Tuning permissiveness for active safety monitoring

Lola Masson, Jérémie Guiochet, Hélène Waeselynck, Kalou Cabrera, Sofia Cassel, Martin Törngren

Tuning permissiveness for active safety monitoring

Lola Masson, Jérémie Guiochet, Hélène Waeselynck, Kalou Cabrera
LAAS-CNRS
Toulouse, France
firstname.lastname@laas.fr

Sofia Cassel, Martin Törngren
KTH
Stockholm, Sweden
sofia.cassel@md.kth.se
martin@md.kth.se

ABSTRACT

Robots and autonomous system have become a part of our everyday life, therefore guaranteeing their safety is a crucial issue. Among the possible methods for guaranteeing safety, monitoring is widely used, but few methods exist to 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 can synthesize a more important number of strategies and we facilitate the reasoning about the trade-off between safety and functionalities.

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 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 facilitates the task since a specification of all possible behaviors is not necessary; it suffices to focus on unsafe behaviors. A *safety monitor* watches the system in operation, and intervenes as soon as potentially dangerous behavior is detected, typically by blocking the system from performing certain 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 tasked with moving objects might be restricted to only standing still, or to moving without being able to drop or pick up anything. The system might be safe, but useless for its purposes.

In this paper, we address the problem of monitoring autonomous systems aiming to avoid too conservative restrictions on behaviors. This is the classical trade-off between safety and availability. For the availability requirements, we explicitly consider the system's purpose, which we call its *functionality*. We specify the sequences of behavior that are essential to the system's function. We then synthesize safety strategies for the monitor to follow, such that when it is triggered, it does not interfere with these sequences. If it is not possible to generate such strategies, we can iteratively limit the system's functionality, e.g., by reducing its speed or the range of some moving part and keep track of this limitation. Our approach thus makes it possible to generate safety monitors that are less restrictive, and it also enables us to reason explicitly about how safety and functionality (i.e. availability) interrelate in the context of a particular system.

The paper is organised as follows: in Section 2, we will discuss related work; we detail the background of this work in Section 3; the main contribution of the paper will be presented in Section 4 and applied to an example in Section 5. We will conclude in Section 6.

2. RELATED WORK

Safety monitoring is a popular form of fault tolerance [5] usually implemented as an independent mechanism that forces the system to stay in a safe state. Several approaches have been used in robotics such as *safety manager* [18], *autonomous safety system* [20], *checker* [19], *guardian agent* [8], or *emergency layer* [10].

In all these works, the specification of the safety strategies (i.e. the rules that the monitor follows to trigger safety interventions) is done ad hoc not using any generic approach. Other authors provide methods to identify safety invariants either from a hazard analysis [22] or from existing implementations [12], or to specify safety strategies in a DSL (Domain Specific Language) in order to generate code [4], [11]. But none of them offers a complete approach to identify invariants from hazards and automatically synthesize safety strategies to implement for monitoring. In contrast, the approach detailed in [16] and [14] provides a complete safety rule identification process, starting from a hazard analysis using the HAZOP-UML [9] technique and using formal verification techniques to synthesize the strategies.

Safety monitoring is related to runtime verification and property enforcement. Runtime verification [13][6] checks for properties (e.g., in temporal logic) by typically adding code into the controller software. Property enforcement [7] extends runtime verification with the ability to modify the

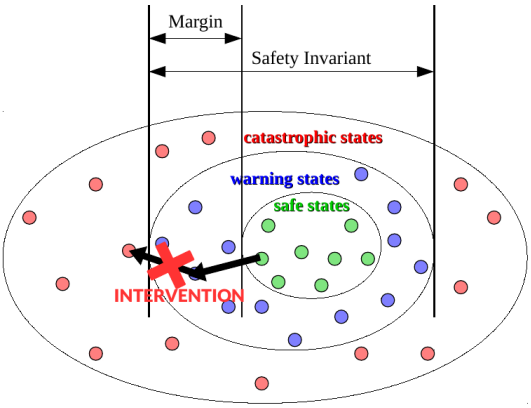

Figure 1: System state space from the perspective of the monitor

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.

