Under consideration for publication in Formal Aspects of Computing

Simulation Relations for Fault-Tolerance

Ramiro Demasi[1], Pablo F. Castro[2], Thomas S.E. Maibaum[3], and Nazareno Aguirre[2]

[1] Fondazione Bruno Kessler, Trento, Italy
e-mail: demasi@fbk.eu
[2] Departamento de Computación, FCEFQyN, Universidad Nacional de Río Cuarto, Río Cuarto, Córdoba, Argentina
Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Argentina
e-mail: {pcastro,naguirre}@dc.exa.unrc.edu.ar
[3] Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada,
e-mail: tom@maibaum.org

Abstract.
We present a formal characterization of fault-tolerant behaviors of computing systems via simulation relations. This formalization makes use of variations of standard simulation relations in order to compare the executions of a system that exhibits faults with executions where no faults occur; intuitively, the latter can be understood as a specification of the system and the former as a fault-tolerant implementation. By employing variations of standard simulation algorithms, our characterization enables us to algorithmically check fault-tolerance in polynomial time, i.e., to verify that a system behaves in an acceptable way even subject to the occurrence of faults. Furthermore, the use of simulation relations in this setting allows us to distinguish between the different levels of fault-tolerance exhibited by systems during their execution. We prove that each kind of simulation relation preserves a corresponding class of temporal properties expressed in CTL; more precisely, *masking fault-tolerance* preserves liveness and safety properties, *nonmasking fault-tolerance* preserves liveness properties, while *failsafe fault-tolerance* guarantees the preservation of safety properties. We illustrate the suitability of this formal framework through its application to standard examples of fault-tolerance.

Keywords: Formal Specification, Simulation Relations, Fault-Tolerance, Program Verification

1. Introduction

The increasing demand for highly dependable and constantly available systems has focused attention on providing strong guarantees for software robustness, understood as the ability of software to continue behaving in an acceptable way despite erroneous behavior during its execution or the existence of an uncooperative environment; this is particularly true for critical systems. Some examples of such critical systems include software for medical devices and software controllers in the avionics and automotive industries. In this context, a problem that requires attention is that of reasoning about *faults*, that is, those unexpected events that

Correspondence and offprint requests to: Pablo F. Castro, Universidad Nacional de Río Cuarto, Ruta Nac. No. 36 Km. 601, Río Cuarto (5800), Córdoba, Argentina. e-mail: pcastro@dc.exa.unrc.edu.ar

could affect a system and may corrupt or degrade its performance. A related problem is that of expressing the properties of systems in the presence of such faults and providing rigorous, or mathematical, proofs of the truth of these properties.

The field of *fault-tolerance* is concerned with providing techniques that can be used to increase the robustness characteristics of software, or computer systems in general. This includes specific mechanisms for achieving fault-tolerance, as well as for appropriately modeling fault-tolerant systems, and expressing and reasoning about fault-tolerant behaviors. Some examples of traditional techniques employed to deal with the occurrence of faults are: *component replication*, *N-version programming*, *exception mechanisms*, *transactions*, etc. All of these techniques can add confidence to critical systems about their capability for dealing with faults; standard references to the field of fault-tolerance are [Avi95, LA90, PS05, SS98, TP00].

Several approaches have been proposed to deal with fault-tolerance in formal settings, with the main aim of mathematically *proving* that a given system effectively tolerates faults. For example, in [Cri85], an approach to design and verify programs that tolerate faults, where faults are formalized as operations performed at random time intervals, is proposed. Another example of a formal approach to fault-tolerance is that presented in [AG93, AK98a, AK98b], where *Unity* programs are complemented with fault steps, and the logic underlying *Unity* is used to prove properties of programs. More recently, formal approaches involving *model checking*, applied to fault-tolerance, have been proposed (e.g., see [BFG02, SECH98, YTK01]). In these approaches, temporal logics are employed to capture fault-tolerance properties of reactive systems, and then *model checking* algorithms are used to automatically verify that these properties hold for a given system. Since model checking provides fully automated analysis (for finite systems), and counterexamples are generated when a property does not hold (which is extremely helpful in finding the source of the problem in the system), model checking-based approaches to fault-tolerance provide significant benefits over other semi-automated or manual formal approaches. However, the languages employed for the description of systems and system properties in model checking do not provide a built-in way of distinguishing between normal and abnormal behaviors. Thus, when capturing fault-tolerant systems, and expressing fault-tolerance properties, the specifier needs to *encode* in some suitable way the faults and their consequences. This makes formulas longer and more difficult to understand, which has an obvious negative impact on the analysis, since the performance of model checking algorithms depends on the length of the formula being analyzed and the counterexamples generated may be harder to follow. It also has an impact on comprehension: the complexity of formulae increases and the understanding of the intention is more difficult to discern.

In this paper, we propose an alternative formal approach for dealing with the analysis of fault-tolerance, which allows for a fully automated analysis, and appropriately distinguishes faulty behaviors from normal ones. This approach provides a formalism for modeling fault-tolerant systems that features a built-in notion of abnormal transition, to capture faults. In this setting, fault-tolerance is characterized by defining simulation relations, between the desired "fault-free" or "ideal" program, and that which tolerates or deals in some way or another with faults. Since, as it is well known, a system may tolerate faults exhibiting different degrees of so called *fault-tolerance*, different simulation relations are provided for different kinds of fault-tolerance. More precisely, the kinds of fault-tolerance that we capture in our setting are *masking*, *nonmasking* and *failsafe* as defined in [Gär99]. *Masking fault-tolerance* corresponds to the case in which the system may completely tolerate the faults, not allowing these to have any observable consequences for the users; *nonmasking fault-tolerance* corresponds to the case in which, after a fault occurs, the system may undergo some process to eventually take the system back to a "good" behavior, but the occurrence of faulty behavior is observable by users; finally, *failsafe fault-tolerance* corresponds to the case in which the system may react to a fault by switching to a behavior that is safe, but in which the system is restricted in its capacity, again a situation observable by a user. Since in this approach fault-tolerance is captured via simulation relations, one is able to check that a system tolerates faults to some degree (masking, nonmasking, failsafe), without the need for user intervention, by employing variations of standard simulation algorithms, which are known to be efficient. In this paper, we follow the main ideas put forward in [Gär99], where the author characterizes each level of fault tolerance according to the class of properties preserved by the system. More precisely, *masking fault-tolerance* is defined as the preservation of liveness and safety properties under the occurrence of faults, while *nonmasking fault-tolerance* is said to occur when the program guarantees the liveness properties of the specification, even when exhibiting faults; and finally, *failsafe fault-tolerance* is defined as the preservation of safety properties in faulty scenarios. We show that our definition of masking/nonmasking/failsafe fault-tolerance by means of simulation relations satisfies the definition of masking/nonmasking/failsafe fault-tolerance given by Gärtner in the aforementioned paper, by proving that each class of simulation relations preserves the corresponding set of temporal properties (see Theorems 3.3, 3.5 and 3.9, below).

We have presented the basic ideas in [DCMA13a]; in this paper we revisit the relations defined in that paper and investigate the properties of the simulation relations defined; we demonstrate that these relations guarantee the preservation of important properties of programs when some basic assumptions, such as *fairness*, about the way in which faults occur are made. As discussed above, this shows that our characterization of (masking, nonmasking and failsafe) fault-tolerance fulfills the basic definitions given in [Gär99], this is a contribution of this paper. Furthermore, we present the algorithms for computing these simulation relations and we prove that these algorithms are correct and have polynomial runtime in the worst case. Le us remark that in this paper we present the theoretical foundations of our framework, we leave as future work the construction of software tools implementing these ideas.

The paper is structured as follows. In Section 2 we introduce the basic concepts needed to understand the rest of the paper. Section 3 presents the definitions of the simulation relations proposed to capture the fault-tolerant behavior, then some basic properties of these relations are proven. The algorithms for checking the existence of these simulation relations are introduced in Section 4, and their complexity is discussed. In Section 5, we introduce four examples to illustrate the applicability of these concepts in practice. Related work is discussed in Section 6. Finally, we discuss some further work and conclusions in Section 7.

2. Background

Let us introduce some concepts that will be necessary throughout the paper. For the sake of brevity, we assume some basic knowledge about model checking; the interested reader may consult [BK08]. We model fault-tolerant systems by means of *colored Kripke structures*, as introduced in [CKAA11]. Given a set of propositional letters $AP = \{p, q, s, \dots\}$, a *colored Kripke structure* is a 5-tuple $\langle S, I, R, L, \mathcal{N}\rangle$, where S is a (non-empty) finite set of states, $I \subseteq S$ is a (non-empty) set of initial states, $R \subseteq S \times S$ is a transition relation, $L : S \to \wp(AP)$ is a labeling function indicating which propositions are true in each state, and $\mathcal{N} \subseteq S$ is a set of *normal*, or "green" states. Without loss of generality, we assume that R is left-total, that is: for every $s \in S$, there is a $s' \in S$ such that $s\ R\ s'$. The complement of $\mathcal{N}$ is the set of "red", abnormal or faulty, states. Arcs leading to abnormal states (i.e., states not in $\mathcal{N}$) can be thought of as faulty transitions, or simply *faults*.

Note that, in this paper, we focus on finite state systems, that is, we consider that our systems can be described by a finite set of states; as argued in [Cla99], many classes of interesting systems can be captured in this way, and this restriction provides the main benefit of enabling automatic verification of systems; indeed this is one of the main assumptions when performing model checking, a successful technique that has been employed to verify important case studies. Infinite state systems can be dealt with using diverse techniques (e.g., abstraction); we do not investigate this here, referring the interested reader to [Cla99, BK08].

As is usual in the definition of temporal operators, we employ the notion of trace. Given a colored Kripke structure $M = \langle S, I, R, L, \mathcal{N}\rangle$, a *trace* is a maximal sequence of states, whose consecutive pairs are in R. That is, a sequence:

$$s_0 s_1 s_2 s_3 \dots$$

is said to be a trace of M when $s_i \in S$ and $s_i R s_{i+1}$ for every i. When a trace of M starts in an initial state, it is called an *execution* of M, and the set of executions of a structure M is denoted by $\mathcal{TR}(M)$. Normal executions are those transiting only through green states; the set of normal executions is denoted by $\mathcal{NT}(M)$.

Given a trace $\sigma = s_0 s_1 s_2 s_3 \dots$, the ith state of σ is denoted by $\sigma[i]$, the final segment of σ starting in position i is denoted by $\sigma[i..]$; and the subsegment $s_i \dots s_j$ (with $0 \leq i \leq j$) is denoted by $\sigma[i..j]$. Moreover, we distinguish among the different kinds of outgoing transitions from a state. We denote by $\dashrightarrow$ the restriction of R to faulty transitions, and $\to$ the restriction of R to non-faulty transitions. We define $Post_N(s) = \{s' \in S \mid s \to s'\}$ as the set of successors of s reachable via non-faulty (or good) transitions; similarly, $Post_F(s) = \{s' \in S \mid s \dashrightarrow s'\}$ represents the set of successors of s reachable via faulty arcs, and $Post(s) = Post_N(s) \cup Post_F(s)$. Analogously, we define $Pre_N(s')$ and $Pre_F(s')$ as the set of predecessors of s' via normal and faulty transitions, respectively; and similarly for $Pre(s)$. We also define the set of predecessors over a set S as $Pre(S) = \bigcup_{s \in S} Pre(s)$. In the same manner, for the set of predecessors over a set S via normal transitions as $Pre_N(S) = \bigcup_{s \in S} Pre_N(s)$. Moreover, $Post^*(s)$ denotes the states which are reachable from s. As explained above, we assume that every state has a successor, thus: $\forall s \in S : Post(s) \neq \emptyset$ [BK08], note that this simplifies the semantics of the logic, since under this assumption all the executions

are infinite sequences, the case of a state without successor can be modeled using a trap state (as explained in [BK08]). We also assume that, in every colored Kripke structure, for every normal (green) state there exists at least one successor state that is also normal (green); formally: $\forall s \in \mathcal{N} : Post_N(s) \neq \emptyset$, and that at least one initial state is green (i.e. $\mathcal{N} \cap I \neq \emptyset$). This guarantees that every system has at least one normal execution, i.e., $\mathcal{NT}(M) \neq \emptyset$, for any M. We denote by $\Rightarrow^*$ the transitive closure of $\dashrightarrow \cup \rightarrow$.

In order to state properties of systems, we use Computation Tree Logic [EC80, EH86] (or CTL), a well-known branching time temporal logic. We briefly introduce this logic below; the interested reader is referred to [BK08]. The syntax of CTL is defined as follows. Let $AP = \{p_0, p_1, \dots\}$ be an enumerable set of propositions. The sets Φ and Ψ of *state formulas* and *path formulas*, respectively, are mutually recursively defined as follows:

$$\begin{array}{lcl} \Phi & ::= & \top \mid p_i \mid \neg\Phi \mid \Phi \rightarrow \Phi \mid \mathsf{A}(\Psi) \mid \mathsf{E}(\Psi) \\ \Psi & ::= & \mathsf{X}\Phi \mid \Phi\, \mathcal{U}\, \Phi \mid \Phi\, \mathcal{W}\, \Phi \end{array}$$

Intuitively, these operators have the following interpretation: A (*for all paths or computations*), E (*for some paths or computations*), $\mathsf{X}\phi$ (*in the next moment in time ϕ is true*), $\phi\, \mathcal{U}\, \psi$ (*ψ is true at some moment in the future, and until ψ becomes true, ϕ is true*) and $\phi\, \mathcal{W}\, \psi$ (*either ψ becomes true in the future and ϕ holds until ψ holds, or ϕ is always true*).

Moreover, $\mathsf{A}(\phi\, \mathcal{U}\, \psi)$ (*on all future paths, ϕ is true until ψ becomes true*), $\mathsf{EX}\phi$ (*on some future path ϕ is true at the next moment*), are examples of combinations of path quantifiers and temporal operators. Other boolean connectives (here, state operators), such as $\wedge$, $\vee$, etc., are defined as usual. Also, traditional temporal operators $\mathsf{G}\phi$ (always in the future ϕ is true) and $\mathsf{F}\phi$ (eventually ϕ is true) can be expressed, as $\mathsf{G}(\phi) \equiv \phi\, \mathcal{W}\, \bot$ (here $\bot$ is the constant $false$), and $\mathsf{F}(\phi) \equiv \top\, \mathcal{U}\, \phi$. When useful, we use these temporal operators that improve the readability of formulas. Furthermore, it can be proved that any CTL formula can be expressed by using the operators $\mathsf{E}\, \mathcal{U}$, EG and EX; this fact will be useful when performing inductive proofs on CTL formulas.

Now, we formally state the semantics of the logic. The standard boolean operators have the usual semantics. We first define the relation $\vDash$ for state formulas as follows:

- $M, s \vDash \mathsf{A}(\psi) \Leftrightarrow$ for every $\sigma \in \mathcal{TR}(M)$, we have that $\sigma \vDash \psi$,
- $M, s \vDash \mathsf{E}(\psi) \Leftrightarrow$ for some $\sigma \in \mathcal{TR}(M)$, we have that $\sigma \vDash \psi$,

The relation $\vDash$ for executions is defined by:

- $M, \sigma \vDash \psi\, \mathcal{U}\, \psi' \Leftrightarrow \exists i \geq 0 : M, \sigma[i] \vDash \psi'$ and $\forall j : 0 \leq j < i : M, \sigma[j] \vDash \psi$.
- $M, \sigma \vDash \psi\, \mathcal{W}\, \psi' \Leftrightarrow$ either $\exists i \geq 0 : M, \sigma[i] \vDash \psi'$ and $\forall j : 0 \leq j < i : M, \sigma[j] \vDash \psi$, or $\forall i \geq 0 : M, \sigma[i] \vDash \psi$,
- $M, \sigma \vDash \mathsf{X}\psi \Leftrightarrow M, \sigma[1] \vDash \psi$.

We denote by $M \vDash \varphi$ the fact that $M, s \vDash \varphi$ holds for every initial state s of M, and by $\vDash \varphi$ the fact that $M \vDash \varphi$ holds for every colored Kripke structure M. CTL-X is the fragment of CTL without the next operator (X), let us note that most of the temporal formulas in this text belong to this sub logic. A CTL formula is in *Positive Normal Form* (or PNF) [BK08] if the negation operator is only applied to proposition letters. A standard result is that any CTL formula can be written in PNF, this is achieved by applying the dualities of the temporal operators, the interested reader is referred to [BK08] for the details.

We should note some points about our model of computation. Several works [AG93, AAE04, Dij76, Gär99] describe programs in a *guarded command* style. A guarded command is composed of a boolean condition over the actual state of the system and an assignment, written as *Guard* $\rightarrow$ *Command*. These syntactical constructions are called actions, and a program consists of a collection of actions. Moreover, some actions are used to represent faults (as done in [AG93, AAE04, Gär99]). Furthermore, distributed systems can be devised, where we may have several programs interacting concurrently by means of shared memory or channels of communications; the interested reader is referred to [Abr10, BK88, CM89, Lam94] for a detailed introduction to this style of programming. The important point here is that we can map these programs to colored Kripke structures (as explained, for instance, in [BK08]), mapping variable valuations to states and actions to transitions; here green transitions represent non-faulty actions and red ones capture faulty actions. An example of this is shown in Figure 2. This is a simple example called *Never 7*, introduced in [Bra06] and also used in [Bon08]. The program has eight states and the system specification requires that state 7 is not reached in the future, and the invariant predicate of the program is the set $\{0, 1, 2\}$. The behavior of this small system can be expressed by the program shown in Figure 1.

Normal Actions:

$$(state = 0) \rightarrow (state := 1)$$
$$(state = 1) \rightarrow (state := 2)$$
$$(state = 2) \rightarrow (state := 0)$$

Faulty Actions:

$$(state = 3) \rightarrow (state := 4)$$
$$(state = 6) \rightarrow (state := 7)$$
$$(state = 1) \rightarrow (state := 3)$$
$$(state = 1) \rightarrow (state := 6)$$
$$(state = 4) \rightarrow (state := 5)$$
$$(state = 5) \rightarrow (state := 7)$$
$$(state = 7) \rightarrow (state := 7)$$

Fig. 1. A simple program "Never 7".

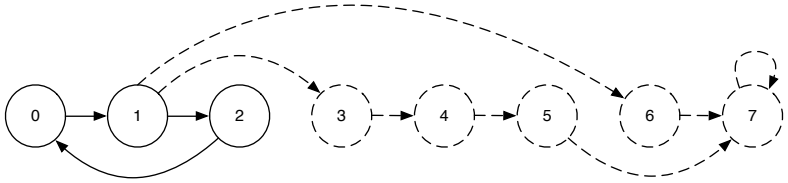

Fig. 2. Never 7 program.

For the sake of simplicity, we assume that we only have boolean variables in our programs; it is straightforward to extend this programming language with other programming types. Note that, in the kinds of programs described below, we may have two actions enabled at the same time; if this happens infinitely often during the execution of a system, we may have scenarios where some actions are neglected infinitely often. To avoid such scenarios, we must introduce the notion of *fair executions*. In order to express this fairness assumption, we follow the ideas introduced in [ABK04], where Kripke structures are augmented with fairness conditions. The authors consider a *transition fairness* defined as follow: "*A path π is fair with respect to the transition fairness condition iff all the transitions that are enabled along π infinitely often are also taken along π infinitely often*".

Definition 2.1 (Set of Fair Executions). Given a colored Kripke structure $M = \langle S, I, R, L, \mathcal{N} \rangle$, we define the set of fair executions of M as follows:

$$\mathcal{FT}(M) = \{\sigma \mid \sigma \in \mathcal{TR}(M) \text{ and } \forall w, t \in S : \forall i : \exists j > i : (t \in Post(\sigma[j]) \wedge \sigma[j] = w) \\ \Rightarrow \forall i : \exists j > i : \sigma[j] = t \wedge \sigma[j-1] = w\}$$

We say that a transition $w \rightarrow t$ is *enabled* in position i in σ, if $\sigma[i] = w$. Definition 2.1 says that actions that are enabled infinitely often are executed infinitely often. In practice, fair programs can be implemented by using schedulers, and it is a standard assumption in concurrency. It is worth noting that in our definition of fair executions we also consider faulty actions, that is, the faulty actions that are enabled infinitely often in an execution, will occur infinitely often. We can also introduce fair normative executions which do not take into account faults, as follows:

Definition 2.2 (Set of Fair Normative Executions). Given a colored Kripke structure $M = \langle S, I, R, L, \mathcal{N} \rangle$, we define the set of fair normative executions of M as follows:

$$\mathcal{FNT}(M) = \{\sigma \mid \sigma \in \mathcal{TR}(M) \text{ and } \forall w, t \in S : \forall i : \exists j > i : (t \in Post_N(\sigma[j]) \ \wedge \sigma[j] = w) \\ \Rightarrow \forall i : \exists j > i : \sigma[j] = t \wedge \sigma[j-1] = w\}$$

When useful, we denote the set of fair (resp. fair normative) executions starting in state s by $\mathcal{FT}(M)(s)$ (resp. $\mathcal{FNT}(M)(s)$). It is worth remarking that our definition of fair (normative) executions is the same as

that given in [ABK04] when the sets of (normative) states is considered as the fairness condition (a set that indicate those states that must be taken into account during the execution.)

The following properties will be useful in the next sections and can be proven by resorting to the properties of fair executions given in [ABK04].

Property 2.1. Given a colored Kripke structure $M = \langle S, I, R, L, \mathcal{N} \rangle$ we have $\mathcal{FT}(M)(s) \neq \emptyset$, for any $s \in S$.

Proof. The proof lies in the observation that M can be considered as a fair structure (as defined in [ABK04]) with $\alpha = S$ (the fairness condition). Let $s \in S$ be any state, by Lemma 2 given in [ABK04] (page 7) we have that any finite path $s_0 s_1 \dots s_k$ can be extended to a fair path, since s is a finite path of length 1 it can be extended to a fair path and so $\mathcal{FT}(M)(s) \neq \emptyset$. □

A similar property is true for normative fairness.

Property 2.2. Given a colored Kripke structure $M = \langle \langle S, I, R, L, \mathcal{N} \rangle$ we have $\mathcal{FNT}(M)(s) \neq \emptyset$, for any $s \in \mathcal{N}$.

