.

We would also like to extend our approach to cover different types of monitor interventions. We could search for multi-level strategies combining guaranteed and non-guaranteed interventions (having a probability of success, possibly depending on the operational situation). The monitor would first try the interventions that would affect the system without compromising the mission (e.g., trajectory re-planing). In case of failure of those interventions, the least permissive but guaranteed ones (e.g., emergency stop) would only be triggered in last emergency.

References

1. NuSMV home page. http://nusmv.fbk.eu/. accessed November 2017.

2. Safety Monitoring Framework. LAAS-CNRS Project, https://www.laas.fr/projects/smof. accessed December 2017.
3. S. Adam, M. Larsen, K. Jensen, and U. P. Schultz. Rule-based Dynamic Safety Monitoring for Mobile Robots. *Journal Of Software Engineering In Robotics*, 2016.
4. A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 2004.
5. N. Delgado, A. Q. Gates, and S. Roach. A taxonomy and catalog of runtime software-fault monitoring tools. *Transactions on Software Engineering*, 2004.
6. Y. Falcone, J.-C. Fernandez, and L. Mounier. What can you verify and enforce at runtime? *International Journal on Software Tools for Technology Transfer*, 2012.
7. J. Fox and S. Das. *Safe and sound - Artificial Intelligence in Hazardous Applications*. 2000.
8. J. Guiochet. Hazard analysis of human-robot interactions with HAZOP-UML. *Safety Science*, 84, 2016.
9. S. Haddadin, M. Suppa, S. Fuchs, T. Bodenmüller, A. Albu-Schäffer, and G. Hirzinger. Towards the robotic co-worker. In *The 14th International Symposium on Robotics Research (ISRR2011)*, 2011.
10. J. Huang, C. Erdogan, Y. Zhang, B. Moore, Q. Luo, A. Sundaresan, and G. Rosu. ROSRV: Runtime Verification for Robots. In *Runtime Verification*. Sept. 2014.
11. H. Jiang, S. Elbaum, and C. Detweiler. Inferring and monitoring invariants in robotic systems. *Autonomous Robot*, 2017.
12. M. Leucker and C. Schallhart. A brief account of runtime verification. *Journal of Logic and Algebraic Programming*, 2009.
13. J. Ligatti, L. Bauer, and D. Walker. Edit automata: enforcement mechanisms for run-time security policies. *IJIS*, 2005.
14. M. Machin, F. Dufossé, J.-P. Blanquart, and J. Guiochet. Specifying Safety Monitors for Autonomous Systems Using Model-Checking. In *Computer Safety, Reliability, and Security, SAFECOMP2014*.
15. M. Machin, G. Jérémie, W. Hélène, J.-P. Blanquart, M. Roy, and L. Masson. Smof - a safety monitoring framework for autonomous systems. *IEEE Transactions On System, Man and Cybernetics: Systems*, 2016.
16. F. Martinelli, I. Matteucci, and C. Morisset. From Qualitative to Quantitative Enforcement of Security Policy. In *Mathematical Methods, Models and Architecture for Computer Network Security*, 2012.
17. L. Masson, J. Guiochet, H. Waeselynck, A. Desfosses, and M. Laval. Synthesis of Safety Rules for Active Monitoring: Application to an Airport Light Measurement Robot. In *2017 First IEEE International Conference on Robotic Computing (IRC)*, 2017.
18. C. Pace and D. Seward. A safety integrated architecture for an autonomous safety excavator. In *International Symposium on Automation and Robotics in Construction*, 2000.
19. S. Roderick, B. Roberts, E. Atkins, and D. Akin. The ranger robotic satellite servicer and its autonomous software-based safety system. *Intelligent Systems*, 2004.
20. SAPHARI. Safe and Autonomous Physical Human-Aware Robot Interaction. Project supported by the European Commission under the 7th Framework Programme, www.saphari.eu, accessed Nov. 2017, 2011-2015.
21. R. Woodman, A. F. Winfield, C. Harper, and M. Fraser. Building aafer robots: Safety driven control. *International Journal of Robotics Research*, 2012.