3. BASELINE AND CONCEPTS

A method to synthesize safety strategies for monitors is proposed in [16]: 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 [9]. Among 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. Note that the reformulation 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 Figure 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 (constraint 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 Figure 1. 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}). The strategy will associate an intervention to states corresponding to a velocity higher than the threshold $V_{max} - margin$ in order to prevent evolution towards V_{max} . 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 fulfil two types of properties: *safety* and *permissiveness* 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 maintain 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 evolve towards 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 any other non-catastrophic state. The safety strategy may remove some of the paths between pairs of states, but not all of the paths.

3.3 SMOF tooling

SMOF tool support [2] includes 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 modelled by their effect on observable state variables; the safety and permissiveness properties, both expressed using CTL (Computation Tree Logic), which allows the expression of reachability properties. The template offers predefined modules, as well as auto-completion facilities. For example, the tool automatically identifies the set of warning states. 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 following. The synthesis tool returns a set of adequate strategies for the given invariant to enforce, where adequate means both safe and permissive.

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 [16], and a maintenance robot in airfield [17].

3.4 Contribution of this paper

This paper revisits the notion of permissiveness, in order to address some limitations of the generic definition adopted in SMOF. By default, it requires the universal reachability of all non-catastrophic states, 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.

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 do so without essentially changing the principles and tools of the SMOF method. The custom permissiveness properties are introduced as an alternative to the generic ones, allowing more strategies to be found and facilitating the iterative trade-off decision process.

4. DEFINING CUSTOM PERMISSIVENESS PROPERTIES

4.1 Process overview

To synthesize a safety strategy using the SMOF tool, we follow the process described in Figure 2. The SMOF algorithm takes as inputs the permissiveness properties, the safety invariant and intervention models. In the shown process, we assume that the user has not chosen to generate generic properties (reachability of all the non-catastrophic states) but has specified custom ones. If no strategy is found, the user must consider whether restricting one of the functionalities is feasible. If this is the case, she may adapt the permissiveness properties consequently, by weakening some reachability requirements. The synthesis algorithm is then launched again to find a strategy, and repeated if another restriction of functionalities is needed. We can therefore precisely qualify the impact of safety on the system's ability to function in a useful manner.

We first present the definition and modeling of permissiveness (Section 4.2), then the binding of permissiveness and safety models (Section 4.3), then the restriction of functionalities (Section 4.4) and finally the integration of this approach in the existing tool chain (Section 4.5).

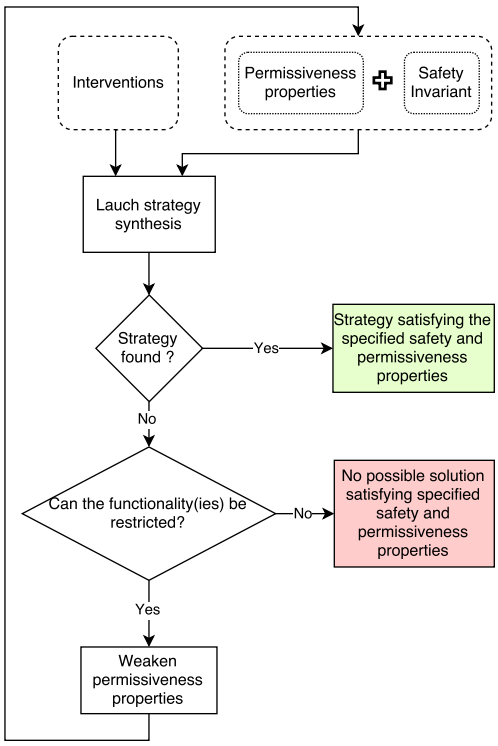

Figure 2: Overview of the strategy synthesis process

4.2 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, transport and place them, two of its functionalities could be "move from A to B" and "pick up an object". We propose to adapt the permissiveness to the functionalities of the system and therefore only require the reachability of a specific subset of states for the strategy synthesis. Some of the functionalities are not related to any of the monitored variables, and therefore do not need to be considered. For example the safety monitor does not affect a display functionality which is not associated to any hazard.

