SMOF: A Safety Monitoring Framework for Autonomous Systems

Mathilde Machin, Jérémie Guiochet, Hélène Waeselynck, Jean-Paul Blanquart, Matthieu Roy, and Lola Masson

***Abstract*—Safety-critical systems with decisional abilities, such as autonomous robots, are about to enter our everyday life. Nevertheless, confidence in their behavior is still limited, particularly regarding safety. Considering the variety of hazards that can affect these systems, many techniques might be used to increase their safety. Among them, active safety monitors are a means to maintain the system safety in spite of faults or adverse situations. The specification of the safety rules implemented in such devices is of crucial importance, but has been hardly explored so far. In this paper, we propose a complete framework for the generation of these safety rules based on the concept of safety margin. The approach starts from a hazard analysis, and uses formal verification techniques to automatically synthesize the safety rules. It has been successfully applied to an industrial use case, a mobile manipulator robot for co-working.**

***Index Terms*—Autonomous system, fault tolerance, model checking, safety, safety monitor, safety rules.**

I. Introduction

NEW safety-critical systems are about to enter into our homes, work places, or hospitals. These systems can operate without human intervention and take their own decisions, while performing tasks in human vicinity. Confidence in the safety of such autonomous systems, e.g., assistive robots, medical robots, or co-workers, is the main barrier to their deployment in everyday life. Indeed, many threats can affect the behavior of these systems and induce catastrophic consequences. Threats may come from faults in the design, from physical failures at run-time, or may be due to complex interactions with users or the environment. These systems have also to cope with many uncertainties in the perception of an unstructured environment. Attaining the required level of confidence calls for the combined utilization of a set of methods that can be grouped into four major categories [1]: 1) fault prevention; 2) fault removal; 3) fault forecasting; and 4) fault-tolerant methods. In this paper, the focus is on fault tolerance.

Manuscript received March 16, 2016; accepted November 9, 2016. Date of publication December 16, 2016; date of current version April 13, 2018. This work was supported in part by 7th Framework Programme, under Grant 287513 and CPSE-Labs, Horizon2020 Programme under Grant 644400, both funded by the European Union: Saphari. This paper was recommended by Associate Editor S. Nahavandi.

M. Machin, J. Guiochet, H. Waeselynck, M. Roy, and L. Masson are with the LAAS-CNRS, University of Toulouse, 31031 Toulouse, France (e-mail: guiochet@laas.fr).

J.-P. Blanquart is with Airbus Defence and Space, F-31402 Toulouse, France.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMC.2016.2633291

No complex system can be considered as fault-free, and this is particularly true of autonomous systems having nondeterministic decisional software. Moreover, adverse and unspecified situations may also induce a hazardous behavior. Fault tolerance mechanisms are needed to deal with residual faults and adverse situations in operation. "Safety monitors" are one such type of mechanisms. Their role is to observe the system and its environment, and to trigger interventions that keep the system in a safe state. In practice, their specification and design is usually done in an *ad hoc* manner. A very limited set of safety rules is considered, and interventions often have a permanent effect (e.g., if a bumper detects contact with an obstacle, it disconnects the power of a mobile robot). We argue that, in the future, versatile autonomous systems will have to deal with a richer set of safety rules, managing autonomously the activation and deactivation of temporary interventions. While ensuring safety, the rules should still permit functionality of the considered system, a property that we call *permissiveness*.

Extending previous work [2]–[4], this paper presents the safety monitoring framework (SMOF) to specify such rules. The contributions are as follows.

1) The proposed framework provides a principled approach for specifying the safety rules of an online monitor. It starts from hazard analysis and uses formal verification techniques to synthesize the rules that address the hazards.
2) The rule synthesis problem is formalized by a set of well-structured concepts. These concepts are integrated into an SMOF modeling template and a tool available online [5], offering predefined elements and auto-completion facilities.
3) The rule synthesis tool of SMOF accommodates both safety and permissiveness requirements. The permissiveness requirements may be tuned to the system under consideration.
4) An industrial case study exemplifies the use of SMOF. It involves a mobile robot for co-working in a manufacturing setting.

This paper is structured as follows. Section II discusses related work. Section III introduces baseline and concepts for safety monitoring and provides an overview of the proposed framework. Section IV presents how the system and properties are formalized. Section V explains the algorithm to synthesize the safety rules, using the new symbolic model verifier (NuSMV) model-checker. Section VI deals with the last step of the process to check the consistency of all synthesized rules. Section VII presents an application to a real industrial case study: it demonstrates the complete approach on a mobile robot, from hazard analysis to implementation of synthesized rules. We conclude in Section VIII, outlining the benefits and limitations of SMOF, and discussing future directions.

II. Related Work

In the dependability community, fault tolerance is defined as the means to avoid service failures in the presence of faults [1]. It is carried out with error detection and recovery mechanisms. This approach has been used in robotics architectures, at functional level [6] or decisional level [7], [8] for instance. A well-established form of fault tolerance dedicated to safety is *safety monitoring*, through which the functional system is forced to a safe state (recovery) should some hazardous behavior be detected (error detection). Safety monitors appear in the literature under many different terms, such as: *safety kernel* [9], *safety manager* [10], *autonomous safety system* [11], *checker* [12], *guardian agent* [13], *safety bag* [14], *emergency layer* [15], or *diverse monitor* [16].

A quite similar approach can be found in the runtime verification community [17], where verification techniques (e.g., theorem proving or model checking) are used to check whether a run of a system under scrutiny satisfies or violates a given property. Most of the work deals with nonindependent layers, focusing on code instrumentation, with dependency issues between the safety layer and functional layer. Nevertheless, in [18]–[20] the independence or isolation issue is tackled.

It is also important to mention supervisor synthesis developed by [21], which is focusing on delivering a correct-by-construction controller. The supervision consists in inhibiting unwanted behavior of the controller (not in launching active interventions). Conceptually similar to supervisor synthesis, the work in [22], presents a synthesized robot controller integrating properties. Again, the focus is on property enforcement by filtering out undesired behavior of the controller. In contrast, the monitoring framework studied in this paper calls for an independent safety device, without any access to the internal behavior of the controller, and able to trigger active interventions.

Other architectural solutions can be mentioned from the literature. Among the previously cited work, the monitor appearing under the term *checker* [12] is inserted between the decisional and functional layers of a software control architecture. Its aim is to detect and recover from deviations between requests from the decisional layer and observations coming from the functional layer. The RoboX tour guide robot [23] has monitoring mechanisms at several levels: the operating system, the software controller, and an independent hardware. Our approach does not consider multilevel monitoring mechanisms, which will be studied in future extensions (see the discussion of future work at the end of this paper). But it addresses issues for which the current solutions are largely *ad hoc*: the identification of which properties to enforce, and the automated synthesis of rules based on a set of candidate safety interventions.

For the safety rule identification process, Woodman *et al.* [24] presented a very similar workflow to ours. They use a hazard analysis technique (HAZOP) to identify hazards and determine the corresponding safety rules, which are if-then-else rules. From sensor observations, the monitor (safety layer) sends actuation inhibitions to both controller and software actuator interface. The main point of the method is to take into account sensor uncertainty. Compared to our approach, permissiveness is implicit and the monitor is actually dependent on the functional layer. Moreover, the determination of safety rules from the hazards is not supported by formal methods. Alexander *et al.* [25] and Black and Koopman [26] have mentioned the need for rigorous specification methods for the safety mechanisms, and in particular pointed out a consistency issue when handling many safety rules. However, they did not provide a systematic tool-supported process such as the one presented in this paper.

To conclude, to the best of our knowledge, there is currently no framework similar to SMOF. Previous work has addressed different issues, like implementation issues in runtime verification (automated generation of code for instrumentation and property verification), language theory issues in supervisor synthesis, and architectural issues in work mixing several monitor and control layers. SMOF targets independent monitoring devices only. It fulfills the need for strengthening the specification of such safety mechanisms, in relation to hazard analysis, and is based on formal methods. As will be exemplified by the industrial case study in Section VII, the formalization also provides useful feedback about which property can or cannot be enforced, depending on the observation and intervention means made available to the monitor. Hence, the approach is intended for early system development stages, where some design decisions are still open and can be tuned to accommodate safety concerns.

III. Baseline and Concepts

A. Concepts

Taking inspiration from the diverse monitor defined by [16], we define a *safety monitor* as a device responsible for safety, in opposition to the main control channel which is responsible for all other functional and nonfunctional requirements of the system. The monitor is the ultimate protection against interaction faults or arbitrary behavior of the control channel that adversely affect safety. To this end, it is equipped with means for context observation (i.e., sensors) and able to trigger safety interventions. The whole safety channel must be assigned a high integrity level. In compliance with safety standards, there must be some degree of independence from the main channel. Moreover, the monitor should justifiably be able to trust the sensors and actuators it uses.

Assuming appropriate implementation means, our focus is on specifying the behavior of the monitor, i.e., on determining which intervention should be applied, and when. Baseline concepts are as follows.