Proof. The proof is similar to above. Let M' be the structure obtained by removing all the states not belonging to $\mathcal{N}$ from M (and the corresponding arcs), and $s \in \mathcal{N}$ note that $\mathcal{FNT}(M) = \mathcal{FNT}(M')$, now we can consider M' as a fair structure with $\alpha = \mathcal{N}$ and so any finite path can be extended to a fair path (Lemma 2 in [ABK04]), and since s is a finite path we have $\mathcal{FNT}(M')(s) \neq \emptyset$ and then $\mathcal{FNT}(M)(s) \neq \emptyset$. □

Note that the restriction to fair executions is reasonable when one inspects the way in which faults are distributed in practice. If a fault has a positive probability of occurring (if it has probability 0, it can be deleted from the model), then during an infinite execution it will occur infinitely often. However, in the case that one wants to restrict the number of occurrences of any fault, the program and the specification of the faults can be modified straightforwardly to do so. The restriction of $\models$ to fair (normative) executions is denoted by $\models_f$ ($\models_{nf}$).

A brief discussion about safety and liveness properties is necessary to cope with the rest of the paper. We define CTL *safety* formulas as those that only contain A (or E) and $\mathcal{W}$ temporal operators, $\vee$ and $\wedge$ operators and propositional variables or their negation. These subset of formulas are described by the following BNF:

$$\Phi_{safe} \quad ::= \quad \mathsf{A}(*p_i \; \mathcal{W} \; *q_i) \mid \mathsf{E}(*p_i \; \mathcal{W} \; *q_i) \mid \Phi_{safe} \wedge \Phi_{safe} \mid \Phi_{safe} \vee \Phi_{safe}$$

and $*p_i$ is p_i or $\neg p_i$. These formulas define (a subset of) safety properties as introduced for branching time in [MT01], where branching time properties are captured as sets of trees. Let us note that, in particular, these formulas define stuttering insensitive safety properties, as defined originally by Lamport in [Lam85]. We show that failsafe simulation (see Section 3) preserves CTL safety properties. An interesting subset of safety formulas are *invariants* which are safety formulas of the form $\mathsf{AG}\varphi$ (being φ a formula without temporal operators). Roughly speaking, invariants capture state properties that hold in every instant during the execution of the system.

On the other hand, we will also define *existential eventuality formulas* as those that only contain the EF temporal operator, $\vee$ and $\wedge$ boolean operators, and propositional variables or their negation. These subset of formulas are described by the following BNF:

$$\Phi_{live} \quad ::= \quad \mathsf{EF}(*p_i) \mid \Phi_{live} \wedge \Phi_{live} \mid \Phi_{live} \vee \Phi_{live}$$

Note that these formulas define (a subset of) liveness properties (as formally defined in [MT01]). In Section 3, we show that non masking simulation preserves existential eventuality formulas; and, furthermore, we show that, if the number of faults observed in the execution of a program is finite, then we can eventually establish any property of the specification, that is, $\mathsf{EF}\varphi$ becomes true, where φ is a property of the specification; this agrees with the standard definition of nonmasking in the related literature [AG93, AAE04, Gär99].

3. Simulations and Fault-Tolerance

In this section we present a number of simulation relations that allow us to capture various levels of fault-tolerance that are common in practice, namely *masking*, *nonmasking*, and *failsafe*. In order to define these

relations, we follow the basic definitions regarding simulation and bisimulation relations given, for instance, in [BK08]. It is worth remarking that the relations presented below, though having similar characteristics to standard simulation and bisimulation relations, are different to these kinds of relations, for instance, masking/nonmasking/failsafe relations are not (necessarily) symmetric, a standard property of bisimulation relations; but they become symmetric when the fault-tolerant version of the system does not exhibit faulty behavior, in contrast to simulation relations which are not symmetric. We discuss this below.

We assume that the system properties we are interested on can be captured by means of a set of safety and liveness properties (as defined in Section 2). Basically, in order to check fault-tolerance, we consider two colored Kripke structures for a system, the first one acting as a specification of the intended behavior and the second as the fault-tolerant implementation. A system will be fault-tolerant if it is able to preserve, to some degree, the safety and liveness properties corresponding to its specification, even in the presence of faults. Our main goal is to capture, via appropriate simulation relations between the system specification and the fault-tolerant implementation, different kinds of fault-tolerance, with different levels of property preservation.

In the following definitions, given a colored Kripke structure with a labeling L, we consider the notion of a sub-labeling: we say that L_0 is a sub-labeling of L (denoted by $L_0 \subseteq L$), if $L_0(s) = L(s) \cap AP'$, for all states s and some $AP' \subseteq AP$. We also say that L_0 is obtained by restricting AP to AP'. The concept of sub-labeling allows us to focus on certain properties of models.

3.1. Masking Fault-Tolerance

Recall that a program is said to be masking tolerant when it continues satisfying its specification even under the occurrence of faults [Gär99]. A minor observation about this definition is useful. Usually, when verifying a component, a piece of software or a module, one is interested in the behavior that is observable through its interface (as understood usually in software engineering [GJM03]); thus, when defining masking tolerance we restrict ourselves to the interface (that is, the visible part) of the component, captured formally by means of the notion of sub-labeling. Let us introduce the notion of masking tolerance simulation.

Definition 3.1. (Masking fault-tolerance) Given two colored Kripke structures $M = \langle S, I, R, L, \mathcal{N} \rangle$ and $M' = \langle S', I', R', L', \mathcal{N}' \rangle$, we say that a relationship $\mathbf{M} \subseteq S \times S'$ is *masking fault-tolerant* for sub-labelings $L_0 \subseteq L$ and $L'_0 \subseteq L'$ iff:

(A) $\forall s_1 \in I : (\exists s_2 \in I' : s_1 \,\mathbf{M}\, s_2)$ and $\forall s_2 \in I' : (\exists s_1 \in I : s_1 \,\mathbf{M}\, s_2)$.

(B) for all $s_1 \,\mathbf{M}\, s_2$ the following holds:

(1) $L_0(s_1) = L'_0(s_2)$;

(2) if $s'_1 \in Post_N(s_1)$, then there exists $s'_2 \in Post(s_2)$ with $s'_1 \,\mathbf{M}\, s'_2$;

(3) if $s'_2 \in Post_N(s_2)$, then there exists $s'_1 \in Post_N(s_1)$ with $s'_1 \,\mathbf{M}\, s'_2$;

(4) if $s'_2 \in Post_F(s_2)$, then either there exists $s'_1 \in Post_N(s_1)$ with $s'_1 \,\mathbf{M}\, s'_2$ or $s_1 \,\mathbf{M}\, s'_2$.

When sub-labelings L_0 and L'_0 are obtained by restricting L and L' to a vocabulary AP' we just say that $\mathbf{M}$ is defined over AP'.

We say that state s_2 is masking fault-tolerant for s_1 when $s_1 \,\mathbf{M}\, s_2$. Intuitively, the intention in the definition is that, starting in s_2, faults can be masked in such a way that the behavior exhibited is the same as that observed when starting from s_1 and executing transitions without faults. Let us explain the above definition. First, note that conditions A, $B.1$, $B.2$ and $B.3$ imply that we have a bisimulation when M and M' do not exhibit faulty behavior. Condition $B.2$ says that the normal execution of M can be masked by an execution of M'; note that here we include the possibility of masking normative behavior of the specification with faulty behavior of the implementation; roughly speaking, the main idea here is that the implementation may use some technique (such as redundancy) to mask faults, in such a way that they are not visible to the user. On the other hand, condition $B.3$ says that the implementation does not add normal (non-faulty) behavior, while condition $B.4$ states that every outgoing faulty transition from s_2 either must be matched to an outgoing normal transition from s_1, or s'_2 is masking tolerant for s_1; this expresses the idea that faulty transitions from the second structure mimic a normal behavior of the first structure. Finally, it is worth

remarking that the condition symmetric to $(B.4)$ is not required, since we are only interested in the masking properties of M'.

Using the notion of masking simulations we can define a relation $\prec_{Masking}$ between colored Kripke structures.

Definition 3.2. Given two colored Kripke structures $M = \langle S, I, R, L, \mathcal{N} \rangle$ and $M' = \langle S', I', R', L', \mathcal{N}' \rangle$:

$M \prec_{Masking} M' \Leftrightarrow$ there is a masking simulation $\mathbf{M} \subseteq S \times S'$

If $M \prec_{Masking} M'$ we say that M' is a masking fault-tolerant implementation of M.

Notice that, if there exists a self-loop at state s'_2, then we can stay forever satisfying $s_1 \, \mathbf{M} \, s'_2$. In this case, fairness is an important assumption which allows us to ensure system progress. Another important remark is that an execution could be fair, but after a while all its transitions become faulty; in this case we say that the execution diverges by faults. To deal with these executions, we will require that from every state it has to be possible to reach another state where non-faulty transitions are enabled, that is, we always have the possibility in the future of executing a correct action.

Definition 3.3 (Fault divergence). We say that a model M **does not diverge by faults** when for every $s \in S$ there exists $s' \in S$ such that $s \Rightarrow^* s'$ and $Post_N(s') \neq \emptyset$. In this case we say that M is a NDF (non-divergent by faults) structure.

That is, a model diverges by faults when it can reach a state where all the actions that can be executed in the future are faulty. The assumption that a model does not diverge by faults is natural in fault-tolerance, where assumptions about the way that faults occur are needed to prove properties about systems. In the case of masking tolerance, which is one of the most benign forms of fault-tolerance, the hypothesis that normal actions are not always neglected by the model being analyzed is required, in particular, to ensure the preservation of liveness properties. Note that this condition can be checked with a depth-first search over the model. Other authors, for instance [AG93, AK98a, AK98b], require that only a finite number of faults should occur in any execution of the system in order to provide masking tolerance; note that this requirement is stronger than the absence of fault divergence.

Example 3.1. Let us consider a memory cell that stores a bit of information and supports reading and writing operations. A state in this system maintains the current value of the memory cell ($m = i$, for $i = 0, 1$), writing allows one to change this value, and reading returns the stored value. Obviously, in this system the result of a reading depends on the value stored in the cell. Thus, a property that one might associate with this model is that the value read from the cell coincides with that of the last writing performed in the system. A potential fault in this scenario occurs when a cell unexpectedly loses its charge, and its stored value turns into another one (e.g., it changes from 1 to 0 due to charge loss). A typical technique to deal with this situation is *redundancy*: use three memory bits instead of one. Writing operations are performed simultaneously on the three bits. Reading, on the other hand, returns the value that is repeated at least twice in the memory bits; this is known as *voting*, and the value read is written back to the three bits.

We take the following approach to model this system: each state is described by variables m and w, which record the value stored in the system (taking *voting* into account) and the last writing operation performed, respectively. First, note that variable w is only used to enable the verification of properties of the model, thus this variable will not be present in any implementation of the memory.

The state also maintains the values of the three bits that constitute the system, captured by boolean variables c_0, c_1 and c_2. For instance, in Figure 3, state s_0 contains the information 11/111, representing the state: $w = 1$, $m = 1$, $c_0 = 1$, $c_1 = 1$, and $c_2 = 1$.

Consider the colored Kripke structures M (left) and M' (right) depicted in Figure 3. M only contains normal transitions describing the expected ideal behavior (without taking into account faults). M' includes a model of a fault: a bit may suffer a discharge and then it changes its value from 1 to 0.

We can show that in this simple case there exists a relation of masking fault-tolerance between M and M' with the sub-labelings L_0 and L'_0 obtained by restricting L and L' to variables m and w, respectively. The relation

$$\mathbf{M} = \{\langle s_0, t_0 \rangle, \langle s_1, t_1 \rangle, \langle s_0, t_2 \rangle\}$$

is masking fault-tolerant for $\langle M, M' \rangle$ and the sub-labelings L_0 and L'_0, obtained by restricting L and L' to variables m and w, respectively. Notice that each pair of $\mathbf{M}$ satisfies each condition of Definition 3.1:

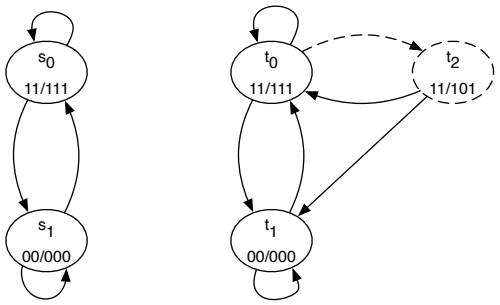

Fig. 3. Two masking fault-tolerance colored Kripke structures.

- $\langle s_0, t_0\rangle$ satisfies condition $B.1$ because $L_0(s_0) = L'_0(t_0)$. Conditions $B.2$ and $B.3$ are satisfied because the transition $s_0 \to s_0$ is masked by $t_0 \to t_0$ and vice versa with $\langle s_0, t_0\rangle \in \mathbf{M}$, and transition $t_0 \to t_1$ masks $s_0 \to s_1$ and vice versa with $\langle s_1, t_1\rangle \in \mathbf{M}$. Finally, condition $B.4$ is satisfied because the faulty transition $t_0 \dashrightarrow t_2$ masks the normal transition $s_0 \to s_0$ with $\langle s_0, t_2\rangle \in \mathbf{M}$.
- $\langle s_1, t_1\rangle$ satisfies condition $B.1$ because $L_0(s_1) = L'_0(t_1)$. Conditions $B.2$ and $B.3$ are satisfied because the normal transition $t_1 \to t_0$ is masked by $s_1 \to s_0$ and vice versa with $\langle s_0, t_0\rangle \in \mathbf{M}$, and the normal transition $t_1 \to t_1$ masks $s_1 \to s_1$ and vice versa with $\langle s_1, t_1\rangle \in \mathbf{M}$.
- $\langle s_0, t_2\rangle$ satisfies condition $B.1$ since $L_0(s_0) = L'_0(t_2)$ (taking into account that reading operations return the value that is repeated at least twice in the memory bits). Conditions $B.2$ and $B.3$ are satisfied because the normal transition $t_2 \to t_0$ masks $s_0 \to s_0$ and vice versa with $\langle s_0, t_0\rangle \in \mathbf{M}$, and the normal transition $t_2 \to t_1$ masks $s_0 \to s_1$ and vice versa with $\langle s_1, t_1\rangle \in \mathbf{M}$.

Now, we provide some results that allow us to apply the definition of masking fault-tolerance to paths.

Lemma 3.1. Let $M = \langle S, I, R, L, \mathcal{N}\rangle$ and $M' = \langle S', I', R', L', \mathcal{N}'\rangle$ be colored Kripke structures, $\mathbf{M} \subseteq S \times S'$ a masking relation over a vocabulary AP', and $s \in S, s' \in S'$ states such that $s\mathbf{M}s'$. If M' is a NDF structure, then for any path $\sigma' \in \mathcal{FT}(M')(s')$, there exists a function $f : \mathbb{N} \to \mathbb{N}$ and a normative path $\sigma \in \mathcal{FNT}(M)(s)$ such that:

- $\sigma[f(i)]\, \mathbf{M}\, \sigma'[i]$,
- f preserves initial segments in $\mathbb{N}$, that is, $f(\{i \mid 0 \le i \le k\}) = \{i \mid 0 \le i \le f(k)\}$[1].

Proof. Note that a path in S is just a function $\sigma : \mathbb{N} \to S$ that respects the transitions in the structure. Let us introduce some auxiliary notions. Given a state s' in S', we define $match(s, s') = \{w \mid w \in Post_N(s) \wedge w\, \mathbf{M}\, s'\}$, that is, the set of normative successors of s that are in relation $\mathbf{M}$ with state s'. Note that this set could be empty or could contain many states. Moreover, given a finite sequence of states $t_0 \dots t_n$ we denote by $num(t_0 \dots t_n, s_i)$ the number of occurrences of state s_i in $t_0 \dots t_n$.

Given $\sigma' \in \mathcal{FT}(M')$, let us define a trace $\sigma^* : \mathbb{N} \to S \cup \{*\}$ where $*$ is a new state, as follows:

$$\sigma^*[i] = \begin{cases} s & \text{if } i = 0, \\ w & \text{for some } w \in match(\sigma^*[\![i]\!], \sigma'[i]) \text{ and} \\ & \forall w' \in match(\sigma^*[\![i]\!], \sigma'[i]) : num(\sigma^*[0, i-1], w) \le num(\sigma^*[0, i-1], w'), \\ * & \text{otherwise,} \end{cases}$$

[1]Note that, given a function $f : A \to B$ and subset $S \subseteq A$, we use f(S) to denote the image of S under f, i.e.: $f(S) = \{y \in B \mid \exists x \in S : f(x) = y\}$.

where $\sigma^*[\![i]\!] = \sigma^*[last(i, \sigma^*)]$, and $last(i, \sigma^*) = max\{j \mid 0 \leq j < i : \sigma^*(j) \neq *\}$, that is, the last position in σ^* that is less than i and contains an element different from $*$.

Intuitively, each state in σ^* different from $*$ simulates a state in S', while $*$ states denote stuttering steps. Moreover, note in the definition of σ^* when choosing some $w\,\mathbf{M}\,\sigma'[i]$; we select the state with the minimum number of occurrences in $\sigma^*[0..i-1]$. This ensures the fairness of the execution, since no state w satisfying that condition will be neglected forever.

An interesting property about σ^* is the following:

$$\forall i \geq 0 : \exists j > i : \sigma^*[j] \neq * \tag{1}$$

That is, σ^* does no contain the infinite string: $****\ldots$. Another interesting property of σ^* is the following:

$$\forall i : \sigma^*[\![i+1]\!]\,\mathbf{M}\,\sigma'[i] \tag{2}$$

A detailed proof of both properties can be found in the appendix.

Now, we define function f as follows:

$$f(i) \stackrel{\mathsf{def}}{=} \#\{k \mid \sigma^*[k] \neq * \wedge 0 < k \leq i\}$$

That is, $f(i)$ counts the number of positions less than or equal to i (starting from 1) in which σ^* does not contain a $*$. Note that f is surjective: given any $n \in \mathbb{N}$, consider the n-th occurrence of some $s \neq *$ in σ^* (by (1) there is such an occurrence), let be k the index of such an occurrence, then $f(k) = n$.

σ is obtained by removing all the $*$'s from σ^*. To do this we use an auxiliary function $g(i)$, defined as follows:

$$\forall i \geq 0 : g(i) \stackrel{\mathsf{def}}{=} min\{k \mid f(k) = i\}$$

This is well defined since f is surjective. Some interesting properties of these functions are the following:

$$\forall i \geq 0 : g(i) = last(g(i+1), \sigma^*) \tag{3}$$

and

$$\forall i \geq 0 : \sigma^*[g(f(i))] = \sigma^*[\![i+1]\!] \tag{4}$$

A detailed proof of these properties is shown in the Appendix. Now, we define the normative trace σ as follows:

$$\forall i \geq 0 : \sigma[i] = \sigma^*[g(i)] \tag{5}$$

That is, for computing $\sigma[i]$ we take σ^* and find the position of the ith symbol in σ^* different from $*$.

Using these definitions we can prove that $\sigma[f(i)]\,\mathbf{M}\,\sigma'[i]$ as follows:

$$\begin{aligned} \sigma[f(i)] &= \sigma^*[g(f(i))] && \text{(def. of } \sigma) \\ &= \sigma^*[\![i+1]\!] && \text{(property above)} \\ &\mathbf{M}\,\sigma[i] && \text{(property above)} \end{aligned}$$

It remains to prove that $\sigma \in \mathcal{FNT}(M)$. First, let us prove that $\sigma \in \mathcal{NT}(M)$; that is, for every i, $\sigma[i+1] \in Post_N(\sigma[i])$. By definition of σ, this is equivalent to: $\sigma^*[g(i+1)] \in Post_N(\sigma^*[g(i)])$, but by (3) this is the same as: $\sigma^*[g(i+1)] \in Post_N(last[g(i+1)], \sigma^*)$ and that holds by the definitions of *match* and σ^*.

Now, we prove that σ is a fair execution. The proof is by contradiction, suppose that for some w and t we have that:

$$\forall i : \exists j > i : \sigma[j] = w \wedge t \in Post_N(w) \wedge \exists i \geq 0 : \forall j > i : \sigma[j] = w \Rightarrow \sigma[j+1] \neq t, \tag{6}$$

and let be $j_1 < j_2 < j_3 \ldots$ the positions that $\sigma[j_k] = w$. Since f is surjective, we have $i_1, i_2, \ldots$ such that $f(i_k) = j_k$, for every k, and since $\sigma[f(i_k)]\,\mathbf{M}\,\sigma'[i_k]$ (as proven above), we have that $\sigma[j_k]\,\mathbf{M}\,\sigma'[i_k]$. Furthermore, $t \in Post_N(\sigma[j_k])$ for every j_k. Since condition $B.2$ we have a set $\{t'_k \mid k \geq 0\}$ such that $t'_k \in Post_N(\sigma'[i_k])$ and $t\,\mathbf{M}\,t'_k$. Note that the number of states in S' is finite, so we must have some state (say t') that occurs infinitely often in $\{t'_k \mid k \geq 0\}$. Also, σ' is a fair execution, then we must have $\sigma'[i_k + 1] = t'$ for an infinite number of i_k's (by the fair successor property), that is, $t \in match(\sigma[j_k], \sigma'[i_k])$, which, by (3) and definition of σ, implies that $t \in match(\sigma^*[last(j_k + 1, \sigma^*)], \sigma'[i_k])$. Now, by the assumption above,

we have some i such that $\forall j > i : \sigma[j] = w \Rightarrow \sigma[j+1] \neq t$, but note that for a j enough large this leads to a contradiction. Since in the definition of σ^* when choosing $\sigma^*[j_k+1]$ (for $j_k > j$) we choose the state in $match(\sigma^*[last(j_k+2, \sigma^*)], \sigma'[i_k])$ occurring the minimum number of times in $\sigma^*[0..j_k]$, since t has been neglected since position i, it must be the state in $match(\sigma^*[last(j_k+1, \sigma^*)], \sigma'[i_k])$ that occurs the minimum number of times in the fragment $\sigma^*[0..j_k]$, and so it will occur at position j_k+1 in σ^*, contradicting (6).