To be used in the synthesis, we must model the permissive-

ness 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 $\mathtt{f}$, which requires observable variable $\mathtt{v}$ (e.g. velocity, or position of a tool) to reach a given value $\mathtt{V_{req}}$ (e.g. cruise velocity) with some tolerance δ. The domain of the variable $\mathtt{v}$ would be partitioned into three classes:

· $\mathtt{0}$ corresponding to values lower than $\mathtt{V_{req}} - \delta$;
· $\mathtt{1}$ to values in $[\mathtt{V_{req}} - \delta, \mathtt{V_{req}} + \delta]$;
· $\mathtt{2}$ to values greater than $\mathtt{V_{req}} + \delta$.

Let $\mathtt{v_f} : \{0, 1, 2\}$ be the abstract variable encoding the partition from the point of vie of the functionality. The SMOF template provides a predefined module to express the continuity constraints on the evolution of the variables values.

The corresponding permissiveness property is the following: $\mathsf{AG}(\mathsf{EF}(\mathsf{v_f} = 1)))$. Indeed, EF specifies that the value of interest is reachable from the initial state, and AG extends this to the reachability from every state. This means that the monitor's interventions must not permanently affect the functionality $\mathtt{f}$ by cutting all paths to useful states.

Every permissiveness property must be expressed following this template: $\mathsf{AG}(\mathsf{EF}(\mathsf{req})))$ where req is a predicate over a variable or set of variables. To each functionality corresponds a (set of) permissiveness property(ies) in CTL. For example, if two movement functionalities can be identified, cruise mode and slow mode, then the reachability of the corresponding range of values must be specified in two separate properties.

4.3 Binding 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 (speed and acceleration for example), or the use of the same observation with two different partitions.

In the first case, we specify the constraints on transitions (using the NuSMV key word `TRANS`) or on states (using `INVAR`). For example, for observations of speed and accelerations, we would write $\mathtt{TRANS}\ \mathtt{next(acc)} = 0 \rightarrow \mathtt{next(speed)} = \mathtt{speed}$, $\mathtt{next(acc)}$ or $\mathtt{next(speed)}$ specifying the value of $\mathtt{acc}$ or $\mathtt{speed}$ 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, and specified as continuous. 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 $\mathsf{v_{inv}} = \{0, 1, 2\}$ where 0 : stationary or slow, 1 : medium and 2 : high, and the one used for the functionality is $\mathsf{v_f} = \{0, 1\}$ where 0 : stationary or slow and 1 : 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:
$\mathtt{INVAR}\ \mathtt{v_{glue}} = 0 \leftrightarrow \mathtt{v_{inv}} = 0\ \&\ \mathtt{v_f} = 0;$
$\mathtt{INVAR}\ \mathtt{v_{glue}} = 1 \leftrightarrow \mathtt{v_{inv}} = 1\ \&\ \mathtt{v_f} = 1;$
$\mathtt{INVAR}\ \mathtt{v_{glue}} = 2 \leftrightarrow \mathtt{v_{inv}} = 2\ \&\ \mathtt{v_f} = 1.$

4.4 Restricting functionalities

In some cases, no strategy can be synthesized with the specified functionalities. One solution is that the user restricts the functionalities, i.e., 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.

The required interval of values can be changed. For example, if the system's objective is to move from point A to point B, then the "move" functionality can be restricted to "move slowly". The system will still achieve its objective, but not as quickly. For example, let us consider a "move" functionality expressed as $\mathsf{F} : \mathsf{AG}(\mathsf{EF}(\mathsf{req}))$, where req means that the velocity v must reach values greater than V_1. Restricting this functionality is expressed as $\mathsf{F}' : \mathsf{AG}(\mathsf{EF}(\mathsf{req}'))$, where req' means that v only has to reach V'_1 with $\mathsf{V}'_1 < \mathsf{V}_1$.

Another way is to allow the monitor's intervention to be irreversible. The "move" functionality could then become impossible after some monitor's intervention has been triggered. In that case, we at least require the functionality to be reachable from the initial state. This is expressed as $\mathsf{EF}(\mathsf{v_{f1}} = \mathsf{Vreq})$. The restriction just consists in this case in removing the AG part from the formula.