A safety invariant (SI) is a sufficient condition to avoid a hazardous situation. We consider, conservatively, that if an SI

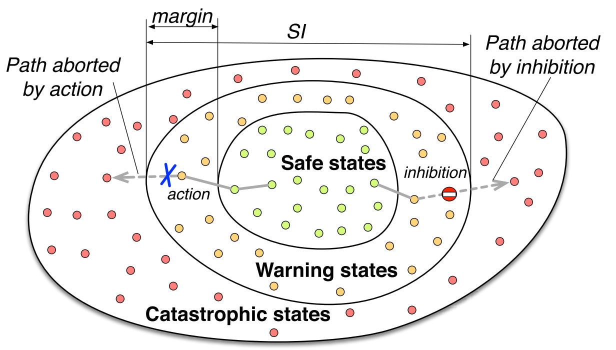


Fig. 1. Partition of system states in catastrophic, warning, and safe states.

is violated, there is no possible recovery action. Thus, any state violating the SI is a *catastrophic state*.

Example: "The robot speed shall not exceed 3 m/s" (the speed beyond which harm is considered to be inevitable).

A safety intervention is an ability of the monitor to constrain the system behavior in order to prevent the system from violating an SI. An intervention is only effective when its *preconditions* are satisfied. We distinguish two types of interventions: 1) inhibitions and 2) actions.

A safety inhibition prevents a change in system state.

Example: "Lock the wheels," with "robot stationary" as a precondition.

A safety action triggers a change in system state.

Example: "Apply brake."

As recovery is not possible, interventions have to be applied before the catastrophe, i.e., in noncatastrophic states with some margin from the catastrophic border. Hence, the set of noncatastrophic states is partitioned into warning states, where interventions are applied, and safe states, in which the system operates without constraint (see Fig. 1). The warning states are defined such that every path from a safe state to a catastrophic state passes through a warning state.

A safety rule defines a way of behaving in some warning states. It is composed of a condition and an intervention to apply when the condition is true. The condition identifies a subset of warning states. The intervention is intended to abort catastrophic paths via these warning states.

Example: "If the robot speed is greater than 2.5 m/s then apply brake." (taking a 0.5 margin below the safety threshold of 3 m/s.

A safety strategy is a set of safety rules intended to ensure an SI. It should abort all paths to the catastrophic states.

Example: "If the robot speed is greater than 2.5 m/s then apply brake; if the ground slope is greater than 10% then apply brake."

The strategy specified for the monitor can be analyzed according to two antagonistic viewpoints.

1) Safety: The strategy is said safe if it ensures the SI in the monitored system, i.e., it guarantees the nonreachability of the catastrophic states.

2) Permissiveness: The strategy is said permissive if the monitored system is able to freely move inside a specified state space. A maximum permissiveness would authorize all possible states including catastrophic ones.

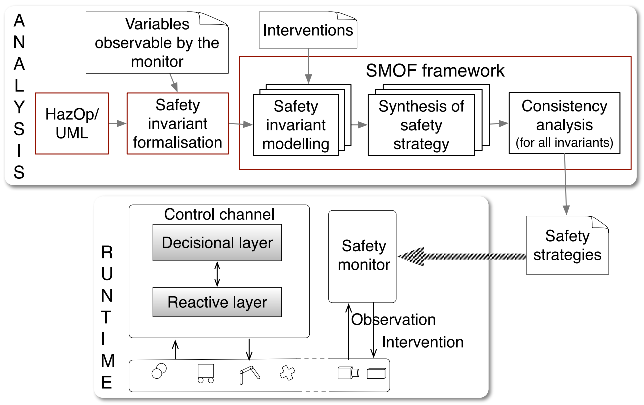

ANALYSIS
Variables observable by the monitor
Interventions
SMOF framework
HazOp/ UML
Safety invariant formalisation
Safety invariant modelling
Synthesis of safety strategy
Consistency analysis (for all invariants)
RUNTIME
Control channel
Decisional layer
Reactive layer
Safety monitor
Safety strategies
Observation
Intervention

Fig. 2. Overview of the process.

For example, a strategy allowing the system to operate at high speed, manipulating a sharp object in human presence, would be highly permissive but unsafe. Intrinsically, safety is ensured by reducing the possible behavior of the system: as soon as the monitor triggers an intervention, it reduces permissiveness. As autonomous systems are particularly versatile, they are supposed to operate in many different states. To achieve the assigned tasks, the monitored system should keep its ability to reach a wide range of states.

In this paper, we propose a systematic approach to produce strategies that are safe but restrict permissiveness only to the extent necessary to ensure safety. Safety is ensured by the identification of safety margins, while permissiveness involves state reachability properties that can be tuned by the user.

B. Process Overview

Fig. 2 presents the overall process to use SMOF. The process starts with an HAZOP-unified modeling language (UML) hazard analysis [27]. In this method, the use of the system is modeled with UML use case and sequence diagrams. Each message is then analyzed with generic keywords such as "none," "as well as," etc. That results in deviations in the system, whose causes, consequences and severity are assessed, focusing on operational hazards. This analysis outputs SIs expressed in natural language. We consider as a running example a mobile robot with a manipulator arm and the SI. *The arm must not be extended beyond the platform when the platform velocity is greater than* V_0.

Each SI is then expressed formally with predicates on variables that are observable by the monitor. This second step may induce an early feedback on the system design, by revealing the lack of key observation mechanisms.

We focus for now only on predicates involving variables compared to fixed thresholds. This type of safety threshold is amenable to formal verification and is used in many real systems. Let us consider two monitor observations: 1) the absolute speed v and 2) a Boolean observation of the arm position a (true when the arm is above the base, false when the arm is extended). The exemplary SI is formalized as $SI = (v < V_0) \lor (a = \text{true})$. Its negation defines the catastrophic states as $v \geq V_0 \land a = \text{false}$.

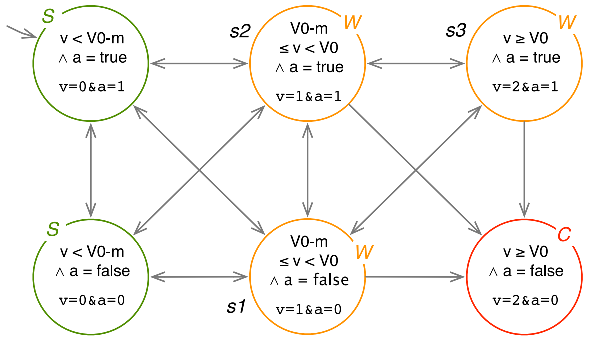


Fig. 3. Exemplary behavior model. There are two state variables: the velocity v ∈ {0,1,2} encoding the partition $\{[0, V_0 - m[, [V_0 - m, V_0[, [V_0, V_{\max}[\}$, and the arm position a ∈ {0,1} encoding {true, false}. The warning states (labeled W) are those that lead the system to the catastrophe C in one step.

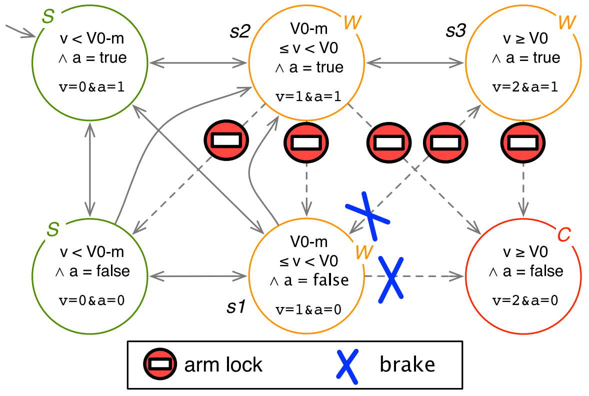


Fig. 4. Exemplary behavior modified by a safety strategy.

The third step is to build state-based models using the SMOF modeling template. In order to keep models simple enough to be validated, each SI is modeled separately. In this step, we determine the partition of noncatastrophic states into safe states and warning states by splitting intervals or sets of variable values. This is done one variable after another. For example, the velocity interval $[0, V_0[$ from SI is partitionable in two intervals according to a margin m : $[0, V_0 - m[$ and $[V_0 - m, V_0[$. In the case of the arm position, the observation is Boolean, hence no margin exists. The resulting model is shown in Fig. 3. There are three warning states in which interventions may be applied. We consider two available interventions: 1) the monitor is able to engage brake (action) and 2) to prevent the arm from extending (inhibition).

The fourth step is the strategy synthesis. Fig. 4 illustrates a returned strategy, which applies brake in $s1$ and arm inhibition in $s2$ and $s3$. The interventions delete a number of transitions, the ones directly leading to the catastrophic state plus extra ones. As the system can still reach all noncatastrophic states, the strategy is safe and permissive. When the model does not admit such a strategy, the user has several options.

1) Reduce permissiveness, for example accept that a given warning state be no longer reachable.
2) Add new interventions, which will have to be implemented in the real system.
3) Modify the model, which may require revisiting the hazard analysis.

As SIs are processed separately, the fifth step checks the consistency of strategies that ensure different invariants. The checked set of strategies are then implemented in a real time safety monitor, for online verification.

The rest of this paper provides a detailed presentation of the steps supported by the SMOF template and tools: modeling, synthesis, and consistency analysis. Then, the complete process is demonstrated on an industrial case study.

IV. Modeling With SMOF Template

An SMOF model formalizes the part of the system related to one SI, seen from the monitor's point of view. It gathers all information necessary to produce strategies that ensure the SI.