Finally, we prove that f preserves initial segments. That is, for any initial segment $\{x \mid 0 \leq x \leq i\}$ in $\mathbb{N}$, $f(\{x \mid 0 \leq x \leq i\}) = \{y \mid 0 \leq y \leq f(i)\}$. First, note that f is monotone (a detailed proof is given in the Appendix) and $f(0) = 0$. Take $k \in f(\{x \mid 0 \leq x \leq i\})$, then $k = f(j)$ for some $j \leq i$, and by monotonicity we have, $f(j) \leq f(i)$ and so $k \in \{y \mid 0 \leq y \leq f(i)\}$. Now, take $k' \in \{y \mid 0 \leq y \leq f(i)\}$, that is, $k' \leq f(i)$. By surjectivity of f, we have $f(k) = k'$ for some k, and $k \leq i$ (otherwise we have a contradiction), and then $k' \in f(\{x \mid 0 \leq x \leq i\})$. Thus both sets are equal. □

Note that, if we only consider normative executions starting in s_1 and s_2 with $s_1 \,\mathbf{M}\, s_2$, by conditions $(B.3)$ and $(B.4)$ of Definition 3.1 we have that, for each normative path starting in s_2, there exists a corresponding path from s_1, where its states are similar by a masking relation. This is proven in the following lemma.

Lemma 3.2. Let $M = \langle S, I, R, L, \mathcal{N}\rangle$ and $M' = \langle S', I', R', L', \mathcal{N}'\rangle$ be colored Kripke structures, $\mathbf{M} \subseteq S \times S'$ a masking relation over a vocabulary AP', and $s \in S, s' \in S'$ states such that $s \,\mathbf{M}\, s'$. Then there exist a function $f : \mathcal{FNT}(M)(s) \to \mathcal{FT}(M')(s')$ such that:

- $\forall \sigma \in \mathcal{FNT}(M)(s) : \forall i \geq 0 : \sigma[i] \,\mathbf{M}\, f(\sigma)[i]$,

Proof. Given $\sigma \in \mathcal{FNT}(M)(s)$ s.t. $\sigma = ss_1s_2\ldots$, then note that we have $s \,\mathbf{M}\, s'$ and by $(B.2)$, if $s_i \,\mathbf{M}\, s'_i$ and $s_{i+1} \in Post_N(s_i)$, then there exists $s'_{i+1} \in Post(s'_i)$ such that $s_{i+1} \,\mathbf{M}\, s'_{i+1}$. Thus, we can define inductively a sequence $f(\sigma) = s's'_1s'_2\ldots$ that satisfies $\sigma[j] \,\mathbf{M}\, f(\sigma)[j]$ for every j. To ensure the fairness of such a construction, when choosing the successor of a given $f(\sigma)[i]$ where $\sigma[i] \,\mathbf{M}\, f(\sigma)[i]$, we select the successor $t \in Post(f(\sigma)[i])$ such that $\sigma[i+1] \mathbf{M} t$ that appears the minimum number of times in $f(\sigma)[0..i]$, which exists by condition $(B.2)$. The result follows. □

Now, we can prove that, in the case of masking simulation, the liveness and safety properties of the normal behavior of the specification are preserved by the implementation. However, note that not all the temporal properties are preserved; formulas with occurrences of the next operator may not be preserved by the implementation. Roughly speaking, this is because condition $(B.4)$ allows implementations to advance in time while staying in the same state on the specification side.

Theorem 3.3. Let $M = \langle S, I, R, L, \mathcal{N}\rangle$ and $M' = \langle S', I', R', L', \mathcal{N}'\rangle$ be colored Kripke structures, $\mathbf{M} \subseteq S \times S'$ a masking relation over a vocabulary AP', and states $s_1 \in S, s_2 \in S'$ be states such that $s \,\mathbf{M}\, s'$. Then:

$$M, s \vDash_{nf} \varphi \Leftrightarrow M', s' \vDash_f \varphi,$$

where φ is a CTL-X formula where all the propositional variables of φ are in AP'.

Proof. We proceed by induction over the structure of the formula φ.

- Base Case: $\varphi = p_i$. From $s \,\mathbf{M}\, s'$ it follows by condition $(B.1)$ for masking fault-tolerance that s and s' have the same propositional valuation. Thus, $M, s \vDash_{nf} p_i \Leftrightarrow M', s' \vDash_f p_i$.
- Inductive Case: we prove the property for operators $\wedge, \neg, \mathsf{E}\mathcal{U}, \mathsf{EG}$ since the other ones can be expressed by means of them.

  Case 1: For $\varphi = \psi \wedge \psi'$ and $\varphi = \neg\psi$ the result follows by a direct application of the inductive hypothesis.

  Case 2: $\varphi = \mathsf{E}(\psi_1 \,\mathcal{U}\, \psi_2)$. Let us prove the right implication, suppose that $M, s \vDash_{nf} (\mathsf{E}(\psi_1 \,\mathcal{U}\, \psi_2))$, then for some path $\sigma \in \mathcal{FNT}(M)$ with $\sigma[0] = s$ we have that:

  $$\exists i \geq 0 : M, \sigma[i] \vDash_{nf} (\psi_2) \text{ and } \forall 0 \leq j < i : M, \sigma[j] \vDash_{nf} \psi_1$$

  Then by Lemma 3.2 we have a path $f(\sigma) \in \mathcal{FT}(M')$ with $f(\sigma)[0] = s'$ s.t. $\forall i \geq 0 : \sigma[i] \,\mathbf{M}\, f(\sigma)[i]$; this implies by induction that:

  $$\exists i \geq 0 : M', f(\sigma)[i] \vDash_f \psi_2 \text{ and } \forall 0 \leq j < i : M, f(\sigma)[j] \vDash_f \psi_1$$

  Then, $M', s' \vDash_f \mathsf{E}(\psi_1 \,\mathcal{U}\, \psi_2)$.

  For the other direction, we proceed as follows. Assume $M', s' \vDash_f \mathsf{E}(\psi_1 \,\mathcal{U}\, \psi_2)$, this means that

there is some $\sigma' \in \mathcal{FT}(s')$ such that $\exists i \geq 0 : M', s' \vDash_f \psi_2$ and $\forall j : 0 \leq j < i : M', \sigma' \vDash_f \psi_1$. Now, by Lemma 3.1, we have a $\sigma \in \mathcal{FNT}(s)$ and a function f such that $\sigma[f(i)]\, \mathbf{M}\, \sigma'[i]$, by induction this implies that $M, \sigma[f(i)] \vDash \psi_2$ and also $M, \sigma[f(j)] \vDash \psi_1$ for $0 \leq j < i$, but since f preserves initial segments of $\mathbb{N}$ we have that $\{f(0), f(1), \ldots, f(i)\} = \{j \mid 0 \leq j < f(i)\}$ and then $M, \sigma[j] \vDash_{nf} \psi_1$ for $0 \leq j < f(i)$ which is implies that: $M, \sigma[0] \vDash_{nf} E(\psi_1\, \mathcal{U}\, \psi_2)$.

Case 3: $\varphi = \mathsf{EG}\psi$. Let us prove the right implication. Suppose $M, s \vDash_{nf} \mathsf{EG}\psi$, this means that there is some $\sigma \in \mathcal{FNT}(s)$ $\forall i \geq 0 : M, \sigma[i] \vDash_{nf} \psi$. By Lemma 3.2 we have some $\sigma' \in \mathcal{FT}(M')(s')$ such that $\sigma[i]\, \mathbf{M}\, \sigma'[i]$, and so by induction the result follows.

For the other direction assume now that $M', s' \vDash_f \mathsf{EG}\psi$, this means that $\forall i \geq 0 : M', \sigma'[i] \vDash_f \psi$, by Lemma 3.1 we have a trace $\sigma \in \mathcal{FNT}(s)$ and a function f such that $\sigma[f(i)]\, \mathbf{M}\, \sigma[i]$, by induction we have that $M, \sigma[f(i)] \vDash \psi$, but since f is surjective this implies that $M, \sigma[i] \vDash_{nf} \psi$ for every i, thus $M, s \vDash_{nf} \mathsf{EG}\psi$.

□

Summing up, in the case of masking simulation, the basic temporal properties of systems without faults, such as invariants or liveness formulas, are preserved by masking tolerant implementations.

3.2. Nonmasking Fault-Tolerance

We now focus on nonmasking fault-tolerance. This kind of tolerance is more permissive than masking fault-tolerance; recall that it allows for the existence of some states that do not mask faults. Intuitively, this type of fault-tolerance allows the system to violate its specification while it is recovering from a fault and eventually returning to a normal behavior. The technical definition of nonmasking [Gär99] states that the liveness properties of the nonfaulty part of the system have to be preserved, whereas the safety properties observed in the correct behavior of the system may not be fully preserved. Furthermore, sometimes a stronger definition of nonmasking is adopted in which the safety properties may be violated but they should be *eventually* reinstated [AAE04, Gär99], which can be captured in CTL with formulas $\mathsf{AF}\varphi$ or $\mathsf{EF}\varphi$ (being φ the safety and liveness properties to be preserved) here we will be interested in the latter kinds of formulas, which intuitively state that the system has a way of recovering from faults, stronger versions of the notion of nonmasking given above can be considered (guaranteeing that all the execution recover from faults), but they are much more complex to capture (and to algorithmically verify) since quantification over paths is needed for formally defining them.

Our characterization of nonmasking fault-tolerance is as follows.

Definition 3.4. (Nonmasking fault-tolerance) Given two colored Kripke structures $M = \langle S, I, R, L, \mathcal{N}\rangle$ and $M' = \langle S', I', R', L', \mathcal{N}'\rangle$, we say that a relation $\mathbf{N} \subseteq S \times S'$ is nonmasking for sub-labelings $L_0 \subseteq L$ and $L'_0 \subseteq L'$, iff:

(A) $\forall s_1 \in I : (\exists s_2 \in I' : s_1\, \mathbf{N}\, s_2)$ and $\forall s_2 \in I' : (\exists s_1 \in I : s_1\, \mathbf{N}\, s_2)$.

(B) for all $s_1\, \mathbf{N}\, s_2$ the following holds:

(1) $L_0(s_1) = L'_0(s_2)$;

(2) if $s'_1 \in Post_N(s_1)$, then there exists $s'_2 \in Post(s_2)$ with $s'_1\, \mathbf{N}\, s'_2$;

(3) if $s'_2 \in Post_N(s_2)$, then there exists $s'_1 \in Post_N(s_1)$ with $s'_1\, \mathbf{N}\, s'_2$;

(4) if $s'_2 \in Post_F(s_2)$, then there exists a $s'_1 \in Post_N(s_1)$ and a $s''_2 \in Post^*(s'_2)$ such that $s'_1\, \mathbf{N}\, s''_2$.

As before, when sub-labelings L_0 and L'_0 are obtained by restricting L_0 and L'_0 to a vocabulary AP' we just say that $\mathbf{N}$ is defined over AP'.

Let us briefly explain this definition. Conditions A, $B.1$, $B.2$ and $B.3$ are the same as the conditions of Definition 3.1. Condition $B.4$ asserts that, if $s_1\, \mathbf{N}\, s_2$ and every "faulty" successor state (say s'_2) of s_2 is not in a nonmasking relation with any normal successor of s_1, then any faulty path fragment starting at s'_2 can be extended to reach a s''_2 such that $s'_1\, \mathbf{N}\, s''_2$ for some normal successor s'_1 of s_1; that is, the system can recover from faults.

We define the relation $\prec_{Nonmask}$ between colored Kripke structures.

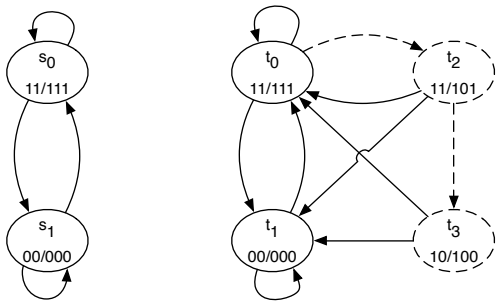

Fig. 4. Two nonmasking fault-tolerance colored Kripke structures.

Definition 3.5. Given two colored Kripke structures $M = \langle S, I, R, L, \mathcal{N}\rangle$ and $M' = \langle S', I', R', L', \mathcal{N}'\rangle$:

$M \prec_{Nonmask} M' \Leftrightarrow$ there is a nonmasking simulation $\mathbf{N} \subseteq S \times S'$

If $M \prec_{Nonmask} M'$ we say that M' is a nonmasking fault-tolerant implementation of M.

At first sight, nonmasking fault-tolerance seems similar to the notion of weak bisimulation used in process algebra [Mil89], where silent steps are taken into account. Notice, however, that, as opposed to weak bisimulation where silent steps produce only non-observable (i.e., internal) changes, faults may produce observable changes in a nonmasking fault-tolerance relation. Let us present an example of nonmasking tolerance.

Example 3.2. For the memory cell introduced in Example 3.1, consider now the colored Kripke structures M (left) and M' (right) depicted in Figure 4. Now we consider that two faults may occur: up to two bits may lose their charge before any normal transition is taken. The relation $\mathbf{N} = \{\langle s_0, t_0\rangle, \langle s_1, t_1\rangle, \langle s_0, t_2\rangle\}$ is nonmasking tolerant for $\langle M, M'\rangle$ and the sub-labelings L_0 and L'_0, obtained by restricting L and L' to variables m and w, respectively.

As discussed above, in nonmasking fault-tolerance, one is interested in preserving the liveness properties of the non-faulty part of the system; for instance, the requests need to be granted, or the program should exhibit some advance towards a given goal, even during a faulty scenario. Note that, in this case, safety conditions do not need to be preserved. We prove that liveness properties defined by means of eventuality CTL formulas are preserved by nonmasking simulation. First, we prove a lemma about normative and faulty executions of two systems related by a nonmasking relation.

Lemma 3.4. Given $M = \langle S, I, R, L, \mathcal{N}\rangle$ and $M' = \langle S', I', R', L', \mathcal{N}'\rangle$, $\mathbf{N} \subseteq S \times S'$ a nonmasking relation over a vocabulary AP', and states $s \in S, s' \in S'$ with $s\,\mathbf{N}\,s'$. If there is a $\sigma \in \mathcal{FNT}(M)(s)$, such that $M, \sigma \vDash \mathsf{F}{*}p$, for some p in AP' (where $*$ is $\neg$ or blank), then there is a $\sigma' \in \mathcal{FT}(M')(s')$ such that $M', \sigma' \vDash \mathsf{F}{*}p$.

Proof. Given $\sigma \in \mathcal{FNT}(M)(s)$, we can define an execution $\sigma' \in \mathcal{FT}(M')$ by induction. We define $\sigma'[0] = s'$, and $\sigma'[i+1] = w$, for some $w \in Post(\sigma'[i])$ such that $\sigma[i+1]\,\mathbf{N}\,w$, which exists by condition $B.2$. To ensure the fairness of σ when choosing w we select that state that occurs the minimum number of times in $\sigma[0..i-1]$, that is, no such a w will be neglected forever.

Now, since $M, \sigma \vDash \mathsf{F} * p$, we have some $k \geq 0$ such that $M, \sigma[k] \vDash *p$, and then by condition $B.1$ of Definition 3.4 and taking into account that $\sigma[k]\,\mathbf{N}\,\sigma'[k]$, we have that $M, \sigma'[k] \vDash *p$. Thus, $M, \sigma' \vDash \mathsf{F} * p$. □

Now, we can prove that, in the presence of fairness, existential eventuality formulas are preserved by nonmasking implementations:

Theorem 3.5. Let $M = \langle S, I, R, L, \mathcal{N}\rangle$ and $M' = \langle S', I', R', L', \mathcal{N}'\rangle$ be colored Kripke structures, $\mathbf{N} \subseteq S \times S'$ a nonmasking relation over a vocabulary AP', and states $s \in S, s' \in S'$ with $s\,\mathbf{N}\,s'$. Then

$$M, s \vDash_{nf} \varphi \Rightarrow M', s' \vDash_{f} \varphi,$$

where φ is an existential eventuality CTL formula and all the propositional variables of φ are in AP'.

Proof. The proof is by induction on φ. For the base case, we have :

- If $\varphi = \mathsf{EF}p$, suppose $M, s \vDash_{nf} \varphi$, that is, there is an execution $\sigma \in \mathcal{FNT}(M)$ such that $M, \sigma[i] \vDash p$, for some i, thus by Lemma 3.4, we have a $\sigma' \in \mathcal{FT}(M')$ such that $M', \sigma'[k] \vDash p$ for some k.

The inductive cases (that is, $\psi_1 \vee \psi_2$ and $\psi_1 \wedge \psi_2$) are direct using the inductive hypothesis. □

Note that this theorem guarantees that implementations preserve the existential liveness properties of specifications. Furthermore, notice that the other direction of this property is not necessarily true. This is mainly because nonmasking implementations may eventually make true some properties, during its faulty behavior, which do not hold in the non-faulty program.

As argued in [AK98b, Gär99], in practice we are interested in those nonmasking programs that have recovery executions to eventually reestablish the safety properties of their specifications, that is, faulty programs may exhibit an incorrect behavior, but at some point they may execute a set of actions to start behaving as expected. Obviously, to guarantee such a property in a nonmasking simulation, we need to avoid scenarios where faults occur in such a way that the system cannot reach a normal execution. In the following theorem, we prove that, when all the executions of the system only exhibit a finite number of faults, then there is an execution that reestablishes the invariants of the system.

Theorem 3.6. Let $M = \langle S, I, R, L, \mathcal{N}\rangle$ and $M' = \langle S', I', R', L', \mathcal{N}'\rangle$ be colored Kripke structures, $\mathbf{N} \subseteq S \times S'$ a nonmasking relation over a vocabulary AP', and states $s \in S, s' \in S'$ with $s\,\mathbf{N}\,s'$. If for every $\sigma' \in \mathcal{FT}(M')(s')$ the number of i's such that $\sigma'[i+1] \in Post_F(\sigma'[i])$ is less than k (for some fixed k), then

$$M, s \vDash_n (\mathsf{AG}\varphi) \Rightarrow M', s' \vDash \mathsf{EFAG}\varphi,$$

where $\mathsf{AG}\varphi$ is an invariant (as defined in Section 2) and all the propositional variables of φ are in AP'.

Proof. To prove this property some observations are useful. First, note that, since we have finite structures, requiring that the number of faults in any execution is bounded, it is equivalent to requiring that no faults occur in any cycle, otherwise we can find a trace (the cycle) where we have an unbounded number of faults. That is, in this case, for any execution $\sigma' \in \mathcal{TR}(M')$, we have an instant i such that $\mathcal{TR}(M')(\sigma'[i]) \subseteq \mathcal{NT}(M')(\sigma'[i])$, that is, we have a point from which all the transitions are non-faulty. Note also that, if $s\,\mathbf{N}\,s'$ with $\mathcal{TR}(M)(s) \subseteq \mathcal{NT}(M)(s)$ and $\mathcal{TR}(M')(s') \subseteq \mathcal{NT}(M')(s')$, then we have that s and s' are bisimilar (there are no faulty actions, and $B.2$ and $B.3$ for Definition 3.4 guarantee a bisimulation) and thus $M, s \vDash \varphi$ iff $M', s' \vDash \varphi$ for any CTL formula φ.

Now, suppose that $M, s \vDash_n (\mathsf{AG}\varphi)$. Take the shortest finite path starting from s' that leads to a $s'_k \notin \mathcal{N}$, that is, we have a path: $s's'_0 \ldots s'_k$ such that $s'_k \in Post_F(s'_{k-1})$ (if there is no such a path the proof is direct since all the paths are non-faulty and there is a bisimulation between s and s'), since $s'_i \in Post_N(s'_{i-1})$ for $0 \leq i \leq k-1$ and by condition $B.3$ of Definition 3.4, we have a path in M: $ss_1 \ldots s_{k-1}$ such that $s_i\,\mathbf{N}\,s'_i$ for $0 \leq i \leq k-1$, and for s_k we have some $s'_{k+t} \in Post^*(s'_k)$ and a $s_k \in Post_N(s_{k-1})$ such that $s_k\,\mathbf{N}\,s'_{k+t}$, we can repeat (at most k times) this procedure until we get a state $s'_{k'}$ from which no more faulty states are reachable and so $\mathcal{TR}(M')(s'_{k'}) \subseteq \mathcal{NT}(M')(s'_{k'})$, and since we have $s_j\,\mathbf{N}\,s_{k'}$ where $s_j \in Post^*_N(s)$ and so $M, s_j \vDash_n \mathsf{AG}\varphi$, and by the observations above we have $M', s_{k'} \vDash \mathsf{AG}\varphi$, considering that $s_{k'} \in Post^*(s)$ we get $M', s' \vDash \mathsf{EFAG}\varphi$. □

Summarizing, this theorem says that, if we assume that the number of faults occurring in executions is finite and bounded by a fixed number, then invariants can be eventually reestablished by means of a recovery procedure. Let us note that the restriction to a finite number of faults during a system failure is a strong assumption about the execution of our system, one may devise scenarios where weaker assumptions guarantee the restoration of a subset of properties. We leave a deeper investigation about this as further work.

3.3. Failsafe Fault-Tolerance

We now present a characterization of failsafe fault-tolerance. Essentially, failsafe fault-tolerance must ensure that the system will stay in a safe state, although it may be limited in its capacity. More technically, this means that the normative safety properties must be preserved, while normative liveness properties may not be respected.

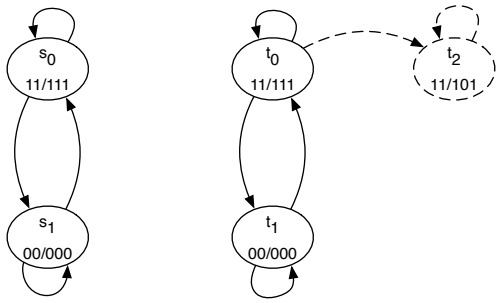

Fig. 5. Two failsafe fault-tolerance colored Kripke structures.

Definition 3.6. (Failsafe fault-tolerance) Given two colored Kripke structures $M = \langle S, I, R, L, \mathcal{N} \rangle$ and $M' = \langle S', I', R', L', \mathcal{N}' \rangle$, we say that a relation $\mathbf{F} \subseteq S \times S'$ is failsafe for sub-labelings $L_0 \subseteq L$ and $L'_0 \subseteq L'$ iff:

(A) $\forall s_1 \in I : (\exists s_2 \in I' : s_1 \,\mathbf{F}\, s_2)$ and $\forall s_2 \in I' : (\exists s_1 \in I : s_1 \,\mathbf{F}\, s_2)$.
(B) for all $s_1 \,\mathbf{F}\, s_2$ the following holds:
  (1) $L_0(s_1) = L'_0(s_2)$;
  (2) if $s'_1 \in Post_N(s_1)$, then there exists $s'_2 \in Post(s_2)$ with $s'_1 \,\mathbf{F}\, s'_2$;
  (3) if $s'_2 \in Post_N(s_2)$, then there exists $s'_1 \in Post_N(s_1)$ with $s'_1 \,\mathbf{F}\, s'_2$;
  (4) if $s'_2 \in Post_F(s_2)$, then either:
    i there exists $s'_1 \in Post_N(s_1)$ with $s'_1 \,\mathbf{F}\, s'_2$ or $s_1 \,\mathbf{F}\, s'_2$, or
    ii $\forall s : (s'_2 \Rightarrow^* s) \Rightarrow L'_0(s_2) = L'_0(s)$.