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.

Functionalities that cannot be restricted and for which no safety strategy can be found can also be propagated back to the hazard analysis step and used to revise the design of the system or its operation rules.

4.5 Integration in SMOF tooling

So far, the management of custom permissiveness has not yet been integrated in the tool. Examples and tutorials can be found online [3]. However, conceptually it would only require minimal changes.

The template has to be extended to facilitate the modeling of the permissiveness properties. Syntactic sugar can be provided so that the user does not directly work with CTL. She could specify the states of interest with a key word for the desired reachability (from the initial state only or from every non-catastrophic state). Also, the model auto-completion facilities have to be adapted to generate the CTL properties for the custom permissiveness. The other parts of the template (safety invariant and interventions models) can be reused unchanged.

The SMOF strategy synthesis algorithm, which is the core of the tool set, needs not to be changed either. It can transparently accommodate both types of permissiveness properties (custom or generic).

5. APPLICATION TO AN EXAMPLE

We will use as a running example a robotic system composed of a mobile platform and an articulated arm (see Figure 3). 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. Some areas of the workspace are prohibited to the robot. The hazard analysis has been performed on this robot in a previous project [21] and a list of thirteen safety invariants has been identified. The complete list can be found in [15]. Only two of them will be detailed in this paper:

SI_1 : the arm must not be extended when the platform moves over a certain speed;
SI_2 : the robot must not enter a prohibited zone.

Figure 3: Manipulator robot from Kuka

5.1 The invariant SI_1

5.1.1 Modeling

We consider the invariant SI_1: *the arm must not be extended when the platform is moving with a speed higher than* $\texttt{speed}_{\texttt{max}}$. The available observations are $\texttt{speed}_{\texttt{inv}}$, the speed of the platform; and $\texttt{arm}_{\texttt{inv}}$, the position of the arm. Note that the variables names are extended with the index `inv` to specify that they are the variables used for the invariant model. The observations are partitioned as detailed in Table 1. Considering the discrete representation of the variables, the catastrophic state can be expressed as $\texttt{cata} : \texttt{speed}_{\texttt{inv}} = 2 \ \& \ \texttt{arm}_{\texttt{inv}} = 1$ (high speed with extended arm).

To express the relevant permissiveness properties, we identify what functionalities are related to the invariant. Let us consider the variables involved in SI_1. The $\texttt{speed}_{\texttt{inv}}$ variable is an observation of the velocity 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 $\texttt{cruise\_speed}_{\texttt{min}}$ value, from any state. To model this functionality we introduce the $\texttt{speed}_{\texttt{fct}}$ variable, which will be partitioned as showed in Table 2. Note that the variables names are extended with the index `fct` to specify that they are the variables used for the functionalities model. This property can be expressed in CTL as $\mathsf{cruise\ motion} : \mathsf{AG}(\mathsf{EF}(\mathsf{speed}_{\mathsf{fct}} = 1))$. The system must also be able to stop or move slowly, thus another functionality is expressed: $\mathsf{slow\ motion} : \mathsf{AG}(\mathsf{EF}(\mathsf{speed}_{\mathsf{fct}} = 0))$. Also, the $\texttt{arm}_{\texttt{inv}}$ 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 arm_{fct} which is partitioned as showed in Table 2. In CTL, we have $\mathsf{arm\ extension} : \mathsf{AG}(\mathsf{EF}(\mathsf{arm}_{\mathsf{fct}} = 1))$ and $\mathsf{arm\ folding} : \mathsf{AG}(\mathsf{EF}(\mathsf{arm}_{\mathsf{fct}} = 0))$.