1) *Behavior:* Automaton of the system in absence of the monitor, containing all paths to the catastrophic states.
2) *Interventions:* Abilities of the monitor to constrain the system behavior.
3) *Safety and Permissiveness:* Desired properties of the monitor action.

Fig. 5 gives a structured view of these concepts. The *behavior* contains *warning states*. A *safety rule* associates an *intervention* (or a combination of interventions) to one of these warning states. A *strategy* is composed of potentially many safety rules. *Safety* and *permissiveness* are *properties* attached to a pair of behavior and strategy. Since the behavior is a fixed part of the model, while the strategy is the varying part changed by the synthesis process, we will feel free to say that the strategy (rather than the pair) satisfies the properties. *Validity* is a side-property linked to the intervention preconditions (see Section IV-B).

To formalize our model in this section, we choose to use languages and tools available in the model checking community. The synthesis presented in Section V is based on the model-checker NuSMV2 [28]. In what follows, code and output of NuSMV are given in typewriter font.

A. Behavior

The SI involves observable variables compared with fixed values. The comparisons determine a first abstraction of the variables into classes of values, e.g., the values that are below or above a safety threshold. The partition is then refined by the consideration for margins (formal conditions for the existence of a margin are studied in [2]). To ease the modeling in NuSMV, the resulting classes of values are encoded by integers, as illustrated by the state predicates in Fig. 3. For example, in the initial state, the class of values $v < V_0 - m$ is encoded as $v = 0$ and overall the abstract version of v is in the range $0 \ldots 2$. The user must also enter a definition of the catastrophic states, with the predefined name `cata`. In the running example, this would be

```
DEFINE cata := v=2 & a=0.
```

The behavior states result from the Cartesian product of the ranges of all abstract variables. The user does not need to list the states: NuSMV transparently builds the state space from the declaration of variables. By default, all the combinations

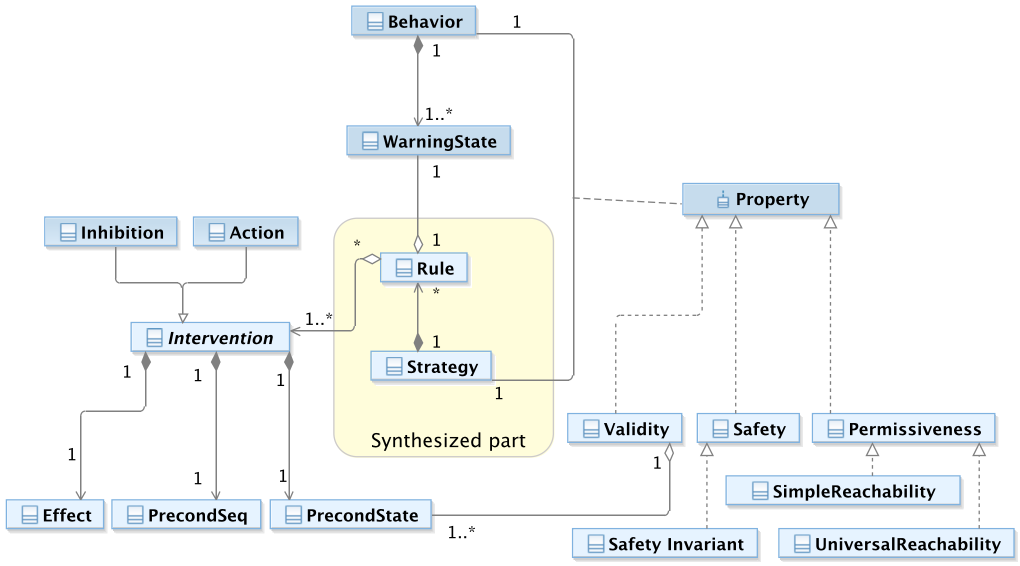


Fig. 5. Meta-model of the SMOF modeling template.

of abstract variable values (i.e., states) are possible and all transitions between pairs of states are implicitly declared. The user can add constraints to delete states and transitions that would be physically impossible.

The most common constraint is the continuity of a variable, e.g., the velocity cannot "jump" from `0` (standing for $[0, V_0 - m]$) to `2` ($[V_0, V_{\max}]$). The SMOF template provides a predefined module `Continuity` to conveniently declare a continuous variable `x`. It encapsulates the constraint `next(x) = x |x+1| x−1`, i.e., the next value of `x` can stay in the same interval or move to an adjacent interval, but it cannot jump from one interval to another that has no common boundary. SMOF also includes the predefined constraint that a catastrophe is irreversible, i.e., the `cata` states are sink states.

The Cartesian product assumes that the variables are independent from each other. If they are not, the user has to declare the dependency constraints. The ability to express them depends on the current abstraction. For example, let us consider two observable variables, position p and velocity v of a robot. To express the constraint that p cannot change when $v = 0$ (a very narrow interpretation of derivative), we need an abstraction distinguishing the concrete zero value of v from other values. In our running example, the partition would be too coarse, as all concrete values in $[0, V_0 - m]$ are encoded by the same abstract value 0. If a dependency is not modeled, the abstract model has less constraints than it should, or from another point of view, it has too many transitions. The over-approximation may affect the permissiveness analysis. However, if this "super-graph" is safe, so is the "real" model.

B. Interventions

An intervention is modeled by its effect and preconditions. The effect is a constraint that cuts some transitions from the current state, to reduce the set of possible next states.

The effect is guaranteed only if the preconditions hold. We distinguish two preconditions. The state precondition, noted

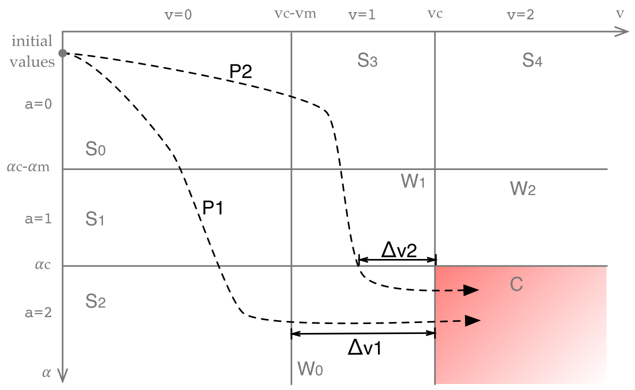


Fig. 6. Example of sequential precondition: "braking is triggered by a transition from a state satisfying $v < v_c - v_m$."

$Precond_{State}$, models the states in which the intervention can be applied. For example, it is not desirable to physically lock the wheels of a moving vehicle. The strategy is said valid if it never applies the intervention in states violating $Precond_{State}$ (*Validity* property). The effectiveness of an intervention may also depend on the system history. We choose to only take into account the state preceding the application of the intervention, and consider a sequential precondition, noted $Precond_{Seq}$, that must hold in this state. It constrains the transitions that trigger the intervention, when moving from a state in which the intervention is not applied to a state in which it is.

Fig. 6 gives an example of sequential precondition for the intervention "braking." The intended effect is to avoid a concrete velocity greater than V_c. Let us consider two paths P1 and P2 in the state space (we assume nine abstract states). In each case, the intervention is triggered upon entering the warning state W_0. Along P1, the braking is triggered as soon as the monitors detects the crossing of the margin threshold $V_c - V_m$. The margin accounts for the sampling period of observation and worst case reaction time, so that the effect is guaranteed. On the contrary, along P2, the velocity can be arbitrarily close to V_c: there is no guarantee that the system has time to brake before V_c is reached.

SMOF provides a predefined `Intervention` module, which could be used as follows to declare the braking preconditions and effect:

```
-- myInterv : Intervention(precondState,
    precondSeq, flag_myInterv, effect)
brake : Intervention(TRUE, v=0 & next(v)=1,
    flag_brake, next(v)!=v+1);
```

In this example, `brake` has no *Precond*$_{State}$ (a `TRUE` value is passed). The *Precond*$_{Seq}$ parameter requires the braking to be triggered on a transition from `v=0` to `v=1` (using the operator `next`). The effect is that the next value of `v` cannot increase. The `flag_brake` variable is a placeholder for the braking condition in synthesized strategies.

The module encapsulates a Boolean variable encoding the status of the intervention, currently applied or not. A predefined constraint states that the intervention is applied if and only if it is asked by the strategy (according to the synthesized `flag`) and the effect is guaranteed (according to the preconditions).

The NuSMV code of the module Intervention and a description of its parameters are given as follows:

```
MODULE Intervention(precond_st, precond_seq, flag,
     effect, mode)
-- precond_st = without next(), written from the
    application state
-- precond_seq = with or without next(), written
    from the step before application (the next
    step of this expr corresponds to step where
    the intervention is applied).
-- flag = condition of application. Without next.
    Must be set to "flag_intervname"
-- effect = with next(), written from the state of
     application

VAR applied : boolean;
ASSIGN
init(applied):= FALSE;
next(applied):= case
applied=FALSE & precond_seq & next(flag &
    precond_st) : TRUE;
applied=TRUE & next(flag & precond_st) : TRUE;
TRUE : FALSE;
esac;

TRANS applied -> effect

-- For validity property
DEFINE __pre_c :=  precond;
```

C. Properties

Safety and permissiveness properties are modeled in computational tree logic (CTL), a branching time logic fully supported by NuSMV. Time along paths is modeled by three operators: 1) X for a property to hold in the next state; 2) G to hold on the entire path; and 3) F to hold eventually. The branching aspect is modeled by A, all the branches, and E, there exists a branch. A CTL operator is composed of one branching operator and one time operator. It is applied on states, or more generally on statements about the system state.