As in the definitions above, when convenient, instead of giving the sub-labelings, we just say that $\mathbf{F}$ is defined over AP'.

As before, we can define a relation of *failsafe* simulation between colored Kripke structures.

Definition 3.7. Given two colored Kripke structures $M = \langle S, I, R, L, \mathcal{N} \rangle$ and $M' = \langle S', I', R', L', \mathcal{N}' \rangle$:

$M \prec_{Failsafe} M' \Leftrightarrow$ there is a failsafe simulation $\mathbf{F} \subseteq S \times S'$

If $M \prec_{Failsafe} M'$ we say that M' is a failsafe fault-tolerant implementation of M.

Let us briefly explain Definition 3.6. Conditions A, $B.1$, $B.2$, $B.3$, $B.4.i$ are the same as those conditions of Definition 3.1 regarding masking fault-tolerance. Condition $B.4.ii$ intuitively asserts that, from s'_2 the system in M' moves to a set of safe states which are equivalent to s'_2 w.r.t. the interface. We now present a simple example to illustrate this notion.

Example 3.3. Consider the colored Kripke structures M (left) and M' (right) depicted in Figure 5. M is the specification of the expected ideal, fault-free, behavior. M', on the other hand, involves the occurrence of one fault. The relation $\mathbf{F} = \{\langle s_0, t_0 \rangle, \langle s_1, t_1 \rangle\}$ is a failsafe fault-tolerance relation for $\langle M, M' \rangle$ and the sub-labelings that are obtained by restricting L and L' to variables m and w.

In the following we prove that our definition of failsafe fault-tolerance preserves safety properties. First, note that a failsafe relation imposes a relationship between the traces of the two models involved in the relation; this is proven in the following lemmas.

Lemma 3.7. Let $M = \langle S, I, R, L, \mathcal{N} \rangle$ and $M' = \langle S', I', R', L', \mathcal{N}' \rangle$ be colored Kripke structures, $\mathbf{F} \subseteq S \times S'$ a failsafe relation over a vocabulary AP', and states $s \in S, s' \in S'$ with $s \,\mathbf{F}\, s'$. If there is a $\sigma' \in \mathcal{FT}(M')(s')$ with $M', \sigma' \vDash *p \,\mathcal{U}\, *q$ (where $*$ is $\neg$ or blank) for some $p, q \in AP'$, then there exists a $\sigma \in \mathcal{FNT}(M)(s)$ such that $M, \sigma \vDash *p \,\mathcal{U}\, *q$.

Proof. Given $\sigma' \in \mathcal{FT}(M')(s')$ and $s \,\mathbf{F}\, s'$, if $M', \sigma' \vDash *p \,\mathcal{U}\, *q$ then we have either:

(a) $M', s' \vDash *q$ or
(b) there is a $k > 0$ such that $M', \sigma'[k] \vDash *q$ and $M', \sigma'[j] \vDash *p$ for every $j < k$.

In the first case, since $s \,\mathbf{F}\, s'$ and condition $B.1$ of Definition 3.4 we obtain $M, s \vDash *q$ and so $M, \sigma \vDash *p \,\mathcal{U}\, *q$, for any $\sigma \in \mathcal{FNT}(M)(s)$, and by Property 2.1, $\mathcal{FNT}(M)(s) \neq \emptyset$ and then we have a $\sigma \in \mathcal{FNT}(M)(s)$ satisfying $M, \sigma \vDash *p \,\mathcal{U}\, *q$.

On the other hand, if (b) holds, then let us define a finite path $s, s_1, s_2, \ldots, s_{k'}$ in M such that $M, s_{k'} \vDash *q$ and $M, s_i \vDash *p$ for $0 \leq i < k'$. First, note that $M', s' \nvDash *q$ and $M', \sigma'[k] \vDash *q$, since $s' \Rightarrow^* \sigma'[k]$ we have that condition $B.4.ii$ does not hold in any path starting from s'; taking into account that $s \,\mathbf{F}\, s'$ we define a sequence $w = w_0, w_1, w_2, \ldots, w_k$ of states in S as follows:

$$w_i = \begin{cases} s & \text{if } i = 0, \\ w & \text{if there is } w \in Post(w_{i-1}) \text{ with } w \,\mathbf{F}\, s'_i, \\ w_{i-1} & \text{otherwise.} \end{cases}$$

Note that for the segment obtained: $w_0, w_1, \ldots, w_k$ we have $w_i \,\mathbf{F}\, s_i$; also note that in w we may have some repeated states, then let $w' = w'_0, w'_1, \ldots, w'_{k'}$ (with $k' \leq k$) the finite path obtained by removing from w all the repeated states that do not have self-loops in M. For this sequence of states w', we have $M, w'_{k'} \vDash *q$ (since $w'_{k'} \,\mathbf{F}\, w_k$) and $M, w'_i \vDash *p$ (since $w'_i \,\mathbf{F}\, s'_j$ for some $i < k'$ and $j < k$), and also $w'_{i+1} \in Post_N(w'_i)$, because repeated states were removed. Now, by Property 2.2, we have some $\sigma_0 \in \mathcal{FNT}(M)(w_{k'})$, and then we can form the path: $\sigma = w' {:} \sigma_0$ (w' concatenated σ_0), also note that $\sigma \in \mathcal{FNT}(M)(s)$ (it is fair since σ_0 is fair). Furthermore, we have $M, \sigma \vDash *p \,\mathcal{U}\, *q$, this finishes the proof. □

Lemma 3.8. Let $M = \langle S, I, R, L, \mathcal{N} \rangle$ and $M' = \langle S', I', R', L', \mathcal{N}' \rangle$ be colored Kripke structures, $\mathbf{F} \subseteq S \times S'$ a failsafe relation over a vocabulary AP', and states $s \in S, s' \in S'$ with $s \,\mathbf{F}\, s'$. If there is a $\sigma \in \mathcal{FNT}(M)(s)$ with $M, \sigma \vDash *p \,\mathcal{W}\, *q$ (where $*$ is $\neg$ or blank) for some $p, q \in AP'$, then there exists a $\sigma' \in \mathcal{FT}(M')(s')$ such that $M', \sigma' \vDash *p \,\mathcal{W}\, *q$.

Proof. Given $\sigma \in \mathcal{FNT}(M)(s)$ such that $M, \sigma \vDash *p \,\mathcal{W}\, *q$ and let $\sigma = s_0 s_1, \ldots,$ by definition of $\vDash$ we have either:

(a) $\exists j \geq 0 : M, \sigma[j] \vDash *q$ and $\forall 0 \leq i < j : M, \sigma[i] \vDash *p$, or
(b) $\forall j \geq 0 : M, \sigma[j] \vDash *p$.

by $B.2$ of Definition 3.6 we can define inductively an execution σ' in $\mathcal{FT}(M')(s')$, say $\sigma' = s'_0 s'_1 \ldots$, such that: $s_i \,\mathbf{F}\, s'_i$, by Definition 3.6 this implies that $L_0(s_i) = L'_0(s'_i)$ for every i. That is if (a) holds, then we have: $\exists j \geq 0 : M', \sigma'[j] \vDash *q$ and $\forall 0 \leq i < j : M', \sigma'[i] \vDash *p$; and if (b) holds we have: $\forall j \geq 0 : M', \sigma'[j] \vDash *p$; thus in any case we get: $M', \sigma' \vDash *p \,\mathcal{W}\, *q$. □

Let us now prove that failsafe implementations preserve safety properties.

Theorem 3.9. Let $M = \langle S, I, R, L, \mathcal{N} \rangle$ and $M' = \langle S', I', R', L', \mathcal{N}' \rangle$ be colored Kripke structures, $\mathbf{F} \subseteq S \times S'$ a failsafe relation over a vocabulary AP', and states $s_1 \in S, s_2 \in S'$ with $s_1 \,\mathbf{F}\, s_2$. Then:

$$M, s_1 \vDash_{nf} \varphi \Rightarrow M', s_2 \vDash_f \varphi$$

where φ is a CTL safety property and all the propositional variables of φ are in AP'.

Proof. We proceed by induction on φ. The base case is as follows:

- If $\varphi = \mathsf{A}(*p \,\mathcal{W}\, *q)$, then suppose $M, s \vDash_{nf} \mathsf{A}(*p \,\mathcal{W}\, *q)$ and not $M', s' \vDash_f \mathsf{A}(*p \,\mathcal{W}\, *q)$. Thus, we have a $\sigma' \in \mathcal{FT}(M')(s')$ such that $M', \sigma' \vDash (*p \wedge \neg *q) \,\mathcal{U}\, (\neg *p \wedge \neg *q)$, consider fresh variables s and t such that $s \equiv *p \wedge \neg *q$ and $t \equiv \neg *p \wedge \neg *q$, that is we have $M', \sigma' \vDash s \,\mathcal{U}\, t$. By Lemma 3.7 we get that for some $\sigma \in \mathcal{FNT}(M)(s)$ we have that $M, \sigma \vDash_{nf} s \,\mathcal{U}\, t$ and so $M, \sigma \vDash_{nf} (*p \wedge \neg *q) \,\mathcal{U}\, (\neg *p \wedge \neg *q)$, which contradicts our initial assumption.
- If $\varphi = \mathsf{E}(*p \,\mathcal{W}\, *q)$, then we have some path $\sigma \in \mathcal{FNT}(M)(s_1)$ such that $M, \sigma \vDash *p \,\mathcal{W}\, *q$, then by Lemma 3.8 we have a $\sigma' \in \mathcal{FT}(M')(s_2)$ such that $M', \sigma' \vDash *p \,\mathcal{W}\, *q$ and so $M', s_2 \vDash \mathsf{E}(*p \,\mathcal{W}\, *q)$.

The inductive cases are direct by using the inductive hypothesis. □

That is, if we have a failsafe relation between s_1 and s_2 and for every state in normal paths starting in s_1,

and a safety property φ holds in the absence of faults, then φ is always true even in the presence of faults in paths starting in s_2.

3.4. Some Properties

The following lemma presents some properties of the fault-tolerance relations defined above.

Lemma 3.10. Let $\sqsubset \in \{\prec_{Masking}, \prec_{Nonmask}, \prec_{Failsafe}\}$, then the following properties hold.

- $\sqsubset$ is transitive,
- If M and M' does not have faults, then: $M \sqsubset M' \Rightarrow M' \sqsubset M$,
- $\sqsubset$ is not necessarily reflexive.

Proof. We prove the properties for any masking relation $\prec_{Masking}$. The proofs for the other relations are similar.

- **Transitivity** Suppose that $M_1 \prec_{Masking} M_2$ and $M_2 \prec_{Masking} M_3$, for colored Kripke structures $M_1 = \langle S_1, I_1, R_1, L_1, \mathcal{N}_1\rangle$, $M_2 = \langle S_2, I_2, R_2, L_2, \mathcal{N}_2\rangle$, and $M_3 = \langle S_3, I_3, R_3, L_3, \mathcal{N}_3\rangle$, and sub-labellings $L_1' \subseteq L_1, L_2' \subseteq L_2$, and $L_3' \subseteq L_3$ with $L_1' = L_2' = L_3'$. By definition of $\prec_{Masking}$ this means that we have relations: $\mathbf{M}_{1,2} \subseteq S_1 \times S_2$ and $\mathbf{M}_{2,3} \subseteq S_2 \times S_3$ such that the relation $\mathbf{M}_{1,3} = \mathbf{M}_{1,2} \circ \mathbf{M}_{2,3}$, defined as usual, is masking fault-tolerant for $\langle M_1, M_3\rangle$. This can be proven by checking the conditions of Definition 3.1:

  (A) Consider the initial state s_1 of M_1. Since $\mathbf{M}_{1,2}$ is masking fault-tolerant, there is an initial state s_2 of M_2 with $\langle s_1, s_2\rangle \in \mathbf{M}_{1,2}$. As $\mathbf{M}_{2,3}$ is masking fault-tolerant, there is an initial state s_3 of M_3 with $\langle s_2, s_3\rangle \in \mathbf{M}_{2,3}$. Thus, $\langle s_1, s_3\rangle \in \mathbf{M}_{1,3}$. In the same way, we can check that for any initial state s_3 of M_3, there is an initial state s_1 of M_1 with $\langle s_1, s_3\rangle \in \mathbf{M}_{1,3}$.

  (B.1) By definition of $\mathbf{M}_{1,3}$, there is a state s_2 in M_2 with $\langle s_1, s_2\rangle \in \mathbf{M}_{1,2}$ and $\langle s_2, s_3\rangle \in \mathbf{M}_{2,3}$. Then, $L_1'(s_1) = L_2'(s_2) = L_3'(s_3)$.

  (B.2) Assume $\langle s_1, s_3\rangle \in \mathbf{M}_{1,3}$. As $\langle s_1, s_2\rangle \in \mathbf{M}_{1,2}$ (for some s_2 by definition of $\circ$), it follows that, if $s_1' \in Post_N(s_1)$, then $\langle s_1', s_2'\rangle \in \mathbf{M}_{1,2}$ for some $s_2' \in Post_N(s_2)$. Since $\langle s_2, s_3\rangle \in \mathbf{M}_{2,3}$ (by definition of $\circ$), we have $\langle s_2', s_3'\rangle \in \mathbf{M}_{2,3}$ for some $s_3' \in Post_N(s_3)$. Hence, $\langle s_1', s_3'\rangle \in \mathbf{M}_{1,3}$

  (B.3) Similar to the proof for *(B.2)*.

  (B.4) Assume $\langle s_1, s_3\rangle \in \mathbf{M}_{1,3}$, that is, for some s_2 we have $s_1\, \mathbf{M}_{1,2}\, s_2$ and $s_2\, \mathbf{M}_{2,3}\, s_3$. Now, if $s_3' \in Post_F(s_3)$, then by condition $B.4$ we have a $s_2' \in Post_N(s_2)$ such that $s_2'\, \mathbf{M}_{2,3}\, s_3'$ or $s_2\, \mathbf{M}_{2,3}\, s_3'$; in the latter case, by definition of $\circ$ we have that $s_1\, \mathbf{M}_{1,3}\, s_3'$ and then condition $B.4$ holds. In the former case, since $s_2' \in Post_N(s_2)$ and $s_1\, \mathbf{M}_{1,2}\, s_2$ by condition $B.3$ we have a $s_1' \in Post_N(s_1)$ such that $s_1'\, \mathbf{M}_{1,2}\, s_2'$ and then by $\circ$ we get $s_1'\, \mathbf{M}_{1,3}\, s_3'$.

  Symmetry: In this case the proof is direct since conditions $A, B.1, B.2$ and $B.3$ imply that there is a bisimulation between M and M'.

  Nonreflexivity: We show that $\prec_{Masking}$ is not reflexive via the following counterexample: given the colored Kripke structure M with state space $S = \{s_0, s_1\}$, and initial state s_0, that is, $I = \{s_0\}$, the structure is depicted in Figure 6. Let us prove that it does not hold $M \prec_{Masking} M$. The proof is by contradiction. Suppose that $M \prec_{Masking} M$ that is we have a relation $\mathbf{M} \subseteq S \times S$ that satisfies Definition 3.1. By condition A we must have $s_0\, \mathbf{M}\, s_0$ (since it is the unique initial state). But note that the pair $\langle s_0, s_0\rangle$ does not satisfy condition $(B.4)$ of Definition 3.1: for $s_1 \in Post_F(s_0)$ there is no normal successor (say s') of s_0 such that $s' \prec_{Masking} s_1$, and so $M \not\prec_{Masking} M$.

□

An important property of simulation and bisimulation relations is that they are preserved by unions; a similar property holds for masking, nonmasking and failsafe relations, as proven in the following theorem.

Theorem 3.11. Given $M = \langle S, I, R, L, \mathcal{N}\rangle$ and $M' = \langle S', I', R', L', \mathcal{N}'\rangle$ two colored Kripke structures, and sub-labellings $L_0 \subseteq L, L_0' \subseteq L'$ with $L_0 = L_0'$. The union of two masking/nonmasking/failsafe relations for sub-labelings L_0 and L_0' is a masking/nonmasking/failsafe relation, respectively.

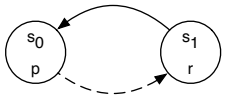

Fig. 6. Counterexample for reflexivity.

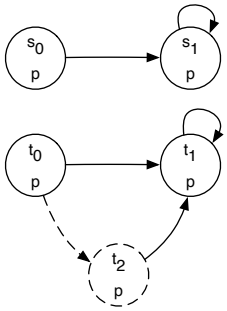

Fig. 7. Counterexample for $\mathsf{FSafe} \cap \mathsf{NMask} \subseteq \mathsf{Mask}$.

Proof. Let us prove that the union of two nonmasking relations is a nonmasking relation. The other ones are similar. Consider two nonmasking relations R and R', conditions A and $B.1, B.2$, and $B.3$ are direct. Let us prove $B.4$, consider a pair of states s_1, s_2 such that $s_1\ R \cup R'\ s_2$, let us assume that for some $s_2' \in Post_F(s_2)$ we have that $s_1'\ R \cup R'\ s_2'$ does not hold for any $s_1' \in Post_N(s_1)$, then, by properties of the union, we get that $s_1'\ R\ s_2'$ and $s_1'\ R'\ s_2'$ do not hold for any $s_1' \in Post_N(s_1)$; which, since R and R' are nonmasking, implies by condition $B.4$ that either we have $s_1\ R\ s_2'$ or $s_1\ R'\ s_2'$ and then, by definition of union, we have $s_1\ R \cup R' s_2'$, that is, condition $B.4$ holds. □

As a corollary, we obtain that there exists a coarsest (that is, maximum w.r.t. inclusion [BK08]) masking/nonmasking/failsafe relation.

Corollary 3.12. Given $M = \langle S, I, R, L, \mathcal{N} \rangle$ and $M' = \langle S', I', R', L', \mathcal{N}' \rangle$ two colored Kripke structures and sub-labellings $L_0 \subseteq L, L_0' \subseteq L'$ with $L_0 = L_0'$, there exists a coarsest masking/nonmasking/failsafe relation for $M \times M'$ and sub-labeling L_0.

Proof. Suppose that M and M' are colored Kripke structures, then the relation $\mathbf{M} \subseteq S \times S'$:

$$\mathbf{M} = \bigcup \{ \mathbf{R} \mid \mathbf{R} \text{ is a masking relation } \}$$

is a masking relation by Theorem 3.11, and by definition it contains any other masking relation. The proof is similar for nonmasking and failsafe relations. □

Finally, we prove that, in the case of NDF structures, the masking relations are also nonmasking and failsafe.

Theorem 3.13. Let Mask, NMask and FSafe be the sets of masking, nonmasking and failsafe relations between two colored Kripke structures M and M' over vocabulary AP', where M' is a NDF structure; then we have:

$\mathsf{Mask} \subseteq \mathsf{FSafe} \cap \mathsf{NMask}$

Proof. Note that the inclusion $\mathsf{Mask} \subseteq \mathsf{FSafe}$ is straightforward by definition of masking and failsafe. Now, let us prove $\mathsf{Mask} \subseteq \mathsf{NMask}$. First note that conditions A, $B.1$, $B.2$, and $B.3$ are the same in each one of the definitions. Suppose that $s\,\mathbf{M}\,s'$, and $t' \in Post_F(s')$, if there is a $t \in Post_N(s)$ such that $t\,\mathbf{M}\,t'$, then we also have that condition $B.4$ of Definition 3.4 holds, and so the relation is also a nonmasking relationship. Now, in the other case we have $s\,\mathbf{M}\,t'$, but since M' is NDF, we can always eventually reach a non-faulty action from t, and thus we have a fragment $t' \Rightarrow^* w$ such that $s'\,\mathbf{M}\,w$ by condition $B.3$. Thus the relation also satisfies condition $B.4$ of nonmasking. □

It is interesting to note that the other inclusion does not hold, that is, we have relations that are both

failsafe and nonmasking but which are not masking fault tolerant. A simple example is given in Figure 7. Consider the relation $R = \{\langle s_0, t_0\rangle, \langle s_1, t_1\rangle\}$, it is a failsafe and also a nonmasking fault-tolerant relation for the two structures shown in the figure, and the sub-labeling obtained by restricting the original labelings to the letter p. It is not a masking fault-tolerant relation because condition $B.4$ of Definition 3.1 does not hold. However, any relation that is nonmasking and failsafe can be extended to a masking relation. Note in the figure above if we add to R the pair $\langle s_0, t_2\rangle$, then the resulting relation is masking.

Theorem 3.14. Let Mask, NMask and FSafe be the sets of masking, nonmasking and failsafe relations between two colored Kripke structures M and M' over vocabulary AP', where M' is a NDF structure; then we have:

$$\forall R \in \mathsf{NMask} \cap \mathsf{FSafe} : \exists R^+ \in \mathsf{Mask} : R \subset R^+$$

Proof. We define R^+. Let $M_1 = \langle S_1, I_1, R_1, L_1, \mathcal{N}_1\rangle$ and $M_2 = \langle S_2, I_2, R_2, L_2, \mathcal{N}_2\rangle$ be Kripke structures, where $R \subseteq S_1 \times S_2$ is defined over a vocabulary AP'. Let $\{w_1, \ldots, w_k\} \subseteq S_2$ be the set of states such that for any w_i in this set there are $s_1 \in S_2, s_2 \in S_2$ with $s_1\ R\ s_2$, and $w_i \in Post_F(s_2)$ and there is no $s_1' \in Post_N(s_1)$ with $s_1'\ R\ w_i$. Note that, if this set is empty, R is masking. Since R is nonmasking we know that there is a s_2'' such that $s_2'' \in Post^*(w_i)$ and $s_1'\ R\ s_2''$ (for some $s_1' \in Post_N(s_1)$), then we define the set:

$$\{\langle s_1', s_2^i\rangle \mid s_2^i \in Post^*(w_i)\}$$

We call this set $C(w_i)$. Note that all the elements in this set have the same labeling wrt AP' which coincides with that of s_1' by definition of failsafe. We define:

$$R^+ = R \cup \bigcup_{w_i} C(w_i)$$

Let us see that this relation is masking. Suppose that $s_1\ R^+\ s_2$. If $s_1\ R\ s_2$, then we know that items $A, B.1, B.2, B.3$ of Definition 3.1 hold. Let us prove $B.4$ holds. By definition of R^+ we know that for any $s_2' \in Post_F(s_2)$ there is a s_1' such that $s_1'\ R^+\ s_2'$, that satisfies condition A (by properties of failsafe) and then it satisfies $B.1, B.2, B.3, B.4$.