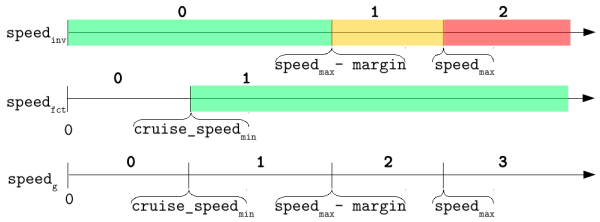

Figure 4: Partitioning of the $speed_g$ variable

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

For the speed, we have two different partitions as presented in Figure 4, one for the $\texttt{speed}_{\texttt{inv}}$ (with discrete values $\{0, 1, 2\}$) and one for the $\texttt{speed}_{\texttt{fct}}$ (with discrete values $\{0, 1\}$). The resulting glue variable $\texttt{speed}_{\texttt{g}}$ will then have four values as presented in Figure 4. We thus have the formal definition:

$\texttt{INVAR speed}_{\texttt{g}} = 0 \leftrightarrow \texttt{speed}_{\texttt{inv}} = 0 \ \& \ \texttt{speed}_{\texttt{fct}} = 0;$
$\texttt{INVAR speed}_{\texttt{g}} = 1 \leftrightarrow \texttt{speed}_{\texttt{inv}} = 0 \ \& \ \texttt{speed}_{\texttt{fct}} = 1;$
$\texttt{INVAR speed}_{\texttt{g}} = 2 \leftrightarrow \texttt{speed}_{\texttt{inv}} = 1 \ \& \ \texttt{speed}_{\texttt{fct}} = 1;$
$\texttt{INVAR speed}_{\texttt{g}} = 3 \leftrightarrow \texttt{speed}_{\texttt{inv}} = 2 \ \& \ \texttt{speed}_{\texttt{fct}} = 1.$

For the arm variable, it is much simpler:

$\texttt{INVAR arm}_{\texttt{g}} = 0 \leftrightarrow \texttt{arm}_{\texttt{inv}} = 0 \ \& \ \texttt{arm}_{\texttt{fct}} = 0;$
$\texttt{INVAR arm}_{\texttt{g}} = 1 \leftrightarrow \texttt{arm}_{\texttt{inv}} = 1 \ \& \ \texttt{arm}_{\texttt{fct}} = 1.$

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 if the speed threshold $\texttt{speed}_{\texttt{max}} - \texttt{margin}$ 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. The interventions are defined in Table 3.

5.1.2 Results

We compare in this section the results obtained without and with the approach through the definition of functionalities. To graphically describe the strategies, we represent

Speed of the platform	**Real speed interval**	**Discrete variable**
The speed is low	$speed_{inv} < speed_{max} - margin$	$\texttt{speed}_{\texttt{inv}} = 0$
The speed is within the margin	$speed_{max} - margin \leq speed_{inv} < speed_{max}$	$\texttt{speed}_{\texttt{inv}} = 1$
The speed is higher than the maximum allowed value	$speed_{inv} \geq speed_{max}$	$\texttt{speed}_{\texttt{inv}} = 2$
Position of the arm		**Discrete variable**
The arm is not extended beyond the platform		$\texttt{arm}_{\texttt{inv}} = 0$
The arm is extended beyond the platform		$\texttt{arm}_{\texttt{inv}} = 1$

Table 1: Partitioning of the variables $speed_{inv}$ and arm_{inv}

Speed of the platform	**Real speed interval**	**Discrete variable**
The platform is considered stopped or the speed is lower than the minimum cruise speed	$speed_{fct} < cruise\_speed_{min}$	$\texttt{speed}_{\texttt{fct}} = 0$
The platform moves at the minimum cruise speed or higher	$speed_{fct} \geq cruise\_speed_{min}$	$\texttt{speed}_{\texttt{fct}} = 1$
Position of the arm		**Discrete variable**
The arm is folded		$\texttt{arm}_{\texttt{fct}} = 0$
The arm is extended beyond the platform		$\texttt{arm}_{\texttt{fct}} = 1$

Table 2: Partitioning of the variables $speed_{fct}$ and arm_{fct}

Name	**Precondition**	**Effect**
Brake	$\texttt{speed}_{\texttt{inv}} = 0$ & $\texttt{next}(\texttt{speed}_{\texttt{inv}}) = 1$	$\texttt{next}(\texttt{speed}_{\texttt{inv}}) = \texttt{speed}_{\texttt{inv}} - 1$
Block_arm	$\texttt{arm}_{\texttt{inv}} = 0$	$\texttt{next}(\texttt{arm}_{\texttt{inv}}) = 0$

Table 3: Definition of the interventions for the invariant SI_1

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 $\{\texttt{safe}_1, \texttt{safe}_2, \texttt{w}_1, \texttt{w}_2, \texttt{w}_3\}$ in Figure 5), from every other non-catastrophic state. Note that the names used have been shortened in the Figure 5: $s_i = speed_{inv}$, $s_{max} = speed_{max}$, $m = margin$ and $a_i = arm_{inv}$. Only one strategy is synthesized by the SMOF algorithm, using both the interventions of braking the platform and blocking the arm. This strategy is represented in Figure 5.