Safety is predefined as the unreachability of the catastrophic states declared by the user, i.e., in CTL, AG $\neg$ *cata*.

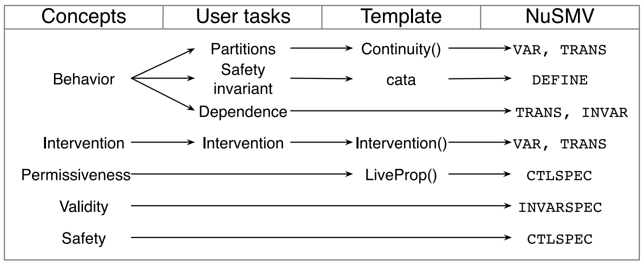


Fig. 7. Concepts and implementation of the SMOF model.

Permissiveness is modeled by two reachability properties applied to any noncatastrophic state s_{nc}.

1) *Simple Reachability (EF* s_{nc}*):* The state s_{nc} is reachable from the initial state.
2) *Universal Reachability (AG EF* s_{nc}*):* The state s_{nc} is reachable from any reachable state.

For safety, we pessimistically consider that several independent variables may change their values simultaneously. We call such simultaneous modifications *diagonal transitions* by reference to the two variable case (see Fig. 3, transition from initial state to $s1$). Relying on those possible but unlikely transitions to ensure permissiveness is not desirable as it would be too optimistic. A more complete definition of reachability properties that ignore diagonal transitions during permissiveness checking is provided in [3] and used by the tools we developed.

The SMOF template has a `LiveProp` module defining the simple and universal reachability of an arbitrary state. A tool automatically instantiates it for each noncatastrophic state of the behavior model entered by the user. By default, universal reachability is required, but the user can edit a script to change the permissiveness requirements of some states. When the strategy synthesis returns no solution for the stringent requirements, it may be a good compromise to accept the simple reachability, or even the unreachability, of states that are not essential to the robot functionality.

D. Summary

The SMOF template and NuSMV model of our running example are available online [5]. The template includes predefined modules, parts to be edited by the user and generated parts. To model the running example, the user only needs to enter five lines, reproduced below, corresponding to the declaration of the abstract variables, of the catastrophic states and of the available interventions

```
-- my var : Continuity(max,init)
v : Continuity(2,0);
a : Continuiy(1,0);
cata := v=2 & a=0;

-- myInterv : Intervention(precondState,
    precondSeq, flag_myInterv, effect)
brake : Intervention(TRUE, v=0, flag_brake, next(v
    )!=v+1);
lock_arm : Intervention(a=1, TRUE, flag_lock_arm,
    next(a)!=0);
```

Fig. 7 links the SMOF concepts, the user tasks, the template modules, and the NuSMV syntactic elements. User tasks are required only for specifying the behavior and interventions. The properties are either predefined (safety) or generated (validity and default permissiveness). As presented below, the tool also generates the list of warning states, i.e., of states having a transition to a catastrophic state. It is done in preparation for the synthesis of strategies: the warning states are candidate states for applying interventions

```
---------------
-- GENERATED --
---------------
------ Warning states
DEFINE flag_st1 := a=0 & v=1;
DEFINE flag_st2 := a=1 & v=1;
DEFINE flag_st3 := a=1 & v=2;
```

V. Synthesis of Strategies

The synthesis algorithm takes as inputs the behavior model for an invariant, the available interventions and the properties to ensure (validity, safety, and permissiveness). It outputs a set of alternative strategies, each of them satisfying the properties. Conceptually, a strategy assigns a combination of interventions to each warning state. Technically, this is encoded by the definition of flags in the model. In the running example, the strategy applying brake in s_1 and arm lock in s_2 and s_3 (see Fig. 4) would be encoded as

```
DEFINE flag_brake := flag_st1;
DEFINE flag_lock_arm := flag_st2 | flag_st3;
```

where flag_sti denotes the definition of a warning state.

For a system with n warning states and m interventions, the number of candidate strategies is 2^{mn} (accounting for all possible combinations of interventions in the warning states). A naive algorithm could enumerate all the strategies, encode them one by one in the model and call NuSMV to check for the satisfaction of the properties. But such a brute force search would be very inefficient. Rather, our algorithm builds a tree of strategies, enabling us to prune branches during the search (branch-and-bound algorithm). A formal description of the tree and pruning criteria is presented hereafter.

A. Tree of Strategies

Let I be the set of interventions, of size m, and $I_C = 2^I$ the set of *intervention combinations*. In the running example, $I = \{a, b\}$ and $I_C = \{a, b, ab, \emptyset\}$ (abbreviating the set notation). Let $S_w = \{s_1, \ldots, s_n\}$ be the set of warning states of the behavior, of size n. A strategy N is a function that maps each warning state to a combination of interventions

$$N : S_w \rightarrow I_C.$$

It is defined by a set of pairs $N = \{(s_1, i_1), \ldots, (s_n, i_n)\}$.

We say N is a *satisfying* strategy if it satisfies the validity, safety, and permissiveness properties. Furthermore, we focus on *minimal* satisfying strategies, i.e., strategies from which no intervention can be removed. A satisfying strategy $N = \{(s_1, i_1), \ldots, (s_n, i_n)\}$ is minimal if there does not exist a different satisfying strategy $N' = \{(s_1, i'_1), \ldots, (s_n, i'_n)\}$ such that $\forall k \in [1, n], i'_k \subseteq i_k$. For instance, let $N_1 = \{(s_1, ab), (s_2, b)\}$ and $N_2 = \{(s_1, a), (s_2, b)\}$ be satisfying strategies. N_1 is not minimal due to the existence of N_2.

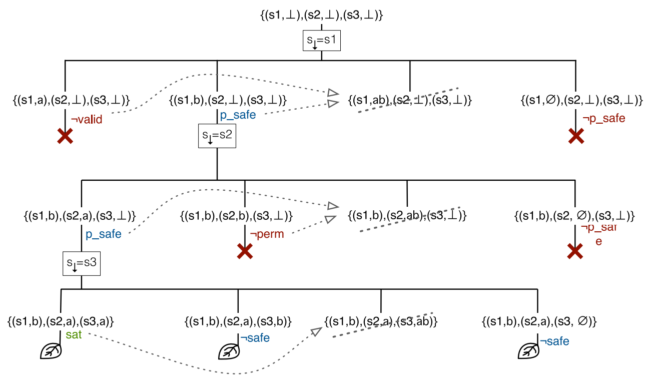


Fig. 8. Exemplary search tree. The chosen state $s_\downarrow$ is framed under the parent node. A cross indicates the pruning of all child subtrees, and dotted lines indicate the pruning of combined siblings. Nodes are labeled by the property determining the pruning.

We now introduce the *undefined* combination of interventions noted $\perp$, allowing us to consider partially defined strategies $N : S_w \rightarrow I_C \cup \{\perp\}$. The search for minimal satisfying strategies can then be seen as the exploration of a tree rooted by the undefined strategy $N_0 = \{(s_1, \perp), \ldots, (s_n, \perp)\}$ (see Fig. 8). Given a node N, building its children requires the choice of a state $s_\downarrow$ in which the interventions are not yet defined, i.e., $N(s_\downarrow) = \perp$. If no such state exists, N is a fully defined strategy and a tree leaf. Otherwise, the children of N, noted N_i, are the 2^m nodes such as $\forall s \neq s_\downarrow,\ N_i(s) = N(s)$ and $N_i(s_\downarrow) = i$, with $i \in I_C$.

As shown in Fig. 8, subtrees are pruned during the search. Given the examination of a node N, the search may prune all its child subtrees (which is visualized by a cross under the node) and also some subtrees rooted by sibling nodes (visualized by dotted lines). More precisely, the pruned sibling nodes N' exhibit a specific relation with the current node N, that of being a *combined sibling*. Let N and N' be children of N_p after choosing $s_\downarrow$. N' is a combined sibling of N if $N'(s_\downarrow) \subset N(s_\downarrow)$. For example, let us consider the children of $N_p = \{(s_1, b), (s_2, \perp), (s_3, \perp)\}$ after choosing $s_\downarrow = s_2$: the node $\{(s_1, b), (s_2, ab), (s_3, \perp)\}$ is a combined sibling of $\{(s_1, b), (s_2, a), (s_3, \perp)\}$.

The decision to prune subtrees depends on the properties satisfied by N. The strategy is thus encoded and model-checked. The encoding interprets the undefined interventions $\perp$ as no intervention in the state. At the end of the tree exploration, the search returns all visited leaf nodes found to be satisfying strategies. In Fig. 8, the search would return a single strategy: the leftmost bottom node labeled `sat`.

B. Pruning Criteria

Table I gives an overview of the pruning criteria.