For the case that $s_1\ R^+\ s_2$ but not $s_1\ R\ s_2$ we know that they have the same labeling by definition of R^+, and as before the rest of the items of the definition of masking hold by the properties of nonmasking and failsafe. □

Summing up, if we have an implementation that is both failsafe and nonmasking, we know that we can obtain a relation that "proves" that this implementation is also masking, thus in practice masking is equivalent to the intersection of nonmasking and failsafe.

4. Checking Fault-Tolerance Properties

Simulation and bisimulation relations are amenable to polynomial computational treatment. In [BK08, HHK95], algorithms for calculating several simulation and bisimulation relations are described and proved to be of polynomial complexity with respect to the number of states and transitions of the corresponding models. We have adapted these algorithms to our setting, thus obtaining polynomial procedures to compute masking, nonmasking and failsafe fault-tolerance. Such algorithms can be used to verify whether $M \sqsubset M'$, with $\sqsubset \in \{\prec_{Mask}, \prec_{Nonmask}, \prec_{Failsafe}\}$.

The goal of this section is to present the algorithms for computing masking, nonmasking, and failsafe relations. In general, these algorithms take as input a colored Kripke structure $M = \langle S, I, R, L, \mathcal{N}\rangle$ and a sub-labeling $L_0 \subseteq L$ and produce as output either the required fault-tolerance relation $\mathbf{M}$, $\mathbf{N}$, or $\mathbf{F}$ in case that it exists or an empty relation, otherwise. Clearly, these algorithms yield at the same time an automatic approach to check whether the fault-tolerant implementation is masking, nonmasking, or failsafe fault-tolerant with respect to the system specification. For this case, the input M is the result of combining two colored Kripke structures $M_1 = \langle S_1, I_1, R_1, L_1, \mathcal{N}_1\rangle$ and $M_2 = \langle S_2, I_2, R_2, L_2, \mathcal{N}_2\rangle$ over AP via disjoint union, i.e., $M_1 \oplus M_2$. The former represents the system specification which exhibits the normal behavior of the system and the latter the fault-tolerant implementation which is the extended version of the system specification augmented with faults. In other words, we can check whether the extended version of the nominal model with faults accomplishes the desired level of fault-tolerance.

The general scheme for computing the masking, nonmasking, or failsafe fault-tolerant relation for a colored Kripke structure $M = \langle S, I, R, L, \mathcal{N} \rangle$ and a sub-labeling $L_0 \subseteq L$ is sketched in Algorithm 1. We develop this main algorithm in the following three steps:

- **Step 1:** computation of condition $B.2$ (same algorithm for all levels of fault-tolerance).
- **Step 2:** computation of condition $B.3$ and $B.4$ for the corresponding level of fault-tolerance (masking, nonmasking, or failsafe).
- **Step 3:** merge of the results obtained in *step 1* and *step 2* to return the corresponding fault-tolerant relation **M**, **N**, or **F**.

Algorithm 1 Computation of a fault-tolerant relation

Input: Colored Kripke structure $M = \langle S, I, R, L, \mathcal{N} \rangle$, a sub-labeling $L_0 \subseteq L$, and a required level of fault-tolerance $R \in \{masking, nonmasking, failsafe\}$
Output: Fault-tolerant relation (**M**, **N**, or **F**)
1: $SimB2 := ComputeB2(M, L_0)$
2: **switch** R :
3: **case** masking
4: $\quad SimB3B4 := ComputeMaskB3B4(M, L_0)$
5: **case** nonmasking
6: $\quad SimB3B4 := ComputeNonMaskB3B4(M, L_0)$
7: **case** failsafe
8: $\quad SimB3B4 := ComputeFailsafeB3B4(M, L_0)$
9: **end switch**
10: $FTSimRel := MergeSimRel(SimB2, SimB3B4)$
11: **return** $FTSimRel$

In the first step of Algorithm 1, we compute a relation $SimB2$ that satisfies condition $B.2$ of Definition 3.1, 3.4 and 3.6. This is performed by procedure $ComputeB2$ on line 1. Notice that this condition is the same for the three levels of fault-tolerance. In the second step, we compute the corresponding relation $SimB3B4$ that satisfies condition $B.3$ and $B.4$ for the required level of fault-tolerance (lines $2-9$). We have developed one algorithm for each case, namely $ComputeMaskB3B4$, $ComputeNonMaskB3B4$, and $ComputeFailsafeB3B4$. In the last step, we combine the results obtained from the previous steps ($MergeSimRel$ on line 10) to return the corresponding fault-tolerant relation.

In the following subsections we explain all the details for each procedure used in the main algorithm.

4.1. Computing Masking Fault-Tolerance

In this subsection we present the algorithms for computing the masking fault-tolerant relation **M** for a colored Kripke Structure $M = \langle S, I, R, L, \mathcal{N} \rangle$ and a sub-labeling $L_0 \subseteq L$, where $\mathbf{M} \subseteq S \times S$. We start describing Algorithm 2 ($ComputeB2$) that computes condition $B.2$ of Definition 3.1.

First, let us note that Algorithm 2 is also used in the algorithms for computing nonmasking and failsafe relations, since the definitions of these relations also require condition $B.2$. This algorithm is an adaptation from a standard procedure for computing simulation relations introduced in [HHK95]. Let us now describe the general idea of Algorithm 2. It takes as input a colored Kripke structure $M = \langle S, I, R, L, \mathcal{N} \rangle$ and a sub-labeling $L_0 \subseteq L$ and computes a $SimB2$ relation, where the states in $SimB2$ satisfy condition $B.2$ of Definition 3.1, 3.4, and 3.6. In the first loop, for each $s_1 \in \mathcal{N}$, the set $SimB2(s_1)$ contains the states that are candidates for masking s_1. Initially, $SimB2(s_1)$ consists of all normal and faulty states with the same labels as s_1 and $RemoveR(s_1)$ contains all states which do not have a normal or faulty successor state masking s_1. Moreover, these states cannot mask any of the predecessors of s_1: note that $Post(s_2) \cap SimB2(s_1) \neq \emptyset$ iff $s_2 \in Pre(SimB2(s_1))$. That is, we first obtain all the predecessors of those states that simulate s_1 ($Pre(SimB2(s_1))$), then we perform the difference between S and this set of predecessors. As a result we obtain a set of states where each state in it does not have a successor that simulates s_1 and thus these states cannot simulate any of the predecessors of s_1. The termination condition of the loop of lines 5-18 is $RemoveR(s_1') = \emptyset$ for all $s_1' \in \mathcal{N}$, in which case there are no states that need to be removed from the sets of

Algorithm 2 Computation of condition $B.2$ of Definition 3.1, 3.4, and 3.6 ($ComputeB2(M, L_0)$)

Input: Colored Kripke structure $M = \langle S, I, R, L, \mathcal{N} \rangle$ and a sub-labeling $L_0 \subseteq L$
Output: Relation $SimB2$ where the states in it satisfy condition $B.2$

1: **for all** $s_1 \in \mathcal{N}$ **do**
2: $SimB2(s_1) := \{s_2 \in S \mid L_0(s_1) = L_0(s_2)\}$
3: $RemoveR(s_1) := S \backslash Pre(SimB2(s_1))$
4: **end for**
5: **while** $\exists\, s_1' \in \mathcal{N}$ with $RemoveR(s_1') \neq \emptyset$ **do**
6: select s_1' such that $RemoveR(s_1') \neq \emptyset$
7: **for all** $s_2 \in RemoveR(s_1')$ **do**
8: **for all** $s_1 \in Pre_N(s_1')$ **do**
9: **if** $s_2 \in SimB2(s_1)$ **then**
10: $SimB2(s_1) := SimB2(s_1) \backslash \{s_2\}$
11: **for all** $s \in Pre(s_2)$ with $Post(s) \cap SimB2(s_1) = \emptyset$ **do**
12: $RemoveR(s_1) := RemoveR(s_1) \cup \{s\}$
13: **end for**
14: **end if**
15: **end for**
16: **end for**
17: $RemoveR(s_1') := \emptyset$
18: **end while**
19: **return** $\{\langle s_1, s_2 \rangle \mid s_2 \in SimB2(s_1)\}$

simulators $SimB2(s_1)$ for $s_1 \in Pre_N(s_1')$. Within the while-loop body, the main idea is to consider all pairs $(s_1, s_2) \in \mathcal{N} \times S$ such that $s_2 \in RemoveR(s_1')$ and $s_1 \in Pre_N(s_1')$. This means that $s_1 \rightarrow s_1'$ but there is no transition $s_2 \rightarrow s_2'$ or $s_2 \dashrightarrow s_2'$ with $s_2' \in SimB2(s_1')$ because we have that $s_2 \in RemoveR(s_1')$, which means that $\nexists\, s_2' \in Post(s_2)$ with $s_2' \in SimB2(s_1')$. This yields that s_1 is not masked by s_2. Therefore, s_2 is removed from $SimB2(s_1)$ if $s_2 \in SimB2(s_1)$. Subsequently, we add to the set $RemoveR(s_1)$ all predecessors s of s_2 such that $Post(s) \cap SimB2(s_1) = \emptyset$. Intuitively speaking, for each $s_1' \in \mathcal{N}$ with $RemoveR(s_1') \neq \emptyset$ we explore all states s_2 that do not have any successor that masks s_1' (line 7). Then, we test whether there exists any normal predecessor s_1 of s_1' (line 8) such that s_1 is masked by s_2. If so, this is not correct because we know that there does not exist any successor of s_2 that masks s_1'; consequently s_2 is removed from $SimB2(s_1)$.

We now briefly explain Algorithm 3 ($ComputeMaskB3B4$). This takes as input a colored Kripke structure $M = \langle S, I, R, L, \mathcal{N} \rangle$ and a sub-labeling $L_0 \subseteq L$ and computes a $MaskB3B4$ relation, where the states in $MaskB3B4$ satisfy condition $B.3$ and $B.4$ of Definition 3.1. For each $s_2 \in S$, the set $MaskB3B4(s_2)$ contains the normal states that are candidates for masking s_2. Initially, $MaskB3B4(s_2)$ consists of all normal states with the same labels as s_2 and $Remove(s_2)$ contains all the normal states which do not have a (normal) successor state masking s_2. Moreover, these states cannot mask any of the predecessors of s_2. The termination condition of the loop of lines 5-28 is $Remove(s_2') = \emptyset$ for all $s_2' \in S$, in which case there are no normal states that need to be removed from the sets of simulators $MaskB3B4(s_2)$ for $s_2 \in Pre_N(s_2')$ or $s_2 \in Pre_F(s_2')$. Within the while-loop body, we take care of normal transitions ($B.3$) and faulty transitions ($B.4$) of Definition 3.1. Firstly, in the for loop of lines 9-16, the main idea is to consider all pairs $(s_1, s_2) \in S \times \mathcal{N}$ such that $s_1 \in Remove(s_2')$ and $s_2 \in Pre_N(s_2')$. This means that $s_2 \rightarrow s_2'$ but there is no transition $s_1 \rightarrow s_1'$ with $s_1' \in MaskB3B4(s_2')$ because we have that $s_1 \in Remove(s_2')$, which means that $\nexists\, s_1' \in Post_N(s_1)$ with $s_1' \in MaskB3B4(s_2')$. This yields that s_2 is not masked by s_1. Therefore, s_1 is removed from $MaskB3B4(s_2)$ if $s_1 \in MaskB2B3(s_2)$. Subsequently, we add to the set $Remove(s_2)$ all normal predecessors s of s_1 such that $Post(s) \cap MaskB3B4(s_2) = \emptyset$. Secondly, in the for loop of lines 18-26, we now consider all pairs $(s_1, s_2) \in S \times S$ such that $s_1 \in Remove(s_2')$ and $s_2 \in Pre_F(s_2')$. This means that $s_2 \dashrightarrow s_2'$ but there is no transition $s_1 \rightarrow s_1'$ with $s_1' \in MaskB3B4(s_2')$ because we have that $s_1 \in Remove(s_2')$, which means that $\nexists\, s_1' \in Post_N(s_1) : s_1' \in MaskB3B4(s_2')$. This yields that s_2 is not masked by s_1. Therefore, s_1 is removed from $MaskB3B4(s_2)$ if $s_1 \in MaskB2B3(s_2)$ and $s_1 \notin MaskB3B4(s_2')$, that is, the last part of condition $B.4$. Subsequently, we add to the set $Remove(s_2)$ all normal predecessors s of s_1 such that $Post(s) \cap MaskB3B4(s_2) = \emptyset$, where we also check that $s \notin MaskB3B4(s_2) \vee s \notin MaskB3B4(Pre_F(s_2))$, that is, the last part of condition $B.4$.

Algorithm 3 Computation of condition $B.3$ and $B.4$ of Definition 3.1 ($ComputeMaskB3B4(M, L_0)$)

Input: Colored Kripke structure $M = \langle S, I, R, L, \mathcal{N} \rangle$ and a sub-labeling $L_0 \subseteq L$
Output: Relation $MaskB3B4$ where the states in it satisfy condition $B.3$ and $B.4$ of Definition 3.1

1: **for all** $s_2 \in S$ **do**
2: $MaskB3B4(s_2) := \{s_1 \in \mathcal{N} \mid L_0(s_1) = L_0(s_2)\}$
3: $Remove(s_2) := \mathcal{N} \backslash Pre_N(MaskB3B4(s_2))$
4: **end for**
5: **while** $\exists s_2' \in S$ with $Remove(s_2') \neq \emptyset$ **do**
6: select s_2' such that $Remove(s_2') \neq \emptyset$
7: **for all** $s_1 \in Remove(s_2')$ **do**
8: {This takes care of non-faulty transitions}
9: **for all** $s_2 \in Pre_N(s_2')$ **do**
10: **if** $s_1 \in MaskB3B4(s_2)$ **then**
11: $MaskB3B4(s_2) := MaskB3B4(s_2) \backslash \{s_1\}$
12: **for all** $s \in Pre_N(s_1)$ with $Post_N(s) \cap MaskB3B4(s_2) = \emptyset$ **do**
13: $Remove(s_2) := Remove(s_2) \cup \{s\}$
14: **end for**
15: **end if**
16: **end for**
17: {This takes care of faulty transitions}
18: **for all** $s_2 \in Pre_F(s_2')$ **do**
19: **if** $s_1 \in MaskB3B4(s_2) \wedge s_1 \notin MaskB3B4(s_2')$ **then**
20: $MaskB3B4(s_2) := MaskB3B4(s_2) \setminus \{s_1\}$
21: **for all** $s \in Pre_N(s_1)$ with $Post_N(s) \cap MaskB3B4(s_2) = \emptyset \wedge (s \notin MaskB3B4(s_2) \vee s \notin MaskB3B4(Pre_F(s_2)))$ **do**
22: $Remove(s_2) := Remove(s_2) \cup \{s\}$
23: **end for**
24: **end if**
25: **end for**
26: **end for**
27: $Remove(s_2') := \emptyset$
28: **end while**
29: **return** $\{\langle s_1, s_2 \rangle \mid s_1 \in MaskB3B4(s_2)\}$

Finally, the masking fault-tolerance **M** relation is obtained by Algorithm 4 ($MergeSimRel$), which takes as input two sets obtained from Algorithm 2 and Algorithm 3 and produces a set combining these sets. This algorithm ensures a correct merging of the relation $SimB2$ and $MaskB3B4$, where we use some notations coming from relation algebra: for a given relation R, $R[s]$ denotes the relational image of s under R, $R^{\sim}$ is the relational inverse of R, whereas π_1 and π_2 are the first and second projections, respectively.

The proofs of correctness and termination are obtained by adapting the corresponding proofs given in [BK08].

Theorem 4.1 (Partial Correctness of Masking). On termination, Algorithms 2, 3 and 4 return a relation $\mathbf{M} \subseteq S \times S$.

Proof. We have to prove that Algorithm 3 ensures conditions $B.3$ and $B.4$ of Definition 3.1. First, let us note that the loop of line 5 has the following loop invariant. For all state $s_2 \in S$:

(a) $Remove(s_2) \subseteq \mathcal{N} \backslash Pre_N(MaskB3B4(s_2))$
(b) for any relation $MaskB3B4$: $\{s_1 \in \mathcal{N} \mid s_1\ MaskB3B4\ s_2\} \subseteq MaskB3B4(s_2) \subseteq \{s_1 \in \mathcal{N} \mid L_0(s_1) = L_0(s_2)\}$
(c) $\forall s_1 \in MaskB3B4(s_2)$, either:
  (1) $\exists s_2' \in Post_N(s_2)$ with $Post_N(s_1) \cap MaskB3B4(s_2') = \emptyset$ and $s_1 \in Remove(s_2')$,
  (2) $\exists s_2' \in Post_F(s_2)$ with $Post_N(s_1) \cap MaskB3B4(s_2') = \emptyset$ and $s_1 \notin MaskB3B4(s_2')$ and $s_1 \in Remove(s_2')$,

Algorithm 4 Combination of $SimRel1$ and $SimRel2$ ($MergeSimRel(SimRel1, SimRel2)$)

Input: Set $SimRel1$ and $SimRel2$
Output: Set R

1: $R = \{\langle s_1, s_2\rangle \mid s_1 \in SimRel2(s_2) \wedge s_2 \in SimRel1(s1)\}$
2: $RemoveL = \{s_1 \in \pi_1(R) \mid \exists s_1' \in Post_N(s_1) : SimRel1(s_1') = \emptyset\}$
3: $RemoveR = \{s_2 \in \pi_2(R) \mid \exists s_2' \in Post_N(s_2) : SimRel2(s_2') = \emptyset\}$
4: **while** $RemoveL \cup RemoveR \neq \emptyset$ **do**
5: choose $s \in RemoveL \cup RemoveR$
6: **if** $s \in RemoveL$ **then**
7: $RemoveL := RemoveL \setminus \{s\}$
8: **for all** $s' \in R[s]$ **do**
9: $R := R \setminus \{\langle s, s'\rangle\}$
10: **if** $R^{\sim}[s'] = \emptyset$ **then**
11: $RemoveR := RemoveR \cup \{s'\} \cup (Pre(s') \cap \pi_2(R))$
12: **end if**
13: $RemoveL = RemoveL \cup (Pre_N(s) \cap \pi_1(R))$
14: **end for**
15: **end if**
16: **if** $s \in RemoveR$ **then**
17: $RemoveR := RemoveR \setminus \{s\}$
18: **for all** $s' \in R^{\sim}[s]$ **do**
19: $R := R \setminus \{\langle s, s'\rangle\}$
20: **if** $R[s'] = \emptyset$ **then**
21: $RemoveL := RemoveL \cup \{s'\} \cup Pre_N(s')$
22: **end if**
23: $RemoveR = RemoveR \cup (Pre(s) \cap \pi_1(R))$
24: **end for**
25: **end if**
26: **end while**
27: **return** R

(3) $\forall s_2' \in Post_N(s_2) : Post_N(s_1) \cap MaskB3B4(s_2') \neq \emptyset$,

(4) $\forall s_2' \in Post_F(s_2) : Post_N(s_1) \cap MaskB3B4(s_2') \neq \emptyset \ \vee \ s_1 \in MaskB3B4(s_2')$.

From (c), we obtain that, when $Remove(s_2') = \emptyset$ for every $s_2' \in S$, then: $\forall s_2 \in S : \forall s_1 \in MaskB3B4(s_2) : \forall s_2' \in Post_F(s_2) : Post_N(s_1) \cap MaskB3B4(s_2') \neq \emptyset \vee s_1 \in MaskB3B4(s_2')$. That is, the relation defined as $s_1 \ MaskB3B4 \ s_2$ satisfies condition $B.4$, and similarly for $B.3$, of Definition 3.1. The proof for Algorithm 2 is similar.

We now prove that *MergeSimRel* generates a masking relation $\mathbf{M}$ from these two sets. We instantiate the parameter set $SimRel1$ and $SimRel2$ with $SimB2$ and $MaskB3B4$, respectively. Note that the invariants of Algorithm 4 are:

- $\mathbf{M} \subseteq R \subseteq \{\langle s_1, s_2\rangle \mid s_1 \in MaskB3B4(s_2) \wedge s_2 \in SimB2(s_1)\}$, for any masking relation $\mathbf{M}$
- $RemoveL = \{s_1 \in \pi_1(R) \mid \exists s_1' \in Post_N(s_1) : SimB2(s_1') = \emptyset\}$
- $RemoveR = \{s_2 \in \pi_2(R) \mid \exists s_2' \in Post(s_2) : MaskB3B4(s_2) = \emptyset\}$

Note that $RemoveL$ and $RemoveR$ contain those elements in R that may violate the definition of masking, this may happen since some successors of a given state are removed from the relation when merging $SimB2$ and $MaskB3B4$; however, on termination $RemoveL$ and $RemoveR$ are empty, and in addition we have $\mathbf{M} \subseteq R \subseteq \{\langle s_1, s_2\rangle \mid s_1 \in MaskB3B4(s_2) \wedge s_2 \in SimB2(s_1)\}$, thus R is a masking relation, and furthermore is the coarsest masking one. □

Lemma 4.2 (Termination of Masking). Algorithms 2, 3 and 4 terminate.