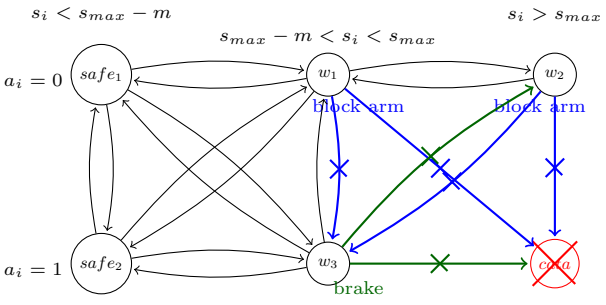

Figure 5: Strategy synthesized for the invariant SI_1 with generic permissiveness properties.

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 $\{\texttt{safe}_1, \texttt{safe}_2\}$. After running the synthesis, in addition to the previous strategy we have the strategy detailed in Figure 6. This strategy only uses the braking intervention. This can be preferable in some cases, because the use of the arm is then never impacted and even if the monitor triggers the brakes the system can keep manipulating objects.

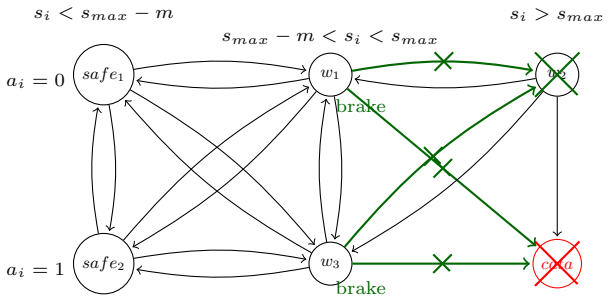

Figure 6: Additional strategy synthesized for the invariant SI_1 with the custom permissiveness properties.

In the previous example, an additional strategy is synthesized as shown in Figure 6 thanks to the definition of custom permissiveness properties. More generally, considering the functionalities to define the permissiveness requirements allows to synthesize a more important number of strategies, 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 The invariant SI_2

5.2.1 *Modeling*

The considered invariant is: *SI_2: the robot must not enter*

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

Table 4: Definition of the interventions for the invariant SI_2

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: $\mathtt{d} : \{0, 1, 2\}$, $\mathtt{0}$ representing the robot into the prohibited zone, $\mathtt{1}$ the robot close to the prohibited zone and $\mathtt{2}$ 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 $\mathtt{0}$ represents the robot stopped and $\mathtt{1}$ the robot moving. A dependency between the distance and the velocity variable 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 Table 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 functionalities $\mathsf{move} : \mathsf{AG}(\mathsf{EF}(\mathtt{v_{fct}} = 1))$ and $\mathsf{stop} : \mathsf{AG}(\mathsf{EF}(\mathtt{v_{fct}} = 0))$ 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:

$\mathtt{INVAR\ v_g} = 0 \leftrightarrow \mathtt{v_{inv}} = 0\ \&\ \mathtt{v_{fct}} = 0;$
$\mathtt{INVAR\ v_g} = 1 \leftrightarrow \mathtt{v_{inv}} = 1\ \&\ \mathtt{v_{fct}} = 1.$

5.2.2 Results

The synthesis of strategies with the braking intervention and the move and stop functionalities does not give any result. Indeed, applying the brakes till stopped violates the permissiveness property associated to the move functionality because 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 either to change the interventions or to accept a restriction of the functionality. 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 performed before the intervention of the monitor. We have the following resulting permissiveness property: $\mathsf{restricted\ move} : \mathsf{EF}(\mathtt{v_{fct}} = 1)$. With the restricted permissiveness property, the synthesis generates one strategy which is modeled in Figure 7.