The first criterion applies to a strategy N that is `!valid`. For instance, in Fig. 8, strategy $\{(s_1, a), (s_2, \perp), (s_3, \perp)\}$ is `!valid` because the intervention that locks the arm folded is applied in a warning state where the arm is unfolded. Child strategies of N define interventions in other warning states, but this will not fix the problem in the first one. Either the first warning state becomes unreachable, and so the child strategy

TABLE I
PRUNING CRITERIA

Synthesis		Node property		Prune relative nodes
V2	V1	1	`!valid`	Descendants and combined siblings
		2	`!perm`	Descendants and combined siblings
		4	`!p_safe`	Descendants
		3	`sat`	Descendants and combined siblings
		5	`p_safe`	Combined siblings

is not permissive, or the state is reachable and the child strategy remains invalid. So, the children of an invalid strategy are either invalid or not permissive. Similarly, a combined sibling of N defines additional interventions in the same warning state, and the first intervention still makes the strategy invalid. Recursively none of the children of the combined siblings are solutions to the problem, we can prune the subtrees of the combined siblings.

Consider now a partial strategy that is not permissive. All its children are `!perm` as well, because adding interventions can only cut transitions. In the same way, its combined siblings are `!perm`. The second criterion prunes descendants and combined siblings of nonpermissive strategies.

The third criterion detects partial strategies that cannot result in safe leaf strategies. It is evaluated using a *subgraph* of the behavior, where the warning states with undefined ($\perp$) interventions are removed. For example, in Fig. 9, the subgraph does not contain s_2 and s_3. The safety analysis focuses on reaching the catastrophic state via the warning states for which the interventions have been decided. If the strategy is safe in this subgraph we say that it is partially safe (`p_safe`). For example, the strategy of Fig. 9 is `p_safe` thanks to the effect of intervention `b` applied in s_1. Now, suppose a strategy is not `p_safe`. The defined interventions fail to remove all catastrophic paths in the subgraph. The descendants of the strategy will not remove these paths. They can only delete transitions from warning states that are currently outside the subgraph. So, all descendants are unsafe and can be pruned.

The usefulness of this criterion depends on how $s_\downarrow$ is chosen at each step of the tree building. In Fig. 9, the target strategy is a child of the root $\{(s_1, \perp), \ldots, (s_n, \perp)\}$, after choosing $s_\downarrow = s_1$. Suppose we had rather chosen $s_\downarrow = s_3$: any child strategy would be trivially `p_safe` because s_3 is unreachable in subgraphs without s_1 and s_2. To avoid such cases, the synthesis algorithm has a constraint on the $s_\downarrow$ chosen to build the children of a node N. It must be such that N' is `!p_safe`, with N' defined by

$$N'(s_\downarrow) = \emptyset \quad \text{and} \quad \forall s \neq s_\downarrow \quad N'(s) = N(s).$$

Under this constraint, the subgraphs are not trivially `p_safe` and the pruning by the third criterion is more efficient.

The fourth criterion, `sat`, which includes safety, validity and permissiveness, discards strategies that are not minimal. Assume N is a satisfying strategy. Its descendants and combined siblings might be `sat` as well, but they involve additional interventions and are thus not minimal. The corresponding subtrees can be pruned. N is appended to the list of solutions returned by the search.

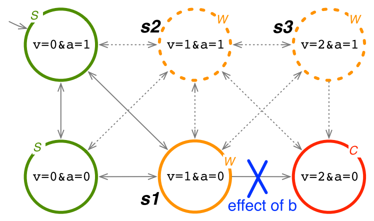


Fig. 9. View to check whether $\{(s_1, b), (s_2, \perp), (s_3, \perp)\}$ is `p_safe`. The states and transitions in plain line constitute the considered subgraph.

The fifth criterion discards the nonminimal strategies that are missed by the fourth one. For instance, in the tree of Fig. 8, $\{(s_1, b), (s_2, a), (s_3, a)\}$ is a satisfying solution. The fourth criterion prunes its sibling $\{(s_1, b), (s_2, a), (s_3, ab)\}$, but not the nonminimal "cousin" $\{(s_1, b), (s_2, ab), (s_3, a)\}$. The fifth criterion is able to prune it at the time of the intervention decision for s_2. It exploits the information that a current node is `p_safe`, e.g., that $\{(s_1, b), (s_2, a), (s_3, \perp)\}$ is `p_safe`. Intuitively, no other intervention is needed in the subgraph. The criterion discards all combined siblings, like the sibling assigning ab to s_2 while a suffices.

Adding the fifth criterion ensures minimality of the returned strategies but may remove solutions. As soon as interventions have a $Precond_{Seq}$, their sufficiency for the subgraph does not imply their sufficiency for the complete behavior. For example, in Fig. 9, assume that intervention `b` is not effective if the preceding state was s_3: we may actually need an additional intervention in s_1 to account for paths via s_3.

Our synthesis algorithm can be configured into two variants, without or with the fifth criterion (see Table I). Variant 1 returns a superset of the set of minimal satisfying strategies. If it returns no solution, then it is sure that no solution exists for the model. If there are many solutions, this version is very long to execute as it explores all of them, plus a number of nonminimal ones. Variant 2 returns a subset of minimal satisfying strategies. It is not conclusive regarding the absence of solutions, but may return minimal strategies with a shorter tree traversal than variant 1.

C. *Evaluation Using Artificial Models*

We have developed a parallelized implementation of the synthesis tool. Its performance is assessed using artificial models. It allows us to consider search spaces with more than 10^{18} strategies. The number of minimal solutions ranges from zero to thousands.

The models, listed in Table II, are generated as follows. One variable has three values, the others have two values. The initial state has all variables at value 0, and the only catastrophic state has variables at their maximum values. Each variable may be controlled by two interventions: an action that decreases the value of the variable, or an inhibition that freezes the value. The models consider various combinations of variables and interventions. For example, model `3var_3a` has three variables controlled by decreasing actions only (suffix `_a`). Model `3var_3i` has the variables controlled by freezing inhibitions only (suffix `_i`). Model `3var_6` has all six possible interventions. First, we did not consider sequential

TABLE II
EXPERIMENTAL PRUNING PERFORMANCE

Model	Number of strategies	**Variant 1** Visited nodes (%)	Solutions	Time	**Variant 2** Visited nodes (%)	Solutions	Time
2var_2A	64	4.7	0	50ms	4.7	0	50ms
2var_2I	64	12.5	1	230ms	12.5	1	150ms
2var_4	4 096	0.56	9	210ms	0.44	6	170ms
3var_3A	10^6	10^{-3}	0	200ms	10^{-4}	0	200ms
3var_3I	10^6	10^{-2}	40	3.1s	10^{-3}	12	1.0s
3var_6	10^{12}	10^{-6}	17 106	4min	10^{-8}	1 128	18.3s
4var_4A	10^{18}	10^{-14}	0	3s	10^{-14}	0	2.7s
4var_4I	10^{18}	-	-	-	10^{-11}	12 954	26min

preconditions, and then we introduced some. Performance is assessed on a Intel Core I7-4770 processor running at 3.4 GHz with 16 GB of memory.

Table II presents the results for interventions with no sequential precondition. For each model, the number of complete strategies (i.e., of tree leaves) is given. It would be the number of steps of brute-force search. The two variants are assessed. Their first column allows a comparison to brute-force search: it gives the ratio of visited nodes (i.e., tree nodes) to the number of complete strategies, expressed as a percentage. It could be higher than 100%, since the tree nodes include both partial and complete strategies. But the observed values are very low, which demonstrates the overall efficiency of the pruning criteria. The second column gives the number of solutions found and the third column the execution time.

Since there is no sequential precondition, variant 2 returns the exact set of minimal solutions. Both variants agree that models with *actions* only (suffix a) have no solution. Intuitively, as the actions force the variables to change, no permissive strategy can be found. For other models, variant 1 keeps a large number of nonminimal solutions. The fourth criterion does not suffice to catch them, the fifth criterion would be needed. As a result, variant 1 explores a larger proportion of nodes and takes significantly longer times than variant 2.

To check the undue removal of solutions by variant 2, we must add sequential preconditions. We made several attempts, which we present briefly. We started by the following preconditions: the interventions of the three-valued variable are effective only if the preceding state has this variable at 0. Synthesis was rerun. A post-processing function analyzed the solutions returned by variant 1 to eliminate the nonminimal ones, giving the exact number of minimal solutions. Variant 2 found all of them in each case. We then strengthened the sequential preconditions with intervariable dependencies. In addition to the previous conditions, the freezing intervention required one of the two-valued variables to be at 1 in the preceding state; the decreasing intervention required the same variable to be at 0. It allowed us to observe the removal of minimal solutions. For example, for model 3var_6, there were 96 minimal solutions and variant 2 returned 64. It was never the case that a model had solutions and variant 2 returned none. We finally created a specific model, with preconditions referring to precise states, and managed to observe zero returned solution while there was one to find.

From what precedes, the best approach is to favor variant 2, and to refine the results by using variant 1 only in cases for which zero or few solutions are found.

Looking again at Table II, it may be surprising that the synthesis fails for a model with four variables (for variant 1) or takes half an hour (for variant 2). One might wonder whether the approach is useful in realistic cases. First, the number of variables is not unrealistic. An SI models only one safety-relevant aspect of a system. In the real system studied by [2], each invariant had no more than two variables. Second, the artificial models we used are generic, i.e., they have many interventions and no variable dependencies. It induces that there may be numerous solutions to find, much more than in real cases. This case is supported by the industrial case study in the next section. The models typically had one solution, and the longest synthesis time observed with variant 1 was 0.32 s.