Proof. We prove termination of Algorithm 3; the proofs for the others are similar. The key point is to note that any state s_1 can only be inserted into $Remove(s')$ once. That is, once we process it, it will

never be inspected again in line 7 of this algorithm. Note that, if $s_1 \in Remove(s_2')$ and let s_2' be the state that is selected in line 6, then $s_1 \notin Pre_N(MaskB3B4(s_2'))$. Moreover, since the $MaskB3B4$ sets are decreasing (line 11 and 20 of Algorithm 3), $s_1 \notin Pre_N(MaskB3B4(s_2'))$ in all further iterations. The only reason to insert s_1 in $Remove(s_2')$ is when $s_1 \in Pre_N(s_1'')$ for some state $s_1'' \in MaskB3B4(s_2')$ with $\{s_1''\} = Post(s_1) \cap MaskB3B4(s_2')$. But then $s_1 \in Pre_N(MaskB3B4(s_2'))$, which is a contradiction, and therefore s_1 will never be added again to $Remove(s_2')$. □

Theorem 4.3 (Complexity of Masking). The masking fault-tolerance relation **M** of a colored Kripke structure $M = \langle S, I, R, L, \mathcal{N} \rangle$ for sub-labeling $L_0 \subseteq L$ obtained by restricting AP to AP', can be computed with algorithms 2 and 3 in a running time of $\mathcal{O}(|S|^2 * |AP'| + |E| * |S|)$ where $|E|$ is the number of edges of the structure and $|S|$ the number of states.

Proof. Let us prove the result for Algorithm 3; the proofs for the others are similar. Let $|E|$ be the number of edges of M. Similar to [HHK95], we use an array to keep track of the numbers $count(s_1, s_2') = |Post_N(s_1) \cap MaskB3B4(s_2')|$. The initialization of $MaskB3B4(s)$ for any s can be done in time $\mathcal{O}(|S| * |AP'|)$; thus, initializing $MaskB3B4$ takes $\mathcal{O}(|S|^2 * |AP'|)$ time. On the other hand, $Remove(s)$ can be computed in time $\mathcal{O}(|S|)$ for any s. Then calculating line 3 of the algorithm takes at most $\mathcal{O}(|S|^2)$ (or $\mathcal{O}(|E|)$) steps. Following the argument of Lemma 4.2, note that the loop of line 5 is executed at most once for each s_1 and $s \to s_1$; furthermore, using the array *counters*, lines $9-16$ can be computed in time $\mathcal{O}(|S|)$ in the worst case; thus, the entire loop takes time $\mathcal{O}(|E| * |S|)$. The same argument holds for lines $18-25$. Thus the algorithm has a running time of $\mathcal{O}(|S|^2 * |AP'| + |E| * |S|)$, which in the worst case is $\mathcal{O}(|S|^2 * |AP'| + |S|^3)$. □

4.2. Computing Nonmasking Fault-Tolerance

Let us now explain Algorithm 5 for computing a relation that satisfies the condition $B.3$ and $B.4$ of Definition 3.4 for a colored Kripke Structure $M = \langle S, I, R, L, \mathcal{N} \rangle$ and a sub-labeling $L_0 \subseteq L$. For each $s_2 \in S$, the set $NonMaskB3B4(s_2)$ will contain the normal states that are nonmasking for s_2. Initially, $NonMaskB3B4(s_2)$ consists of all normal states with the same labels as s_2, while $Remove(s_2)$ contains all the normal states which do not have successor states simulating some successor state from s_2. We also consider a set $Remove^+(s_2)$, which intuitively contains those states that do not have successor states simulating some state reachable from s_2. Both sets are updated while inspecting the structure. This is the main difference with Algorithm 3. Note that in the algorithm we use the set of all states reachable from a state s starting with a faulty action; this is defined formally as: $Post^+(s) = \{s'' \mid \exists s' \in Post_F(s) : s' \Rightarrow s''\}$. Similarly, we define the transitive-reflexive closure as: $Post^*(s) = \{s'' \mid s \Rightarrow s''\}$. Inside the loop of lines 6-33, we compute the sets $Remove$ and $Remove^+$ and the collection $NonMaskB3B4(s)$ is updated following conditions $B.3$ and $B.4$.

The termination condition of the loop of lines 6-33 is $Remove(s_2') \cup Remove^+(s_2') = \emptyset$ for all $s_2' \in S$, in that case there are no normal states that need to be removed from the sets of simulators $NonMaskB3B4(s_2)$ for $s_2 \in Pre_N(s_2')$ or $s_2 \in Pre_F(s_2')$. Within the while-loop body, we take care of normal transitions ($B.3$) and faulty transitions ($B.4$) of Definition 3.4. Firstly, in the for loop of lines 9-19, the main idea is to consider all pairs $(s_1, s_2) \in S \times \mathcal{N}$ such that $s_1 \in Remove(s_2')$ and $s_2 \in Pre_N(s_2')$. This means that $s_2 \to s_2'$ but there is no transition $s_1 \to s_1'$ with $s_1' \in NonMaskB3B4(s_2')$ because we have that $s_1 \in Remove(s_2')$, which means that $\nexists\, s_1' \in Post_N(s_1)$ with $s_1' \in NonMaskB3B4(s_2')$. This yields that s_2 is not nonmasking by s_1. Therefore, s_1 is removed from $NonMaskB3B4(s_2)$ if $s_1 \in NonMaskB2B3(s_2)$. Subsequently, we add to the set $Remove(s_2)$ all normal predecessors s of s_1 such that $Post_N(s) \cap NonMaskB3B4(s_2) = \emptyset$. Secondly, in the for loop of lines $20-31$, we now consider all pairs $(s_1, s_2) \in S \times S$ such that $s_1 \in Remove^+(s_2')$ and $s_2 \in Pre_F(s_2')$. This means that $s_2 \dashrightarrow s_2'$ but there is no transition $s_1 \to s_1'$ with $s_1' \in NonMaskB3B4(s_2'')$ for some $s_2'' \in Post^*(s_2')$. This yields that s_2 is not nomasking by s_1. Therefore, s_1 is removed from $NonMaskB3B4(s_2)$ if $s_1 \in NonMaskB2B3(s_2)$. Subsequently, we add to the set $Remove(s_2)$ and $Remove^+(s_2)$ all normal predecessors s of s_1 such that $Post_N(s) \cap NonMaskB3B4(Post^*(s_2)) = \emptyset$, where we also check that $s \notin NonMaskB3B4(Pre_F(s_2))$.

Note that the transitive closure can be computed in cubic time w.r.t. the set of states; however, in practice, when constructing the graph that describes our system, we can also construct the transitive closure at the same time, improving the complexity of Algorithm 5. Finally, we combine the sets $SimB2$ and $NonMaskB3B4$ through Algorithm 4 by instantiating the parameter set $SimRel1$ and $SimRel2$ with $SimB2$ and $NonMaskB3B4$, respectively.

Let us prove the correctness of the algorithm for computing nonmasking relations.

Algorithm 5 Computation of condition $B.3$ and $B.4$ of Definition 3.4 ($ComputeNonMaskB3B4(M, L_0)$)

Input: Colored Kripke structure $M = \langle S, I, R, L, \mathcal{N} \rangle$ and a sub-labeling $L_0 \subseteq L$
Output: Relation $NonMaskB3B4$ where the states in it satisfy condition $B.3$ and $B.4$ of Definition 3.4

1: **for all** $s_2 \in S$ **do**
2: $NonMaskB3B4(s_2) := \{s_1 \in \mathcal{N} \mid L_0(s_1) = L_0(s_2)\}$
3: $Remove(s_2) := \mathcal{N} \backslash Pre_N(NonMaskB3B4(s_2))$
4: $Remove^+(s_2) := \mathcal{N} \backslash Pre_N(NonMaskB3B4(Post^*(s_2)))$
5: **end for**
6: **while** $\exists\ s_2' \in S$ with $Remove(s_2') \cup Remove^+(s_2') \neq \emptyset$ **do**
7: select s_2' such that $Remove(s_2') \cup Remove^+(s_2') \neq \emptyset$
8: **for all** $s_1 \in Remove(s_2') \cup Remove^+(s_2')$ **do**
9: **if** $s_1 \in Remove(s_2')$ **then**
10: **for all** $s_2 \in Pre_N(s_2')$ **do**
11: **if** $s_1 \in NonMaskB3B4(s_2)$ **then**
12: $NonMaskB3B4(s_2) := NonMaskB3B4(s_2) \backslash \{s_1\}$
13: **for all** $s \in Pre_N(s_1)$ with $Post_N(s) \cap NonMaskB3B4(s_2) = \emptyset$ **do**
14: $Remove(s_2) := Remove(s_2) \cup \{s\}$
15: $Remove^+(s_2) := Remove^+(s_2) \cup \{s\}$
16: **end for**
17: **end if**
18: **end for**
19: **end if**
20: **if** $s_1 \in Remove^+(s_2')$ **then**
21: **for all** $s_2 \in Pre_F(s_2')$ **do**
22: **if** $s_1 \in NonMaskB3B4(s_2)$ **then**
23: $NonMaskB3B4(s_2) := NonMaskB3B4(s_2) \backslash \{s_1\}$
24: **for all** $s \in Pre_N(s_1)$ with $Post_N(s) \cap NonMaskB3B4(Post^*(s_2)) = \emptyset \wedge (s \notin NonMaskB3B4(Pre_F(s_2)))$ **do**
25: $Remove^+(s_2) := Remove(s_2) \cup \{s\}$
26: $Remove(s_2) := Remove(s_2) \cup \{s\}$
27: **end for**
28: **end if**
29: **end for**
30: **end if**
31: **end for**
32: $Remove(s_2) := \emptyset$
33: **end while**
34: **return** $\{\langle s_1, s_2 \rangle \mid s_1 \in NonMaskB3B4(s_2)\}$

Theorem 4.4 (Partial Correctness of Algorithm 5). On termination, Algorithm 5 returns **N**.

Proof. We have to prove that Algorithm 5 ensures conditions $B.3$ and $B.4$ of Definition 3.4. In the first place, notice that the loop of line 6 maintains the following loop invariant. For all faulty states $s_2 \in S$:

(a) $Remove(s_2) \subseteq \mathcal{N} \backslash Pre_N(NonMaskB3B4(s_2))$
(b) $Remove^+(s_2) \subseteq \mathcal{N} \backslash Pre_N(NonMaskB3B4(Post^*(s_2)))$
(c) for any relation $NonMaskB3B4$: $\{s_1 \in \mathcal{N} \mid s_1\ NonMaskB3B4\ s_2\} \subseteq NonMaskB3B4(s_2) \subseteq \{s_1 \in \mathcal{N} \mid L_0(s_1) = L_0(s_2)\}$
(d) $\forall s_1 \in NonMaskB3B4(s_2)$, either:
  (1) $\exists s_2' \in Post_N(s_2)$ with $Post_N(s_1) \cap NonMaskB3B4(s_2') = \emptyset \wedge s_1 \in Remove(s_2')$, or
  (2) $\exists s_2' \in Post_F(s_2)$ with $Post_N(s_1) \cap NonMaskB3B4(Post^+(s_2')) = \emptyset\ \wedge s_1 \in Remove^+(s_2')$, or
  (3) $\forall s_2' \in Post_N(s_2) : Post_N(s_1) \cap NonMaskB3B4(s_2') \neq \emptyset$ and $\forall s_2' \in Post^+(s_2) : Post_N(s_1) \cap NonMaskB3B4(s_2') \neq \emptyset$

From (d), we obtain that, when $Remove(s_2') = \emptyset$ for all $s_2' \in S$, then $\forall s_2 \in S : \forall s_1 \in NonMaskB3B4(s_2)$:

- $\forall s_2' \in Post_N(s_2) : Post_N(s_1) \cap NonMaskB3B4(s_2') \neq \emptyset$ and
- $\forall s_2' \in Post^+(s_2) : Post_N(s_1) \cap NonMaskB3B4(s_2') \neq \emptyset$

That is, conditions $B.3$ and $B.4$ of Definition 3.4 hold. Finally, the proof that procedure $MergeSimRel$ merges $SimB2$ and $NonMaskB3B4$ in a correct way is similar to the proof given for the masking relation **M**. □

The proof of termination of Algorithm 5 is similar to the proof of Lemma 4.2; the reader is referred to that result. With respect to the complexity of computing the nonmasking fault-tolerance relation **N**, it maintains polynomial complexity with respect to the traditional simulation algorithms, as proven in the following theorem.

Theorem 4.5. The running time of the algorithm for checking a nonmasking relationship is in worst case $\mathcal{O}(|S|^4 + |S|^2 * |AP'|)$.

Proof. First, using the same reasoning as in Theorem 4.3, we get that verifying $B.2$ can be performed in time $O(|E| * |S|)$. Now, to calculate the set $Remove(s_2)$ and $Remove^+(s_2)$ of lines 3-4, we need to compute the transitive closure of $\rightarrow$ which can be done in time $O(|S|^3)$. Since this set is calculated once per each state we have that lines 1-5 take time $\mathcal{O}(|S|^4)$ and $NonMaskB3B4$ is calculated in $\mathcal{O}(|S|^2 * |AP'|)$. On the other hand, as explained in the proof of Theorem 4.3, lines 6-33 take time $\mathcal{O}(|E| * |S|)$. Thus, in the worst case the time complexity of the algorithm is $\mathcal{O}(|S|^4 + |S|^2 * |AP'|)$. □

4.3. Computing Failsafe Fault-Tolerance

We finally present Algorithm 6 for computing a relation that satisfies the condition $B.3$ and $B.4$ of Definition 3.6 for a colored Kripke Structure $M = \langle S, I, R, L, \mathcal{N} \rangle$ and a sub-labeling $L_0 \subseteq L$. The scheme of this algorithm is similar to that of Algorithm 3. This is because both masking and failsafe fault-tolerance require that the safety properties have to be guaranteed. However, liveness properties are not necessarily preserved in failsafe tolerance. The main difference with Algorithm 3 is in line 18, where we allow the faulty system to stay in a safe set of states. In other words, in the case that the simulation relation is not preserved, the system has the option to stay in a set of safe states. To this end, we introduce a set $Eq(s)$ containing the closure of reachable states with the same labeling. In the case that there exists some state reachable from the origin with a different labeling, we set $Eq(s)$ to empty.

Let us describe more in details Algorithm 6. In the first for loop, for each $s_2 \in S$, the set $FailsafeB3B4(s_2)$ contains those normal states that are candidates for simulating s_2 which is initialized with all normal states with the same labels as s_2. Moreover, the set $Remove(s_2)$ contains all the normal states which do not have successor states simulating some successor state from s_2 and we also initialized the set $Eq(s_2)$ as explained above. The termination condition of the loop of lines 6-27 is $Remove(s_2') = \emptyset$ for all $s_2' \in S$, in that case there are no normal states that need to be removed from the sets of simulators $FailsafeB3B4(s_2)$ for $s_2 \in Pre_N(s_2')$ or $s_2 \in Pre_F(s_2')$. Within the while-loop body, we take care of normal transitions ($B.3$) and faulty transitions ($B.4$) of Definition 3.6. Firstly, in the for loop of lines 8-16, we compute condition $B.3$ in the same way that is computed in Algorithm 3 in lines 7-16. Secondly, in the for loop of lines $17-25$, we now consider all pairs $(s_1, s_2) \in S \times S$ such that $s_1 \in Remove(s_2')$ and $s_2 \in Pre_F(s_2')$. This means that $s_2 \dashrightarrow s_2'$ but there is no transition $s_1 \rightarrow s_1'$ with $s_1' \in FailsafeB3B4(s_2')$ because we have that $s_1 \in Remove(s_2')$. Moreover, we check that the set $Eq(s_2)$ is empty, i.e., for a faulty successor s_2' of s_2, there does not exist an state s with $s_2' \Rightarrow^* s$ and $L_0(s_2) = L_0(s)$. This yields that s_2 is not simulated by s_1. Therefore, s_1 is removed from $FailsafeB3B4(s_2)$ if $s_1 \in FailsafeB2B3(s_2)$ and $Eq(s_2) = \emptyset$, that is, the last part of condition $B.4$. Subsequently, we add to the set $Remove(s_2)$ all normal predecessors s of s_1 such that $Post_N(s) \cap FailsafeB3B4(s_2) = \emptyset$, where we also check that $s \notin FailsafeB3B4(Pre_F(s_2)) \vee FailsafeB3B4(Eq(Pre_F(s_2))) \neq \emptyset)$, that is, the last part of condition $B.4$. Finally, we combine the sets $SimB2$ and $FailsafeB3B4$ through Algorithm 4 by instantiating the parameter set $SimRel1$ and $SimRel2$ with $SimB2$ and $FailsafeB3B4$, respectively.

The proofs of correctness and termination are similar to Theorem 4.1 and Lemma 4.2, with some minor changes.

Theorem 4.6 (Partial Correctness of Failsafe). On termination, Algorithms 2 and 6 return **F**.

Algorithm 6 Computation of condition $B.3$ and $B.4$ of Defintion 3.6 ($ComputeFailsafeB3B4(M, L_0)$)

Input: Colored Kripke structure $M = \langle S, I, R, L, \mathcal{N} \rangle$ and a sub-labeling $L_0 \subseteq L$
Output: Relation $FailsafeB3B4$ where the states in it satisfy condition $B.3$ and $B.4$ of Definition 3.6

1: **for all** $s_2 \in S$ **do**
2: $FailsafeB3B4(s_2) := \{s_1 \in \mathcal{N} \mid L_0(s_1) = L_0(s_2)\}$
3: $Remove(s_2) := \mathcal{N} \backslash Pre_N(FailsafeB3B4(s_2))$
4: $Eq(s_2) := \{s \mid s_2 \Rightarrow^* s \wedge L_0(s_2) = L_0(s) \wedge (\nexists s' : s_2 \Rightarrow^* s' \wedge L_0(s) \neq L_0(s'))\}$
5: **end for**
6: **while** $\exists\ s_2' \in S$ with $Remove(s_2') \neq \emptyset$ **do**
7: select s_2' such that $Remove(s_2') \neq \emptyset$
8: **for all** $s_1 \in Remove(s_2')$ **do**
9: **for all** $s_2 \in Pre_N(s_2')$ **do**
10: **if** $s_1 \in FailsafeB3B4(s_2)$ **then**
11: $FailsafeB3B4(s_2) := FailsafeB3B4(s_2) \backslash \{s_1\}$
12: **for all** $s \in Pre_N(s_1)$ with $Post_N(s) \cap FailsafeB3B4(s_2) = \emptyset$ **do**
13: $Remove(s_2) := Remove(s_2) \cup \{s\}$
14: **end for**
15: **end if**
16: **end for**
17: **for all** $s_2 \in Pre_F(s_2')$ **do**
18: **if** $s_1 \in FailsafeB3B4(s_2) \wedge Eq(s_2) = \emptyset$ **then**
19: $FailsafeB3B4(s_2) := FailsafeB3B4(s_2) \backslash \{s_1\}$
20: **for all** $s \in Pre_N(s_1)$ with $Post_N(s) \cap FailsafeB3B4(s_2) = \emptyset \wedge (s \notin FailsafeB3B4(Pre_F(s_2)) \vee FailsafeB3B4(Eq(Pre_F(s_2))) \neq \emptyset)$ **do**
21: $Remove(s_2) := Remove(s_2) \cup \{s\}$
22: **end for**
23: **end if**
24: **end for**
25: **end for**
26: $Remove(s_2) := \emptyset$
27: **end while**
28: **return** $\{\langle s_1, s_2 \rangle \mid s_1 \in FailsafeB3B4(s_2)\}$

Proof. We have to prove that Algorithm 6 ensures conditions $B.3$ and $B.4$ of Definition 3.6. As explained above, condition $B.2$ is computed using Algorithm 2, which is correct by Theorem 4.1. For the other conditions, first note that the loop of line 6 has the following loop invariant. For all states $s_2 \in S$:

(a) $Remove(s_2) \subseteq (\mathcal{N} \backslash Pre_N(FailsafeB3B4(s_2)))$
(b) for any relation $FailsafeB3B4$: $\{s_1 \in \mathcal{N} \mid s_1\ FailsafeB3B4\ s_2\} \subseteq FailsafeB3B4(s_2) \subseteq \{s_1 \in \mathcal{N} \mid L_0(s_1) = L_0(s_2)\}$
(c) $\forall s_1 \in FailsafeB3B4(s_2)$, either:
  (1) $\exists s_2' \in Post_N(s_2)$ with $Post_N(s_1) \cap FailsafeB3B4(s_2') = \emptyset$ and $s_1 \in Remove(s_2')$,
  (2) $\exists s_2' \in Post_F(s_2)$ with $Post_N(s_1) \cap FailsafeB3B4(s_2') = \emptyset \wedge Eq(s_2) = \emptyset$ and $s_1 \in Remove(s_2')$,
  (3) $\forall s_2' \in Post_N(s_2) : Post_N(s_1) \cap FailsafeB3B4(s_2') \neq \emptyset$,
  (4) $\forall s_2' \in Post_F(s_2) : Post_N(s_1) \cap FailsafeB3B4(s_2') \neq \emptyset \vee s_1 \in FailsafeB3B4(s_2') \vee Eq(s_2) \neq \emptyset$

From (c), we obtain that, when $Remove(s_2') = \emptyset$ for every $s_2' \in S$, then $\forall s_2 \in S : \forall s_1 \in FailsafeB3B4(s_2)$:

- $\forall s_2' \in Post_N(s_2) : Post_N(s_1) \cap FailsafeB3B4(s_2') \neq \emptyset$, and
- $\forall s_2' \in Post_F(s_2) : Post_N(s_1) \cap FailsafeB3B4(s_2') \neq \emptyset \vee s_1 \in FailsafeB3B4(s_2') \vee Eq(s_2) \neq \emptyset$.

That is, the relation defined as $s_1\ FailsafeB3B4\ s_2$ satisfies conditions $B.3$ and $B.4$, respectively, of Definition 3.6. Finally, the proof that procedure $MergeSimRel$ merges sets $SimB2$ and $FailsafeB3B4$ in a correct way is similar to the proof given for the masking relation $\mathbf{M}$. □

Normal Actions:

$$x = z \wedge y = z \rightarrow x, y := \neg x, \neg y$$
$$z \neq maj(x, y, u) \wedge u \neq maj(x, y, u) \rightarrow z, u := maj(x, y, u), maj(x, y, u)$$

Fig. 8. The Muller C-element program with majority voting (fault-intolerant version).