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.

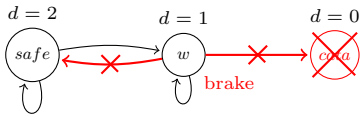

Figure 7: Strategy synthesized for the invariant SI_2 with restricted functionality.

As future work, we wish to synthesize multi-level strategies using higher level interventions (plan a new trajectory for example) that would avoid reaching these deadlock states, and thus avoid the need of a human intervention.

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 using CTL following a simple template. In defining the functionalities we require the reachability of a reduced set of states. Therefore, a more important number of strategies can be synthesized by the SMOF tool. In the studied example, it provided new strategies only requiring the use of one intervention instead of two. Also some of the problems which had no solutions with a generic definition of permissiveness properties could have one or several with custom properties.

Whenever it is not possible to synthesize a safety strategy, our approach proposes 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 to 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 in the existing SMOF tooling would only require minor changes on the template. The synthesis algorithm can remain unchanged. In future work, we wish to implement those minor changes and test them to evaluate the scalability issues on existing examples.

We also work on extending our approach to cover different types of monitor interventions. For example, we could search for multi-level strategies combining guaranteed and non-guaranteed interventions (having a probability of success different fro one, possibly depending on the operational situation). The monitor would first try the most permissive interventions even if their effect is not guaranteed, and would trigger the less permissive but safe ones only in last emergency.

7. REFERENCES

[1] NuSMV home page. http://nusmv.fbk.eu/.

[2] Safety Monitoring Framework. LAAS-CNRS Project, https://www.laas.fr/projects/smof. accessed September 2017.

[3] Safety Monitoring Framework | resources. LAAS-CNRS Project, https://www.laas.fr/projects/smof/resources. accessed September 2017.

[4] S. Adam, M. Larsen, K. Jensen, and U. P. Schultz. Rule-based Dynamic Safety Monitoring for Mobile Robots. *Journal Of Software Engineering In Robotics*, 7(1), 2016.

[5] 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*, 1(1):11–33, Jan. 2004.

[6] N. Delgado, A. Q. Gates, and S. Roach. A taxonomy and catalog of runtime software-fault monitoring tools. *Transactions on Software Engineering*, 30(12):859–872, 2004.

[7] Y. Falcone, J.-C. Fernandez, and L. Mounier. What can you verify and enforce at runtime? *International Journal on Software Tools for Technology Transfer*, 14(3):349–382, 2012.

[8] J. Fox and S. Das. *Safe and sound - Artificial Intelligence in Hazardous Applications*. AAAI Press - The MIT Press, 2000.

[9] J. Guiochet. Hazard analysis of human-robot interactions with HAZOP-UML. *Safety Science*, 84, 2016.

[10] 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.

[11] 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.

[12] H. Jiang, S. Elbaum, and C. Detweiler. Inferring and monitoring invariants in robotic systems. *Autonomous Robot*, 41(4), Apr. 2017.

[13] M. Leucker and C. Schallhart. A brief account of runtime verification. *Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009.

[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, J. Guiochet, T. Guhl, S. Walther, and V. Magnanimo. Saphari d1.3.1. report on safety monitoring framework and safe control strategies. LAAS-CNRS 15009, 2015.

[16] 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*, Dec. 2016.

[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)*, Apr. 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] F. Py and F. Ingrand. Dependable execution control for autonomous robots. In *International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[20] S. Roderick, B. Roberts, E. Atkins, and D. Akin. The ranger robotic satellite servicer and its autonomous software-based safety system. *IEEE Intelligent Systems*, 19(5), 2004.

[21] SAPHARI. Safe and Autonomous Physical Human-Aware Robot Interaction. Project supported by the European Commission under the 7th Framework Programme, www.saphari.eu, accessed May 2015, 2011-2015.

[22] R. Woodman, A. F. Winfield, C. Harper, and M. Fraser. Building aafer robots: Safety driven control. *International Journal of Robotics Research*, 31(13), 2012.