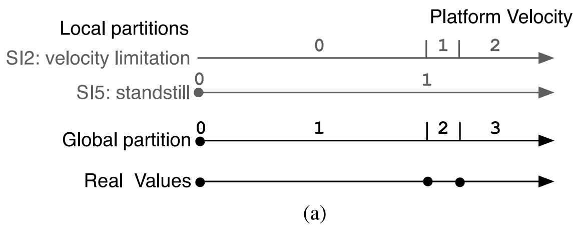
VI. ANALYSIS OF CONSISTENCY

Within our approach, each SI is modeled separately and has a dedicated safety strategy. To check the effect of merging the strategies retained for each invariant, the SMOF models are turned into NuSMV modules and gathered in one global model. A main module contains the glue information.

First, the variables from different SMOF models may be dependent. The user has to add the dependencies, in the same way as she entered them in SMOF models. A particular case of dependency is to use the same observation in two SMOF models. For example, consider an invariant of velocity limitation and another invariant using the information of whether or not the system is at stop. These invariants share a common observation, velocity, but they have a different partition of its values. The merging of the partitions follows a systematic approach illustrated in Fig. 10.

Second, the launching of interventions has an impact on the whole system. Each time an intervention is asked by one local strategy, it may have an effect in all the SMOF models in which the intervention is modeled. The main module controls this by means of a global flag, defined as the disjunction of the local flags for this intervention. When the global flag is true, the effect of the intervention is determined in each SMOF model, under the local preconditions.

Analysis of consistency aims to check whether two strategies, synthesized from different invariants, apply incompatible interventions at the same time (e.g., braking and accelerating).



```
INVAR pf_vel=0 <-> si4.pf_vel=0 & si2.pf_vel=0;
INVAR pf_vel=1 <-> si4.pf_vel=1 & si2.pf_vel=0;
INVAR pf_vel=2 <-> si4.pf_vel=1 & si2.pf_vel=1;
INVAR pf_vel=3 <-> si4.pf_vel=1 & si2.pf_vel=2;
```

(b)

Fig. 10. Merging of two partitions of the platform velocity observation: for velocity limitation (invariant SI2) and standstill determination (SI5). (a) Principle: form a global partition based on the real values. (b) Formal encoding: declare a global variable and glue constraints.

The model-checker easily detects roughly inconsistent cases like both increasing and decreasing a variable. But there might be less obvious cases not captured in the abstract models. So, we require the user to specify the forbidden combinations. Given a pair of incompatible interventions (i, j), their nonconcomitance is formulated as

$$AG\neg(\text{globalFlagInterv}_i \wedge \text{globalFlagInterv}_j).$$

Permissiveness is also rechecked. An intervention launched by one SMOF model could impair the reachability of states of other SMOF models.

VII. Industrial Case Study

To demonstrate our approach, we apply the whole process, from HAZOP analysis down to implementation, to a case study provided by KUKA Robotics.

The system is composed of a mobile platform and an articulated arm with seven axis (see Fig. 11). It is an industrial co-worker in a manufacturing setting, sharing its workspace with human workers. It takes and places boxes, which contain parts, on shelves, tables, or on the top of the robot platform in order to convey them. A restricted area is defined, i.e., an area forbidden to the robot. The robot arm can be hand-guided when an operator needs to manipulate the gripped box to inspect it. This interaction phase begins and ends with an haptic gesture, a vertical push on the robot "elbow."

The system is equipped with a safety layer satisfying most of the implementation assumptions of our framework. In particular, the programmable safety behavior associates interventions with conditions on observation variables. Classically, only fixed thresholds are admitted to define the conditions. The monitor can observe a small subset of system variables and has two possible interventions: 1) engaging the arm brake and 2) engaging the platform brake.

A. HAZOP-UML

The system use cases have been modeled in 15 UML sequence diagrams. The HAZOP analysis results in more than hundred HAZOP lines with a nonzero severity. In practice, it does not mean that 100 SIs need to be modeled. There are many similar HAZOP lines that can easily be grouped. The analysis ends up with 13 SIs listed in Table III.

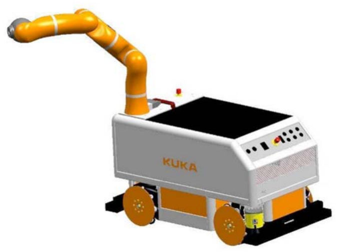


Fig. 11. Mobile manipulator of KUKA.

TABLE III
SIs, Resulting From HAZOP-UML Analysis

SI1	The velocity of robot arm must not be greater than V_0.
SI2	The velocity of robot platform must not be greater than V_1.
SI3	The robot must not enter the restricted area.
SI4	The robot platform must not collide with a human.
SI5	The robot arm must not be extended beyond the platform footprint when the platform moves.
SI6	A gripped box must not be tilted more than α_0.
SI7	A collision between a human and the robot arm must not hurt the human.
SI8	The velocity of any point of the robot must not be greater than V_2.
SI9	The robot arm must not drop a box.
SI10	The robot arm must not clamp human parts.
SI11	The robot gripper must not clamp human parts.
SI12	The robot must not override boxes laid on tables, shelves and robot storage.
SI13	The robot must follow the hand-guiding.

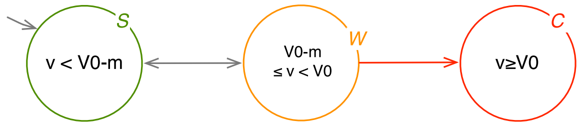
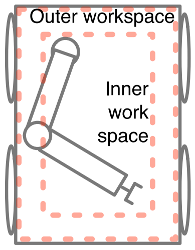


Fig. 12. Behavior of the SMOF model for SI1.

B. SMOF Models and Synthesized Strategies

In the following discussion, we put emphasis on how the approach is impacted by the limited observation and intervention means available to the safety layer. For this real system, some invariants had straightforward models, some required the elaboration of indirect observations derived from the direct ones, and some were impossible to address, pointing to a lack of observability or controllability.

1) Simple SMOF Models: The first SI, SI1, is the limit of the arm velocity, formalized by $v < V_0$. Defining the velocity of a robot arm with seven axis is not obvious. The available observation v is the maximum translational Cartesian velocity among the velocities of the axis center points. The SMOF model is very simple, as there is only one observation, which admits a margin (see Fig. 12). A unique strategy is synthesized and, as expected, the arm brakes are engaged in the margin state. Safety threshold V_0 and margin value have been calculated by KUKA engineers. The SMOF model for the limit of the platform velocity (SI2) is similar.



(a)

Outer \ Inner	inside	outside
inside	0	1
outside	-	2

(b)

Fig. 13. Observation of the arm extension. (a) Workspaces. (b) Values of the observed arm position.

2) Elaboration of Indirect Observations: SI3 states that the system has to stay away from the restricted area. The SMOF model uses one intervention, platform braking, and one observation variable, the difference between the distance to the restricted area (sensed) and the braking distance (computed from the platform velocity). The variable is not a direct observation but its computation from available observations is simple enough to be done in the safety monitor. Similarly to SI3, a collision between platform and human (SI4) is avoided by observing the difference between the braking distance and the distance to any obstacle (sensed by laser).

SI5 inspired our running example: *the arm must not be extended beyond the platform when the platform moves*. In the real system, the safety layer can observe whether all points of the arm are inside a rectangular area defined by the user. The area corresponding to the footprint of the platform gives us the safety threshold: if any arm point is outside this area, the arm is extended. To take a margin, we define an inner rectangular workspace (see Fig. 13). The arm position can then be encoded by three values, depending on the observations for the two areas. Platform velocity is observed as a Boolean variable (`0` for standstill, `1` for movement) with no margin.

For this invariant, both interventions are relevant. The margin of the arm extension (i.e., the distance between the two workspaces) is based on the braking distance of the arm. To ensure that the arm will not reach the outside of the platform, the braking must be engaged as soon as the border of the inner workspace is crossed. Hence, arm braking has a sequential precondition (the preceding state is such that `arm_pos=0`). The platform braking has no sequential precondition but a state one. It prevents the platform from starting if applied in a state where it is already standstill. Given the inertia of the platform and the brake power, it is not possible for the initially standstill platform to reach any significant velocity between the time of the braking command and the time when brakes are effectively engaged. Hence, the platform braking can be seen as instantaneously effective, ensuring `next(pf_vel)=0`.

The synthesized strategy is to brake the platform when the platform is standstill and the arm is close to or beyond the footprint limit. The arm brake is engaged when the platform is moving and the arm is in the margin.

3) Elaboration of Hypothesized Values: In some cases, the state variables of interest cannot be derived from observations. Their value must be hypothesized based on the observations, in a conservative way.

SI6—*a box in the gripper must not be tilted too much*. The arm is able to pick boxes containing parts. As the boxes have an open top, parts may fall if the box is too much tilted. To formalize this invariant, the required observations are: 1) the presence of a box in the gripper and 2) the angles (or the maximum absolute angle, denoted `alpha`) of the gripper with respect to the Cartesian x- and y-axis. While `alpha` is observable, the presence of a box is not. We make the assumption that the gripper takes a box when the end-effector is close to the robot storage and the gripper closes. (For sake of simplicity, we only consider here the case of the robot storage, and exclude the cases of tables and shelves.) The box is released when the gripper opens. Hence, our modeling involves a hypothesized state variable, `box`, that is updated according to the observation of the gripper status (open, close) and the distance z to the storage from close ($= 0$) to far away ($= 2$).