Normal and Recovery Actions:

$$x = z \wedge y = z \rightarrow x, y := \neg x, \neg y$$
$$z \neq maj(x, y, u) \wedge u \neq maj(x, y, u) \rightarrow z, u := maj(x, y, u), maj(x, y, u)$$
$$x = z \wedge y \neq z \rightarrow x := \neg x$$
$$x \neq z \wedge y = z \rightarrow y := \neg y$$

Faulty Actions:

$$x = z \wedge y = z \rightarrow x := \neg x$$
$$x = z \wedge y = z \rightarrow y := \neg y$$
$$u = z \rightarrow z := \neg u$$

Fig. 9. The Muller C-element fault-tolerant program with majority.

With respect to the complexity of the failsafe fault-tolerance relation **F**, a similar proof as for Algorithm 3 shows that it also has a polynomial complexity.

Theorem 4.7 (Complexity of Failsafe). The time complexity of Algorithm 6 is in $\mathcal{O}(|E| * |S| + |E| * |AP'|)$.

5. Illustrating Examples

In order to show the suitability of our approach to reason about fault-tolerance, we apply the ideas presented above to four examples of typical fault-tolerant systems.

5.1. The Muller C-element

The Muller C-element [MC80] is a simple delay-insensitive circuit which contains two boolean inputs and one boolean output. Its logical behavior is described as follows: if both inputs are true (resp. false) then the output of the C-element becomes true (resp. false). If the inputs do not change, the output remains the same. Ideally, the inputs change simultaneously. In [AG93], the following (informal) specification of the C-element with inputs x and y and output z is given:

(i) Input x (resp. y) changes only if $x \equiv z$ (resp., $y \equiv z$), (ii) Output z becomes true only if $x \wedge y$ holds, and becomes false only if $\neg x \wedge \neg y$ holds; (iii) Starting from a state where $x \wedge y$, eventually a state is reached where z is set to the same value that both x and y have. Ideally, both x and y change simultaneously. Faults may delay changing either x or y.

We consider an implementation of the C-element with a majority voting circuit involving three inputs, where an extra input u to the circuit is added. Then, the predicate $maj(x, y, u)$ returns the value of the majority circuit, which is assumed to work correctly, and is defined as $maj(x, y, u) = (x \wedge y) \vee (x \wedge u) \vee (y \wedge u)$. In addition to the traditional logical behavior of the C-element, u and z have to change at the same time, where the output z is fed back to the input u.

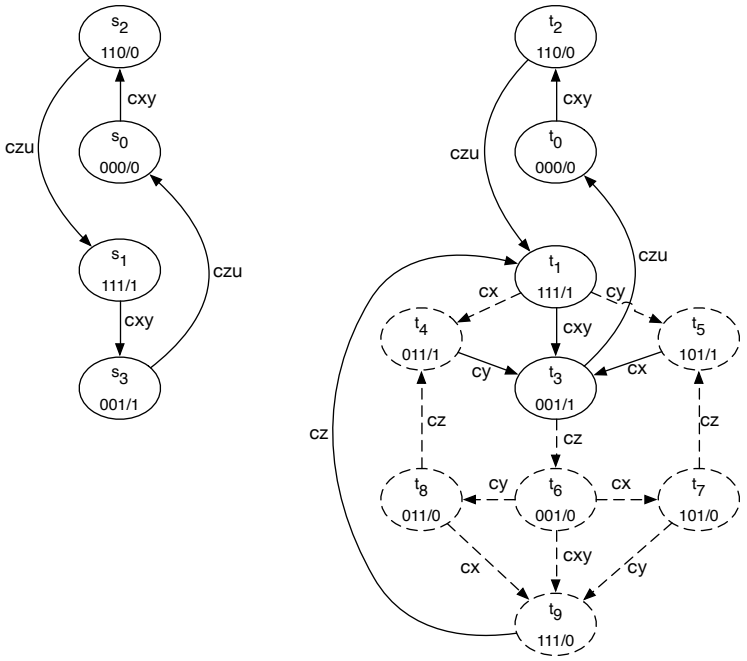

Fig. 10. A nonmasking fault-tolerance for the Muller C-element with a majority circuit.

Figure 10 shows two models of this circuit. M exhibits the ideal behavior of the C-element containing only normal transitions. M' takes into account the possibility of faults occurring, and provides a reaction to these. Every state in these models is composed of boolean variables x, y, u, and z, where x, y, and u represent the inputs, and z represents the output. For instance, the state s_0 contains the information $000 \setminus 0$ interpreted (reading from left to right) as $x = 0$, $y = 0$, $u = 0$, and $z = 0$.

For the sake of clarity, we have labelled the transitions with subsets of the set $\{cx, cy, cu, cz\}$ of actions; action cx (resp., cy and cu) is the action that changes input x (resp., y and u); cz is the action of changing output z. When the actions cx and cy are executed in the same transition, we just write cxy. It is important to remark that these action names do not belong to the corresponding formal model (note that colored Kripke structures do not consider labels in the transitions), though one may enrich the states with more data (boolean variables) to indicate the occurrence of actions.

We consider two types of faults: *(i)* a delay may occur in the arrival of some of the inputs x or y (i.e., they do not change simultaneously), and *(ii)* a delay in the signal from z to u occurs. We can observe these classes of faults in the faulty states (indicated by dashed circles) when either x and y or u and z do not match one another. Notice that these two models described in Figure 10 correspond to the fault-intolerant and the fault-tolerant program in Figures 8 and 9 in a guarded command style, respectively.

The relation $R_{c-element} = \{\langle s_0, t_0\rangle, \langle s_1, t_1\rangle, \langle s_2, t_2\rangle, \langle s_3, t_3\rangle\}$ is a nonmasking fault-tolerant relation for $\langle M, M'\rangle$ and the sub-labelings obtained by restricting the original labelings to letters u, x, y, z. Therefore, when the majority circuit behaves correctly, this implementation tolerates delays of inputs.

5.2. The Byzantine Generals Problem

An interesting example of a fault-tolerant system is the Byzantine generals problem, introduced originally in [LSP82]. This is an agreement problem, where we have a general with $n-1$ lieutenants. The communication between the general and his lieutenants is performed through messengers. The general may decide to attack an enemy city or to retreat; then, he sends the order to his lieutenants. Some of the lieutenants might be traitors. As a consequence, traitors might deliver false messages or perhaps they avoid sending a message that they received. The loyal lieutenants must agree on attacking or retreating after $m+1$ rounds of communication,

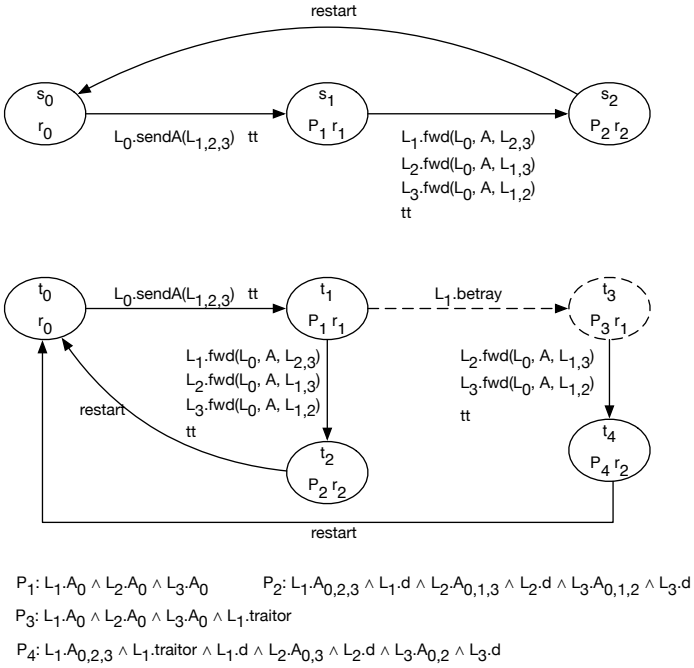

Fig. 11. A masking fault-tolerance relation for the Byzantine generals problem.

where m is the maximum numbers of traitors. The algorithm can ensure correct operation only if fewer than one third of the lieutenants are traitors. We assume the following: $\mathsf{L_0}$ is the general, the messages are delivered correctly and all the lieutenants can communicate directly with each other; in this scenario they can recognize who is sending a message. Faults can convert loyal lieutenants into traitors. Finally, traitors cannot forge messages on behalf of loyal lieutenants.

In Figure 11 two models of this problem are described. M exhibits the ideal behavior of the Byzantine agreement for a general ($\mathsf{L_0}$) and three loyal lieutenants ($\mathsf{L_1}$, $\mathsf{L_2}$, and $\mathsf{L_3}$). On the other hand, M' expresses the same behavior that M does, but considering the presence of lieutenant $\mathsf{L_1}$ as a traitor. We specify this problem following the ideas introduced in [CM09]. We have the following predicates: $\mathsf{L_i.A_{l_1,\ldots,l_n}}$ (this predicate indicates that $\mathsf{L_i}$ has received a message from lieutenants $\mathsf{l_1}, \ldots, \mathsf{l_n}$ saying that he must attack). We have a violation predicate $\mathsf{L_i.traitor}$ for each lieutenant (this predicate is true when $\mathsf{L_i}$ is a traitor) and $\mathsf{L_i.d}$ (this predicate is true when $\mathsf{L_i}$ has decided to attack), $\mathsf{r_j}$ (this predicate is true when we are in round j). Each state has the predicates ($\mathsf{P_1}$, $\mathsf{P_2}$, $\mathsf{P_3}$, or $\mathsf{P_4}$) which hold in it and the relevant information about the current round. Each transition is labeled with some of the following actions: $\mathsf{L_i.sendA(L_{l_1,\ldots,l_n})}$ (lieutenant $\mathsf{L_i}$ sends to lieutenants $\mathsf{l_1}, \ldots, \mathsf{l_n}$ the message of attack), $\mathsf{L_i.fwd(L_k, A, L_{l_1,\ldots,l_n})}$ (the lieutenant $\mathsf{L_i}$ forwards to lieutenants $\mathsf{l_1}, \ldots, \mathsf{l_n}$ the message of attack that he received from $\mathsf{L_k}$), $\mathsf{L_i.betray}$ (lieutenant $\mathsf{L_i}$ becomes a traitor). We consider a clock that allows lieutenants to synchronize; the action tt increments the clock by one unit of time. The specification uses $\mathsf{m}+1$ rounds of messages, which are coordinated by means of the clock, where m is the number of traitors for which the specification ensures that the loyal lieutenants will agree on a decision. In this case the relation

$$R_{byzantines} = \{\langle s_0, t_0\rangle, \langle s_1, t_1\rangle, \langle s_2, t_2\rangle, \langle s_1, t_3\rangle, \langle s_2, t_4\rangle\}$$

is a masking fault-tolerant relation for $\langle M, M'\rangle$ where the sub-labelings are obtained by restricting the

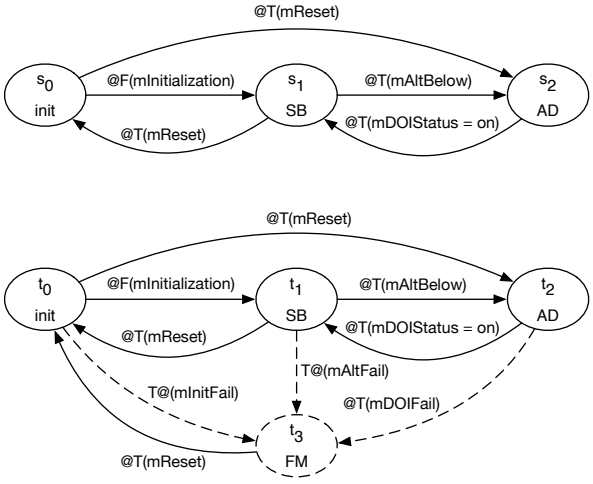

Fig. 12. A nonmasking fault-tolerance for the Altitude Switch Controller.

original labeling to $AP \setminus \{\mathsf{L}_0.\mathsf{traitor}, ..., \mathsf{L}_n.\mathsf{traitor}\} \cup \{\mathsf{L}_0.\mathsf{A}_{\mathsf{l}_1,...,\mathsf{l}_n}, \ldots \mathsf{L}_n.\mathsf{A}_{\mathsf{l}_1,...,\mathsf{l}_n}\}$; this means that the model on the bottom tolerates the existence of one traitor (specifically, L_1). This can be generalized to support more lieutenants and traitors.

5.3. Altitude Switch (ASW)

The Altitude Switch (ASW) controller in an avionics system is responsible for turning on a Device of Interest (DOI) when the aircraft altitude is below a pre-specified threshold. We have adapted this real-world avionics example from [JHAL09].

Basically, the ASW controller reads a set of variables and produces an output. There exist four internal variables, a mode variable that determines the operating mode of the system, and four monitored variables that represent the state of the altitude sensors. The monitored variables are as follows: (1) *mAltBelow* is true if the altitude is below a pre-specific threshold. (2) *mDOIStatus* is true if the DOI is powered on; (3) *mInhibit* is true when the DOI power-on is inhibited, and (4) *mReset* is true if the system is being reset. The ASW program can be in three different modes: (1) the *Initialization (Init)* mode when the ASW system is initializing; (2) the *AwaitDOIon (AD)* mode if the system is waiting for the DOI to power on, and (3) the *Standby (SB)* mode for all other cases.

The ASW system can be subject to hardware malfunctions that may alter the ASW controller. In order to deal with potential faults, the system is designed to tolerate three time-out faults: (1) initialization fault, (2) altimeter fault, and (3) DOI fault. These types of faults require the system to stay in a given state for a specific amount of time. Because we do not include the notion of time in this example, we model these faults as on/off flags. As a consequence of these malfunctions, a new mode, *Fault (FM)*, is added to the mode class to indicate the presence of faults in the system. Figure 12 shows two models of this program. M exhibits the ideal behavior of the ASW controller containing only normal transitions. M' takes into account the possibility of faults occurring, and provides a reaction to these. Every state in these models identifies some of the modes: *Init*, *AD*, *SB*, or *FM*. Transitions are labeled by events that represent the changes among the different modes. Let us note that these events can be encoded in the colored Kripke structures with fresh propositional variables; for instance, in state s_0 one can have a variable *mReset*, which is true when the corresponding event has occurred and then a transition to state s_2 is enabled, this mechanism can be reflected by adding more intermediate states (representing the changes in the events); the model we present

in Figure 12 is a simplified one (one can think of it as an abstract description of the corresponding model); however, it illustrates how the notion of nonmasking applies to this scenario.

We use the following notation to represent events: *@T(c)*, where c represents the condition value in the before state. Once the event *@F(mInitialization)* occurs, which means that the initialization is complete, the system moves from *Init* to *SB*. It returns to *Init* when the pilot pushes the *reset* button (*@T(mReset)* occurs). Moreover, the system moves from *SB* to *AD* through the action *altBelow* when the aircraft descends below the threshold altitude (*@T(mAltBelow)*), but requiring as precondition that powering on is not inhibited, and the *DOI* is not powered on. Once the *DOI* signals that it is powered on *T(mDOIStatus = on)*, the system goes from *AD* to *SB*. Furthermore, the system returns from *AD* to *Init* when the pilot pushes the *reset* button (*@T(mReset)* occurs). All these events correspond to the ideal behavior of the ASW program. On the other hand, we add other monitoring variables *mInitFail*, *mAltFail*, and *mDOIFail* to model the occurrences of faults, perturbing the normal behavior of the system. Consequently, when the system *detects* any of these faults, it goes into the faulty mode (*FM*). Finally, the problem specification requires that the program does not change its mode from *Standby* (SB) to *AwaitDOI* (AD) if the altitude sensors failed, i.e., when *@T(mAltFail)* occurs. Moreover, from the faulty state *FM*, the program can only go into the *Initialization* mode. In fact, the program can recover from the *faulty* state if the system has been reset by the pilot (*@T(mReset)* occurs).

As a result, the relation $R_{asw-controller} = \{\langle s_0, t_0\rangle, \langle s_1, t_1\rangle, \langle s_2, t_2\rangle, \langle s_0, t_3\rangle\}$ is a nonmasking fault-tolerant relation for $\langle M, M'\rangle$. We obtain a similar result compared to [JHAL09], where the authors use the term eventual masking ("*the system enters a fault handling state but eventually recovers to normal behavior*") for this kind of fault-tolerance behavior. We remark that the specification on this example has been slightly simplified from the original one described in [JHAL09]. For example, the logic in the events that cause transitions in our models is much simpler compared to the ones in [JHAL09].

5.4. A Simple Train System

We now consider a simple train system. Train systems control the movement of trains through a network of rail segments. Fault-tolerance is a key aspect of these systems: a fault in the system may cause a train collision and the loss of human life. These kinds of systems are the object of active research in the fault-tolerance community (see [AB08, Abr06, GH90]).

The general version of our system consists of n trains and m rail segments. Rail segments are connected to other rail segments, where in each of these connections the rails are equipped with a signal which indicates if the segment is occupied or not. The signals can be green (when the segment is free) or red (when another train is in the segment). We have the following predicates. For each $0 \leq i \leq n$ and $0 \leq j \leq m$, we have a predicate $t_i.r_j$ which indicates that train i is in the segment j; the predicate $r_j.green$ ($r_j.red$) expresses that the signal of segment j is green (red), respectively. Moreover, $t_i.stop$ denotes that train t_i is stopped and $r_i R r_j$ means that segments i and j are connected. We have the following actions: $t_i.move(j)$: train t_i moves to segment j, $t_i.stops$: train t_i stops, $r_i.ggreen$: the signal of r_i is set to green, and $r_i.gred$: the signal of r_i is set to red.

The specification requires that, if there is a train in a segment, then the signal for this segment must be red. On the contrary, if there is no train in the segment, then the signal for the segment must be green. Moreover, when a train detects that another train is already in the same segment, a fault has occurred, and it ought to stop to avoid train collisions. Besides, when a train is in a segment where all the connected segments have their signals set to red, the train will stop. Finally, an ideal safety property of this system is that there are not two trains in the same segment; that is, this property must hold in the specification when no faults are observed.

This system is implemented by a sensor in each segment which detects the existence of two trains. Furthermore, the movement of trains between rail segments is controlled by a pair of sensors called *Block Instruments* in railways. Essentially, these devices have the task of activating the green or red signal. The basis of operation of the system is very simple, but what is more significant is the inherent fail-safety built into it. In case a fault occurs, the interlocking relay logic turns the signal to red.

In order to provide a better understanding of this example and the particular situation for failsafe fault-tolerance, let us consider trains t_i with $0 \leq i \leq 2$ and rail segments r_j with $0 \leq j \leq 4$, where they are connected as follow: $r_0 R r_1$, $r_1 R r_2$, $r_2 R r_3$, $r_2 R r_4$, and $r_3 R r_4$. In Figure 13 two models of this problem are described. M exhibits the ideal behavior of the simple train system for three trains on five rail segments. On

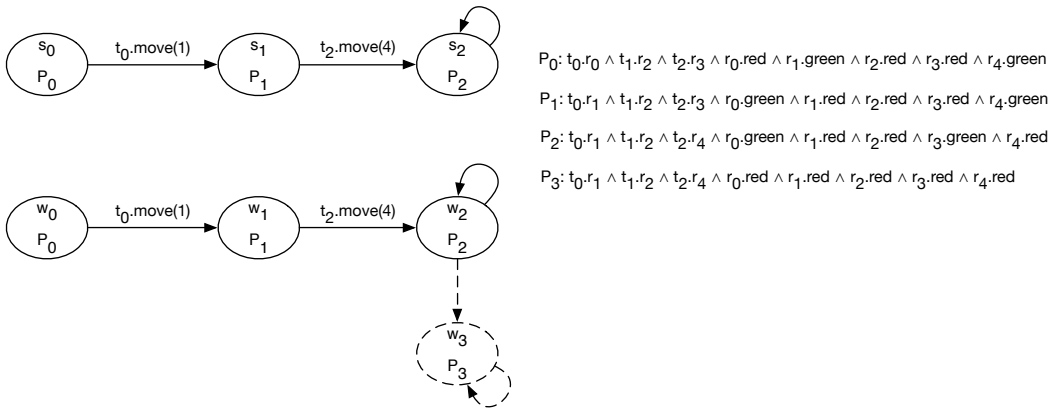

Fig. 13. A failsafe fault-tolerance for a Simple Train System.

the other hand, M' expresses the same behavior that M does, but considering the presence of faults. Notice that in this figure we illustrate part of the behavior for this set of train and rail segments, starting from a state s_0 in which trains are already located on the rail segments. The states are labeled with predicates (P_0, P_1, P_2, or P_3) pointing out the formulas holding in each state. Moreover, transitions are labeled with some of the actions described above. We remark that the models in Figure 13 are partial, covering only a specific scenario of normal behavior to describe a simple situation for failsafe fault-tolerance. Note also the self-loop in state s_2; we assume that in this case the trains have reached their destination. Of course, a complete model (which may have millions of states) will have more states allowing the trains to start their itinerary again. Regarding the model M', faults may occur that affect the communication of the system or the behavior of the sensors. The block instruments on each rail segment react to this malfunction, turning the signal to red (P_3). It is clear that this state still guarantees the (safety) property that no two trains are located in the same segment. However, reaching state w_3 means that all trains will stop and they will stay there (represented by the self-loop on w_3) until an external recovery action is performed. In this case the relation

$$R_{train-system} = \{\langle s_0, w_0\rangle, \langle s_1, w_1\rangle, \langle s_2, w_2\rangle\}$$

is a failsafe fault-tolerant relation for $\langle M, M'\rangle$ where the sub-labelings are obtained by restricting the original labeling to $t_i.r_j$ with $0 \leq i \leq 2$ and $0 \leq j \leq 4$.

Notice that a fault leading to a failsafe state is not considered as a failure in a safety critical system. The main idea of a failsafe design is to detect the faults and mask its impact until some recovery actions are undertaken.