The safety property is first formulated as `cata := alpha=2 & box=1`, i.e., the gripper presumably carries a box and the angle is above the safety threshold. The synthesis returns no result. The monitor is indeed not able to prevent the direct transition form the warning state `alpha=2 & z=0 & gripper=open & box=0` to `cata`. In this case, making parts fall close to the robot storage is not very hazardous because they fall from a low height. The new version of the safety property is `cata := alpha=2 & box=1 & z=2`. The synthesis returns one strategy that is to brake the arm when a box is in the gripper, the end-effector is in the margin distance or far from the robot storage, and the angle is greater than the margin threshold.

Another example is SI7. The collision between the arm and a human being is hypothesized via the arm velocity and the external torque, computed from the torques sensed and a physical model of the arm (mass and geometry, internal torques from weight and intended acceleration). An adequate strategy is synthesized.

4) Unsuccessful Cases: Invariants SI8–13 were not addressed due to lack of observable variables or interventions. For example, the vectorial sum of platform and arm velocities is not observable (SI8). An intervention to lock the gripper would be necessary to prevent box dropping (SI9). In many observability issues, the relevant variables could have been indirectly derived from existing observations, but in a way that was deemed too complex: the derived information would not be trustable enough to be used by the safety monitor. A relevant example is the hand-guiding of the robot (SI13). Checking whether the arm follows the intended movement would be computationally expensive (accounting for the movement and torques sensed on all axes) and untrustable. As the interaction protocol begins and ends with haptic gestures, the monitor could not even trustily determine that a hand-guiding phase is ongoing. A solution would be to simplify the protocol (e.g., by using a push button), and to replace SI13 by a stringent limitation of arm velocity and acceleration during hand-guiding. It would ensure that the robot is harmless whether or not it correctly follows the guiding.

By providing feedback on the SIs that can or cannot be ensured, our approach can help the designers to modify the critical features of the system in the early stages.

TABLE IV
DEFINITION OF TEST CASES AND RESULTS OF THEIR EXECUTION

Definition of test cases			Executions				Assessment
Number	Description	Test objective	Monitor	Arm braking	Platf. braking	Violated SI	Test verdict
0	Arm movement (above the inner workspace), then platform movement.	Permissiveness	No				PASS
			Yes				
1	The controller tries to reach an arm velocity that is above the safety threshold, with a normal acceleration.	SI1	No			SI1	PASS
			Yes	√			
2	The controller tries to reach an arm velocity that is above the safety threshold, with a high acceleration.	SI1	No			SI1	FAIL
			Yes	√		SI1	
3	The controller tries to reach a platform velocity that is above the safety threshold, with a normal acceleration.	SI2	No			SI2	PASS
			Yes		√		
4	The controller tries to reach a platform velocity that is above the safety threshold, with a high acceleration.	SI2	No			SI2	FAIL
			Yes		√	SI2	
5	The controller tries to reach the restricted area.	SI3	No			SI3	PASS
			Yes		√		
6	During the platform movement, the arm moves and goes outside the platform footprint.	SI5	No			SI5	PASS
			Yes	√	√		
7	During the arm movement, the arm goes outside the footprint and stops there. Then the platform moves.	SI5	No			SI5	PASS
			Yes		√		
8	During the platform movement, the arm moves above the inner workspace.	Permissiveness (SI5)	No				PASS
			Yes				
9	The arm moves beyond the platform footprint, and comes back to stop above the inner workspace. Then, the platform moves.	Permissiveness (SI5)	No				PASS
			Yes		√		

C. Analysis of Consistency

The SMOF models and strategies for SI1–7 are gathered in one global model. The partitions of arm velocity (from SI1 and SI7) and platform velocity (from SI2 and SI5) are merged as exemplified by Fig. 10. The triggering of arm braking and platform braking are synchronized: as soon as one SMOF model triggers a brake, the main module triggers the same brake in all SMOF models (with an effect that depends on local preconditions).

The consistency analysis checks that incompatible interventions are not applied at the same time. In this system, there are only two interventions and they are not incompatible. We nevertheless checked their nonconcomitance and NuSMV returned a counterexample. But again, no issue is raised by a concomitant application of arm and platform braking. The global model also passed the permissiveness checks.

D. Implementation and Tests

For demonstration purposes, we implemented and tested the synthesized strategies on a KUKA robot.

1) Experimental Constraints: The experiments are done on a real system in the environment of the KUKA laboratory. As a consequence, experiments requiring collisions with a human being are dangerous. We were not able to test the strategies corresponding to the collision with the arm (SI7) and the platform (SI4). Furthermore, the robot made available to us had no gripper, making it impossible to implement the strategy of SI6. Finally, we did not have programming access to the safety layer. We had to implement the strategies in a separate thread in the high-level Java interface of the system. It does not provide us the same guarantees as the safety layer would, in particular as regards real-time behavior.

2) Test Cases: To test the strategies, we define a base case: from an initial standstill position, the arm moves above the inner workspace, stops, and then the platform moves. All other test cases introduce a deviation from the base case that targets an SI. The first column of Table IV presents the test cases and the associated test objectives.

When the objective is to test permissiveness, the behavior of the robot controller is not dangerous and should not be disturbed by the monitor. The base case serves this objective, as do cases 8 and 9. Case 8 checks that the arm is free to move during platform movement, as long as it stays above the inner workspace. Case 9 checks that the platform is free to move after the arm is back to a safe position.

When the objective is to test safety, the behavior is dangerous and the monitor must avoid the violation of an invariant. Cases 1–4 test a velocity too high, of the arm or the platform, reached with normal or high acceleration. Case 5 moves the platform into the forbidden area. Cases 6 and 7 let the platform move with the arm in a dangerous position. Either the arm unfolds as the platform moves (case 6), or it is stopped at a dangerous position before the platform moves (case 7).

3) Results: Each case is tested twice, with and without the monitor. Table IV reports the triggered interventions and violated invariants. The safety of the runs is also controlled visually by the operator. The final pass or fail verdict depends on the test objective.

For permissiveness, a test case passes if the run without the monitor is safe, and the other run with the monitor has a similar behavior (similar execution time, similar final position of the arm and the platform, no spurious intervention). Cases 0, 8, and 9 all have a pass verdict. For case 9, note that the platform braking is not a spurious intervention. It is applied at the beginning of the run, when the arm is extended and the platform should not move. But the braking is no longer active when the platform really moves later in the run.

For safety, a test case passes if the execution is unsafe without the monitor and becomes safe with it. Two of the test cases fail, those checking velocity thresholds with a high acceleration. The adequate intervention is triggered too late to be effective: there is a transient threshold overrun before velocity decreases. The problem is mainly due to experimental constraints, since we were not allowed to use the real-time safety layer. The margin values rely on the input sampling rates and bounded intervention delays, but the Java implementation does not ensure them. We repeated cases 2 and 4 several times and the real-time behavior varied. In one execution, the monitor did not even observe the traversal of a warning state: its inputs directly jumped from a value lower than the margin threshold to a value greater than the safety threshold, following a path that does not exist in the SMOF model.

These fail cases underline the criticality of the calculation assumptions taken for margins. They concern not only the dynamics of the system (e.g., its maximal acceleration) but also its data processing part. Indeed, no margin can be calculated if the processing has an unpredictable real-time behavior. In a realistic setting, the monitor should be in the safety layer.

Putting aside the real-time issues of the Java-based implementation, the experimentation revealed no other problem. As far as the strategies could be assessed given the experimental constraints, they were found safe and permissive.

VIII. Conclusion

Active safety monitors for autonomous systems have to deal with complex safety rules, inducing several interventions that should be consistent. To develop such active monitors, we proposed a formalized process based on the definition of warning states linked to a safety margin. In these warning states it is possible to trigger interventions before the system goes in catastrophic states. We proposed a complete framework, SMOF, starting from hazard analysis and ending in safety rules synthesis. In this paper, we particularly focus on the safety rule generation algorithm, and on its validation on a real industrial case study.

A major benefit of SMOF is that it provides a systematic and formal approach for the expression of safety rules, whereas it is usually done *ad hoc*, based on the expertise of the analysts only. The models are also of great importance to describe the monitor behavior in order to take it into account for the development of the functional layer. Indeed, safety margins and interventions need to be nonambiguous to determine controller reactions. Our approach is based on the use of a well-known formal language (CTL), and we proposed a template available online to simplify the use of our tool. Most of the complex verifications (like permissiveness) are automatically generated and checked. More precisely, for the selection of intervention, we propose to take into account the warning state where the system is, but also the path followed to reach this state. A last notable result is that after application on a real use case, there is no combinatorial explosion of the algorithm, and its performance is acceptable.

A main limitation of using our approach lies in the expression of dependencies or partitions of observable variables. Indeed, the efficiency of the generation could be highly increased when the analyst has a good level of expertise on interrelations between variables. Another drawback is that the current version of SMOF does not include a mechanism to activate/deactivate the safety rules depending on the task performed by the system. In most of autonomous applications, this would represent an important issue as systems tend to be more and more versatile. The proposed approach is also limited to the functional level, with simple expression of the SI using propositional logic. For now, we do not consider interventions like blocking requests from decisional layer.

Future directions concern the extension of the framework to the definition of a several warning regions, in order to trigger interventions with different levels of efficiency. For instance, a soft intervention might reduce the speed, and if needed a second hard intervention could activate emergency stop. This approach is also linked to the implementation on different layers (hardware and software) with different integrity levels. We also plan to extend SMOF to the observation of the decisional layer (e.g., task or trajectory plans), and possible intervention as rejecting requests from the decisional layer. Finally, the SMOF is about to be transferred to industry in the context of the European Cyber-Physical Systems Engineering Laboratories Project.

Acknowledgment

The authors would like to thank T. Guhl, S. Walther, and V. Magnanimo, employees of KUKA and partners of SAPHARI.

References

[1] A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr, “Basic concepts and taxonomy of dependable and secure computing,” *IEEE Trans. Depend. Secure Comput.*, vol. 1, no. 1, pp. 11–33, Jan./Mar. 2004.

[2] A. Mekki-Mokhtar, J.-P. Blanquart, J. Guiochet, D. Powell, and M. Roy, “Safety trigger conditions for critical autonomous systems,” in *Proc. IEEE Pac. Rim Int. Symp. Depend. Comput. (PRDC)*, 2012, pp. 61–69.

[3] M. Machin *et al.*, “Specifying safety monitors for autonomous systems,” in *Proc. Int. Conf. Comput. Safety Rel. Security (SAFECOMP)*, Florence, Italy, 2014, pp. 262–277.

[4] M. Machin *et al.*, “Model-checking and game theory for synthesis of safety rules,” in *Proc. 16th IEEE Int. Symp. High Assurance Syst. Eng. (HASE)*, Daytona Beach Shores, FL, USA, 2015, pp. 36–43.

[5] *Safety Monitoring Framework*, Lab. Anal. Archit. Syst., Toulouse, France, accessed on Jun. 1, 2015. [Online]. Available: https://www.laas.fr/projects/smof

[6] D. Crestani, K. Godary-Dejean, and L. Lapierre, “Enhancing fault tolerance of autonomous mobile robots,” *J. Robot. Auton. Syst.*, vol. 68, pp. 140–155, Jun. 2015.

[7] B. Lussier *et al.*, “Fault tolerant planning for critical robots,” in *Proc. Int. Conf. Depend. Syst. Netw. (DSN)*, Edinburgh, U.K., 2007, pp. 144–153.

[8] F. Ingrand and M. Ghallab, “Deliberation for autonomous robots: A survey,” *Artif. Intell.*, pp. 1–40, Nov. 2014.

[9] J. Rushby, “Kernels for safety,” in *Proc. Symp. Safe Secure Comput. Syst.*, Glasgow, U.K., 1989, pp. 210–220.

[10] C. Pace and D. Seward, “A safety integrated architecture for an autonomous safety excavator,” in *Proc. Int. Symp. Autom. Robot. Constr.*, Taipei, Taiwan, 2000, pp. 29–39.

[11] S. Roderick, B. Roberts, E. Atkins, and D. Akin, “The ranger robotic satellite servicer and its autonomous software-based safety system,” *IEEE Intell. Syst.*, vol. 19, no. 5, pp. 12–19, Sep./Oct. 2004.

[12] F. Py and F. Ingrand, “Dependable execution control for autonomous robots,” in *Proc. Int. Conf. Intell. Robots Syst. (IROS)*, Sendai, Japan, 2004, pp. 1136–1141.

[13] J. Fox and S. K. Das, *Safe and Sound: Artificial Intelligence in Hazardous Applications*. Cambridge, MA, USA: MIT Press, 2000.
[14] P. Klein, "The safety-bag expert system in the electronic railway interlocking system Elektra," *Expert Syst. Appl.*, vol. 3, no. 4, pp. 499–506, 1991.
[15] S. Haddadin *et al.*, "Towards the robotic co-worker," in *Robotics Research* (Springer Tracts in Advanced Robotics), vol. 70, C. Pradalier, R. Siegwart, and G. Hirzinger, Eds. Heidelberg, Germany: Springer, 2011, pp. 261–282.
[16] *Functional Safety of Electrical / Electronic / Programmable Electronic Safety-Related Systems–Part 7: Overview of Techniques and Measures*, IEC Standard IEC61508, p. 153, 2010.
[17] M. Leucker and C. Schallhart, "A brief account of runtime verification," *J. Logic Algebraic Program.*, vol. 78, no. 5, pp. 293–303, 2009.
[18] A. Goodloe and L. Pike, "Monitoring distributed real-time systems: A survey and future directions," NASA Langley Res. Center, Nat. Aeronaut. Space Admin., Hampton, VA, USA, Tech. Rep. NASA/CR-2010-2167242010. [Online]. Available: http://www.cs.indiana.edu/~lepike/pubs/survey.pdf
[19] L. Pike, S. Niller, and N. Wegmann, "Runtime verification for ultra-critical systems," in *Proc. 2nd Int. Conf. Runtime Verification*, San Francisco, CA, USA, 2011, pp. 310–324.
[20] A. Kane, T. Fuhrman, and P. Koopman, "Monitor based oracles for cyber-physical system testing: Practical experience report," in *Proc. 44th Annu. IEEE/IFIP Int. Conf. Depend. Syst. Netw. (DSN)*, Atlanta, GA, USA, 2014, pp. 148–155.
[21] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Control Optim.*, vol. 25, no. 1, pp. 206–230, 1987.
[22] S. Bensalem, M. Gallien, F. Ingrand, I. Kahloul, and T.-H. Nguyen, "Toward a more dependable software architecture for autonomous robots," *IEEE Robot. Autom. Mag.*, vol. 16, no. 1, pp. 66–77, 2009.
[23] N. Tomatis *et al.*, "Designing a secure and robust mobile interacting robot for the long term," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, Taipei, Taiwan, 2003, pp. 4246–4251.
[24] R. Woodman, A. F. T. Winfield, C. Harper, and M. Fraser, "Building safer robots: Safety driven control," *Int. J. Robot. Res.*, vol. 31, no. 13, pp. 1603–1626, 2012.
[25] R. Alexander, N. Herbert, and T. Kelly, "Deriving safety requirements for autonomous systems," in *Proc. SEAS DTC Tech. Conf.*, Edinburgh, U.K., 2009.
[26] J. Black and P. Koopman, "System safety as an emergent property in composite systems," in *Proc. Int. Conf. Depend. Syst. Netw. (DSN)*, Estoril, Portugal, 2009, pp. 369–378.
[27] J. Guiochet, "Hazard analysis of human–robot interactions with HAZOP-UML," *Safety Sci.*, vol. 84, pp. 225–237, Apr. 2016.
[28] A. Cimatti *et al.*, "NuSMV 2: An opensource tool for symbolic model checking," in *Computer Aided Verification*. Heidelberg, Germany: Springer, 2002, pp. 359–364.

Mathilde Machin received the Engineer degree in electrical engineering and industrial computing from ENSEEIHT, Toulouse, France, in 2012, and the Ph.D. degree in dependable computing and fault tolerance from the LAAS-CNRS, University of Toulouse, Toulouse, in 2015, with a focus to ensure safety in autonomous systems by a generic method for developing online independent safety monitoring.

She is currently with Airbus subsidiary specialized in dependability.

Jérémie Guiochet received the Ph.D. degree in model-based safety analysis of service robotics from INSA Toulouse, Toulouse, France, in 2003.

He is an Assistant Professor of computer science with the University of Toulouse III, Toulouse and with the Dependable Computing and Fault Tolerance Group at LAAS-CNRS, France. His current research interests include safety assessment, fault removal, and fault-tolerance in safety-critical autonomous systems.

Hélène Waeselynck received the Engineer degree from the National Institute of Applied Sciences of Toulouse, Toulouse, France, in 1989, and the Ph.D. degree in computer science from the National Polytechnic Institute of Toulouse, Toulouse, in 1993.

She is a Senior Research Scientist (Directrice de Recherche) with the LAAS-CNRS, University of Toulouse, Toulouse. Her current research interests include software testing and verification, with an emphasis on dependable computing systems.

Jean-Paul Blanquart received the Ph.D. degree in computer science from the University of Toulouse, Toulouse, France.

He is a Senior Specialist of Safety, Dependability, Security, FDIR, Airbus Defence and Space, Toulouse. His current research interests include formal model-based approaches to support the expression and validation of dependability and safety properties, and on the integration of such approaches into industrial processes.

Matthieu Roy received the Ph.D. degree from IRISA/University of Rennes, Rennes, France.

He is a CNRS Researcher with the Dependable Computing and Fault Tolerance Group, LAAS-CNRS, University of Toulouse, Toulouse, France. On the experimental and practical side, he has strong ties with the automotive industry, and has recently developed platforms to study human behavior in the context of human-carried distributed systems. His current research interests include theory of distributed systems, embedded systems, and real-time systems.

Lola Masson received the Engineer degree in automation and industrial computing from ENSEEIHT, Toulouse, France, in 2015. She is currently pursuing the Ph.D. degree with the Dependable Computing and Fault Tolerance Group, LAAS-CNRS, University of Toulouse, Toulouse.

She researches on multilevel active safety monitoring, such as robotic architecture, safety analysis, and fault tolerance.