6. Related Work

Our work is closely related to several formal approaches to fault-tolerance. The programming style we use here for describing our programs was introduced in [AG93, AK98a, AK98b, Gär99], where programs are written using Dijkstra's guarded command language, and faults are specified as distinguished actions. In these works, the authors characterize fault tolerance using sets of states, for instance, a set P of states captures the desired invariant of the program, whereas a set T is employed to indicate those states that tolerate faults. In this setting, masking, nonmasking and failsafe tolerance are formalized making use of liveness and safety properties, where properties in general are written using first-order logic; no temporal operators are employed in these works. In our opinion, temporal logic and the restriction to finite state systems provide key benefits when verifying concurrent and reactive programs, in particular with respect to automatic verification, where the model checking community has demonstrated relative success when verifying hardware and embedded systems [BK08, Cla99]. Also it is important to remark that, in the cited works, the authors assume a linear view of time when specifying systems and properties. In our approach, we focus on branching time properties of programs. As pointed out by several authors [FMR07, AAE04, EL09, BFG02, YTK01], branching time

is useful for capturing important properties related to fault-tolerance. A simple example of this is given in [AAE04] by means of the formula: $\mathsf{AG}(D_i \Rightarrow \mathsf{EG}D_i)$, which intuitively expresses that: *if process i crashes, it may stay down forever*; this property cannot be expressed in LTL, for instance. In [BFG02], ACTL (a branching time logic with actions) is used for expressing properties over models, where systems are specified by means of process algebras; in that work the authors concentrate on model checking fault-tolerant systems, the different levels of tolerance are not investigated by the authors.

In [JHAL09, JHAL10] is presented a model-based method for specifying and verifying the required behaviour of a fault-tolerant system. The formal foundations of their method relies on formal notions like *or-composition*, *partial refinement*, and *fault-tolerant extension*. The authors covers two classes of fault-tolerance: *transparent masking* [JHAL09] and *partial masking* [JHAL10]; these can be related to the notion of *masking* and *nonmasking* fault-tolerance presented in our paper, respectively. As it is stated by the authors, in *transparent masking*, the component's fault-tolerant behaviour is a *full refinement* of its normal behaviour. On the other hand, in *partial masking*, the system behaviour is a *partial refinement* due to the refinement holds during the normal system behaviour but may not hold during fault-handling. It is well-known that simulation relations are used in theoretical computer science in order to formally stablish notions of modelling abstraction and refinement in hierarchical systems. Clearly, the goal of the authors in [JHAL09, JHAL10] is different to our aim because they introduce a formal method for building fault-tolerant systems. But, we think that our approach can be adapted (fully coverage of the notion of *transparent* and *partial masking*) to support and complement their method by checking that the constructed fault-tolerant extension is correct for the required behaviour of a fault-tolerant system.

Another interesting approach is that of retrenchment [BP98, BPJS07], as opposed to conventional refinement, retrenchment can provide a formal account of the relationship between a nominal model and an extended model enriched with the envisaged faults the system is designed to be robust against. Some approaches that exploit this technique are described in [BB06, BC04], where fault trees are mechanistically extracted from the simulation relation of a suitable retrenchment, designed to capture the difference between an "ideal" and a "faulty" behaviour of a system. These works rely on a particular stepwise simulation notion called punctuation simulation for retrenchment [BP99], which is approached in a different manner to refinement simulation. In our work, we also present different variations of standard simulation relations but we focus on comparing the executions of a system that exhibits faults with executions where no faults are present, for capturing specific levels of fault-tolerance.

We can also mention the work presented in [Jan95, Jan97], where various notions of bisimulation are investigated with the aim of capturing fault-tolerant properties, in the context of process algebras. A related approach is described [FH07], where the authors present a theoretical framework using π-calculus to capture n-fault-tolerance in a distributed setting (i.e., the tolerance to n faulty nodes). An obvious difference w.r.t. our work is that we use a state based approach and a temporal logic to reason about state based models, in contrast to the aforementioned works where process algebras are employed for modeling systems, and the associated logic is a variation of Hennesy-Milner logic (HML). HML is a simple logic with formulas of the style $\langle act_i \rangle \varphi$, where act_i represents and action, φ is another formula and $\langle act_i \rangle$ is diamond modality of modal logic. HML can only express properties about a finite number of steps into the future. For instance, reachability properties (of the style $\mathsf{AF}\varphi$ or $\mathsf{EF}\varphi$) cannot be captured with HML; that is, this logic is not expressive enough to capture important temporal properties [BS07]. Also, the notions of masking, nonmasking and failsafe fault-tolerance are not investigated in these works. Finally, it is important noting that preliminary definitions of the simulation relations described here were discussed in [DCMA13a]. In that paper we do not prove the properties of the introduced relations, and no algorithms were given for computing nonmasking and failsafe tolerance. Additionally, the relation of failsafe given in that paper is more restrictive that the one given here; in contrast to the definition of [DCMA13a], where a unique safe state was permitted, the failsafe relation presented in Section 3 allows the system to stay in a set of safe states, thus generalizing the original definition.

7. Final Remarks

We have presented a characterization of different levels of fault-tolerance by means of simulation relations. This formalization is simple and uses standard notions of simulation relations, by relating an operational system specification and a corresponding fault-tolerant implementation. Moreover, our approach to capturing fault-tolerance enables us to automatically verify, for example, that a given implementation of a system masks

certain faults, or recovers from these faults, by employing variants of traditional bisimulation algorithms applied to our context. Indeed, we have adapted well known (bi)simulation algorithms to our setting, so that one can automatically check if a system implementation exhibits some degree of fault-tolerance. We have also studied the complexity of the resulting algorithms, and proved that they preserve the time complexity of traditional bisimulation algorithms. We have also studied properties of our formalizations of fault-tolerance, showing that different kinds of temporal properties are preserved, depending on the degree of fault-tolerance that a system exhibits. Moreover, we have also presented results relating the different kinds of fault-tolerance.

It is interesting to discuss whether it is convenient to use simulation relations instead of model checking when verifying fault-tolerance. Note that algorithms for model checking CTL have complexity $O(|E| * |\phi|)$ [BK08], where $|\phi|$ is the size of the formula to be verified; since formulas are usually much shorter than models, model checking a given fault tolerant property ϕ should be more efficient than using the algorithms described here. However, note that simulation relations cannot be expressed in CTL, that is, we cannot employ model checking to prove that a faulty implementation exhibits fault-tolerant behavior (i.e., it preserves the properties of the original nonfaulty system), instead we could use a model checker to prove that some property is preserved by the faulty implementation; however, in this case we should codify faults using CTL constructs which could cumbersome both the formula to be verified and the model describing the system. We plan to implement symbolic versions of the algorithms given in Section 4 to analyze their performance in practice. Also, it is interesting to point out that the ideas presented here admit other interesting applications, for instance, we have been exploring the use of these ideas to automatically construct fault-tolerant programs from specifications (this is called program synthesis). Synthesis of programs has been extensively investigated in the context of linear time logic [PR89], while in our case it is necessary to deal with a branching time formalism. As argued in [AAE04], some important properties related to fault-tolerance require branching time operators; in that paper, a framework for synthesis of programs from branching time specifications is introduced. We have used the notion of masking relation presented here to extend the algorithm introduced in [AAE04], to automatically synthesize programs that mask faults. The results obtained in this line of research are detailed in [DCMA13b] where an algorithm for synthesizing masking fault-tolerant programs is presented, we have implemented this algorithm, the resulting tool is described in [DCR$^+$15]. As future work, we plan to extend our tool to synthesize nonmasking and failsafe fault-tolerant programs based on the corresponding relations presented in this paper. Finally, we also plan to extend our framework to accommodates multitolerance. Arora and Kulkarni formalized this concept in [AK98a]. Essentially, in multitolerance, the set of fault actions is divided into classes, and different fault classes may require diverse levels of fault-tolerance (masking, nonmasking, or failsafe). For example, one class of faults may require masking fault-tolerance, while another class may demand only failsafe fault-tolerance. Thereby, "multitolerance refers to the ability of a system to tolerate multiple classes of faults, each in a possibly different way". Then, we could define an algorithm in order to support multitolerance by computing the required level of fault-tolerance for each particular fault class.

Acknowledgements

This work was partially supported by the APC (NSERC) funded project NECSIS and IBM through an IBM CAS Fellowship to the first author. It was also partially supported by the Argentinean Agency for Scientific and Technological Promotion (ANPCyT), through grants PICT 2012 No. 1298 and PICT 2013 No. 0080; and by the MEALS project (EU FP7 programme, grant agreement No. 295261).

References

[AAE04] Paul C. Attie, Anish Arora, and E. Allen Emerson. Synthesis of fault-tolerant concurrent programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 26(1):125–185, 2004.

[AB08] Zair Abdelouahab and Reginaldo I. Braga. An adaptive train traffic controller. In *An Adaptive Train Traffic Controller, Springer Netherlands*, pages 550–555, 2008.

[ABK04] Benjamin Aminof, Thomas Ball, and Orna Kupferman. Reasoning about systems with transition fairness. In Franz Baader and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 11th International Conference, LPAR 2004*, volume 3452 of *Lecture Notes in Computer Science*, pages 194–208. Springer, 2004.

[Abr06] Jean-Raymond Abrial. Train systems. In Michael J. Butler, Cliff B. Jones, Alexander Romanovsky, and Elena

Troubitsyna, editors, *Rigorous Development of Complex Fault-Tolerant Systems [FP6 IST-511599 RODIN project], RODIN Book*, volume 4157 of *Lecture Notes in Computer Science*, pages 1–36. Springer, 2006.

[Abr10] Jean-Raymond Abrial. *Modeling in Event-B - System and Software Engineering.* Cambridge University Press, 2010.

[AG93] Anish Arora and Mohamed G. Gouda. Closure and convergence: A foundation of fault-tolerant computing. *IEEE Transactions on Software Engineering*, 19(11):1015–1027, 1993.

[AK98a] Anish Arora and Sandeep S. Kulkarni. Component based design of multitolerant systems. *IEEE Trans. Software Eng.*, 24(1):63–78, 1998.

[AK98b] Anish Arora and Sandeep S. Kulkarni. Detectors and Correctors: A theory of fault-tolerance components. In *18th International Conference on Distributed Computing Systems, ICDCS 1998*, pages 436–443. IEEE Computer Society, 1998.

[Avi95] Algirdas A. Avizienis. *Software Fault Tolerance*, volume 2, chapter The Methodology of N-Version Programming, pages 22–45. John Wiley & Sons, 1995.

[BB06] Richard Banach and Marco Bozzano. Retrenchment, and the generation of fault trees for static, dynamic and cyclic systems. In *Computer Safety, Reliability, and Security, 25th International Conference, SAFECOMP 2006, Gdansk, Poland, September 27-29, 2006, Proceedings*, pages 127–141, 2006.

[BC04] Richard Banach and R. Cross. Safety requirements and fault trees using retrenchment. In *Computer Safety, Reliability, and Security, 23rd International Conference, SAFECOMP 2004, Potsdam, Germany, September 21-24, 2004, Proceedings*, pages 210–223, 2004.

[BFG02] Cinzia Bernardeschi, Alessandro Fantechi, and Stefania Gnesi. Model checking fault tolerant systems. *Softw. Test., Verif. Reliab.*, 12(4):251–275, 2002.

[BK88] Ralph-Johan Back and Reino Kurki-Suonio. Distributed cooperation with action systems. *ACM Trans. Program. Lang. Syst.*, 10(4):513–554, 1988.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008.

[Bon08] Borzoo Bonakdarpour. *Automated Revision of Distributed and Real-Time Programs.* PhD thesis, Michigan State University, 2008.

[BP98] Richard Banach and Michael Poppleton. Retrenchment: An engineering variation on refinement. In *B'98: Recent Advances in the Development and Use of the B Method, Second International B Conference, Montpellier, France, April 22-24, 1998, Proceedings*, pages 129–147, 1998.

[BP99] Richard Banach and Michael Poppleton. Retrenchment and punctured simulation. In *Integrated Formal Methods, Proceedings of the 1st International Conference on Integrated Formal Methods, IFM 99, York, UK, 28-29 June 1999*, pages 457–476, 1999.

[BPJS07] Richard Banach, Michael Poppleton, Czeslaw Jeske, and Susan Stepney. Engineering and theoretical underpinnings of retrenchment. *Sci. Comput. Program.*, 67(2-3):301–329, 2007.

[Bra06] Bastian Braun. Implementing automatic addition and verification of fault tolerance. Master's thesis, RWTH Aachen University, 2006.

[BS07] Julian Bradfield and Colin Stirling. 12 modal mu-calculi. In Johan Van Benthem Patrick Blackburn and Frank Wolter, editors, *Handbook of Modal Logic*, volume 3 of *Studies in Logic and Practical Reasoning*, pages 721 – 756. Elsevier, 2007.

[CKAA11] Pablo F. Castro, Cecilia Kilmurray, Araceli Acosta, and Nazareno Aguirre. dCTL: A branching time temporal logic for fault-tolerant system verification. In Gilles Barthe, Alberto Pardo, and Gerardo Schneider, editors, *Software Engineering and Formal Methods - 9th International Conference, SEFM 2011*, volume 7041 of *Lecture Notes in Computer Science*, pages 106–121. Springer, 2011.

[Cla99] Edmund M. Clarke. *Model checking.* MIT Press, 1999.

[CM89] K. Mani Chandy and Jayadev Misra. *Parallel program design - a foundation.* Addison-Wesley, 1989.

[CM09] Pablo F. Castro and Thomas S. E. Maibaum. Deontic logic, contrary to duty reasoning and fault tolerance. *Electr. Notes Theor. Comput. Sci.*, 258(2):17–34, 2009.

[Cri85] Flaviu Cristian. A rigorous approach to fault-tolerant programming. *IEEE Trans. Software Eng.*, 11(1):23–31, 1985.

[DCMA13a] Ramiro Demasi, Pablo F. Castro, T. S. E. Maibaum, and Nazareno Aguirre. Characterizing fault-tolerant systems by means of simulation relations. In Einar Broch Johnsen and Luigia Petre, editors, *Integrated Formal Methods, 10th International Conference, IFM 2013*, volume 7940 of *Lecture Notes in Computer Science*, pages 428–442. Springer, 2013.

[DCMA13b] Ramiro Demasi, Pablo F. Castro, T. S. E. Maibaum, and Nazareno Aguirre. Synthesizing masking fault-tolerant systems from deontic specifications. In Dang Van Hung and Mizuhito Ogawa, editors, *Automated Technology for Verification and Analysis - 11th International Symposium, ATVA 2013*, volume 8172 of *Lecture Notes in Computer Science*, pages 163–177. Springer, 2013.

[DCR+15] Ramiro Demasi, Pablo F. Castro, Nicolás Ricci, Thomas Stephen Edward Maibaum, and Nazareno Aguirre. syntmaskft: A tool for synthesizing masking fault-tolerant programs from deontic specifications. In *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, pages 188–193, 2015.

[Dij76] Edsger W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.

[EC80] E. Allen Emerson and Edmund M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In J. W. de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium, ICALP 1980*, volume 85 of *Lecture Notes in Computer Science*, pages 169–181. Springer, 1980.

[EH86] E. Allen Emerson and Joseph Y. Halpern. "sometimes" and "not never" revisited: on branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, 1986.

[EL09] Jonathan Ezekiel and Alessio Lomuscio. Combining fault injection and model checking to verify fault tolerance in multi-agent systems. In Carles Sierra, Cristiano Castelfranchi, Keith S. Decker, and Jaime Simão Sichman, editors, *8th International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2009*, pages 113–120. IFAAMAS, 2009.

[FH07] Adrian Francalanza and Matthew Hennessy. A theory for observational fault tolerance. *J. Log. Algebr. Program.*, 73(1-2):22–50, 2007.

[FMR07] Tim French, John Christopher McCabe-Dansted, and Mark Reynolds. A temporal logic of robustness. In *Frontiers of Combining Systems, 6th International Symposium, FroCoS 2007, Liverpool, UK, September 10-12, 2007, Proceedings*, pages 193–205, 2007.

[Gär99] Felix C. Gärtner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Computing Surveys*, 31, 1999.

[GH90] Gérard D. Guiho and Claude Hennebert. SACEM software validation (experience report). In François-Régis Valette, Peter A. Freeman, and Marie-Claude Gaudel, editors, *Proceedings of the 12th International Conference on Software Engineering, ICSE 1990*, pages 186–191. IEEE Computer Society, 1990.

[GJM03] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of software engineering (2. ed.)*. Prentice Hall, 2003.

[HHK95] Monika Rauch Henzinger, Thomas A. Henzinger, and Peter W. Kopke. Computing simulations on finite and infinite graphs. In *36th Annual Symposium on Foundations of Computer Science, FOCS 1995*, pages 453–462. IEEE Computer Society, 1995.

[Jan95] Tomasz Janowski. *Bisimulation and Fault-Tolerance*. PhD thesis, University of Warwick, United Kingdom, 1995.

[Jan97] Tomasz Janowski. On bisimulation, fault-monotonicity and provable fault-tolerance. In Michael Johnson, editor, *Algebraic Methodology and Software Technology, 6th International Conference, AMAST 1997*, volume 1349 of *Lecture Notes in Computer Science*, pages 292–306. Springer, 1997.

[JHAL09] Ralph D. Jeffords, Constance L. Heitmeyer, Myla Archer, and Elizabeth I. Leonard. A formal method for developing provably correct fault-tolerant systems using partial refinement and composition. In Ana Cavalcanti and Dennis Dams, editors, *Formal Methods, Second World Congress, FM 2009*, volume 5850 of *Lecture Notes in Computer Science*, pages 173–189. Springer, 2009.

[JHAL10] Ralph D. Jeffords, Constance L. Heitmeyer, Myla Archer, and Elizabeth I. Leonard. Model-based construction and verification of critical systems using composition and partial refinement. *Formal Methods in System Design*, 37(2-3):265–294, 2010.

[LA90] Peter A. Lee and Thomas Anderson. *Fault Tolerance: Principles and Practice*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2nd edition, 1990.

[Lam85] Leslie Lamport. Solved problems, unsolved problems and non-problems in concurrency. *Operating Systems Review*, 19(4):34–44, 1985.

[Lam94] Leslie Lamport. The temporal logic of actions. *ACM Trans. Program. Lang. Syst.*, 16(3):872–923, 1994.

[LSP82] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The Byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[MC80] Carver Mead and Lynn Conway. *Introduction to VLSI systems*. Addison-Wesley, 1980.

[Mil89] Robin Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989.

[MT01] Panagiotis Manolios and Richard J. Trefler. Safety and liveness in branching time. In *16th Annual IEEE Symposium on Logic in Computer Science, Boston, Massachusetts, USA, June 16-19, 2001, Proceedings*, pages 366–374, 2001.

[PR89] Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Sixteenth Annual ACM Symposium on Principles of Programming Languages, POPL 1989*, pages 179–190. ACM Press, 1989.

[PS05] Ignatius S. W. B. Prasetya and S. Doaitse Swierstra. Formal design of self-stabilizing programs. *J. High Speed Networks*, 14(1):59–83, 2005.

[SECH98] Francis Schneider, Steve M. Easterbrook, John R. Callahan, and Gerard J. Holzmann. Validating requirements for fault tolerant systems using model checking. In *3rd International Conference on Requirements Engineering, ICRE 1998*, pages 4–13. IEEE Computer Society, 1998.

[SS98] Daniel P. Siewiorek and Robert S. Swarz. *Reliable Computer Systems (3rd Ed.): Design and Evaluation*. A. K. Peters, Ltd., Natick, MA, USA, 1998.

[TP00] Wilfredo Torres-Pomales. Software fault tolerance: A tutorial. Technical report, NASA Technical Memorandum TM-2000-210616, 2000.

[YTK01] Tomoyuki Yokogawa, Tatsuhiro Tsuchiya, and Tsuchiya Kikuno. Automatic verification of fault tolerance using model checking. In *8th Pacific Rim International Symposium on Dependable Computing, PRDC 2001*, pages 95–102. IEEE Computer Society, 2001.

A. Proofs of Properties

In this section we prove some properties introduced in Section 3.

Proof of Property: $\forall i : \sigma^*[\![i+1]\!]\,\mathbf{M}\,\sigma'[i]$

The proof is by induction on i. For the base case $i = 0$, $\sigma^*[\![1]\!] = \sigma^*[0] = s\,\mathbf{M}\,s' = \sigma'[0]$. For the inductive case we have to prove that $\sigma^*[\![i+2]\!]\mathbf{M}\sigma'[i+1]$. By definition of *last*, if $\sigma^*[i+1] \neq *$, then $\sigma^*[\![i+2]\!] = \sigma^*[i+1]$. Furthermore, by definition of σ^*, $\sigma^*[i+1] = w$ for some $w \in match(\sigma[\![i+1]\!], \sigma'[i+1])$, thus by definition of *match*(), we have $\sigma^*[i+1]\,\mathbf{M}\,\sigma'[i+1]$, and since $\sigma[\![i+2]\!] = \sigma^*[i+1]$, we get $\sigma^*[\![i+2]\!]\,\mathbf{M}\,\sigma'[i+1]$.

Let us prove the case $\sigma^*[i+1] = *$. In this case, by definition of $\sigma^*[\![i+2]\!]$, we get $\sigma^*[\![i+2]\!] = \sigma^*[\![i+1]\!]$, and by induction we get: $\sigma^*[\![i+1]\!]\mathbf{M}\sigma'[i]$. By definition of σ^* we have $\sigma^*[i+1] = *$ since $\nexists w \in match(\sigma[\![i+1]\!], \sigma'[i+1])$, but taking into account that $\sigma'[i+1] \in Post(\sigma'[i])$ and $\sigma^*[\![i+1]\!]\,\mathbf{M}\,\sigma'[i]$, by $B.4$ we must have that $\sigma^*[\![i+1]\!]\,\mathbf{M}\,\sigma'[i+1]$ that is (recalling that $\sigma^*[\![i+2]\!] = \sigma^*[\![i+1]\!]$) we get $\sigma^*[\![i+2]\!]\,\mathbf{M}\,\sigma'[i+1]$.

Proof of Property: $\forall i \geq 0 : \exists j > i : \sigma^*[j] \neq *$ (Used in Lemma 3.1)

The proof is by contradiction. Suppose that $\exists i \geq 0$ such that $\forall j > i : \sigma^*[j] = *$, let k be the least of such i's, that is, $\sigma^*[k] \neq *$ and $\forall j > k : \sigma^*[j] = *$. This means that $\sigma^*[last(k+h, \sigma^*)] = \sigma^*[\![k]\!]$ for all $h \geq 0$. Furthermore, since σ' is fair and a NDF structure we have a $j > k$ such that $\sigma'[j+1] \in Post_N(\sigma'[j])$. Now, note that we have $\sigma^*[\![j+1]\!]\mathbf{M}\sigma'[j]$ (by the property above) and since $\sigma^*[k+h] = *$ (for every $h \geq 0$) we must have $\nexists w \in match(\sigma^*[\![k+h]\!], \sigma'[k+h])$, that is for $k+h = j+1$, we have $\nexists w \in match(\sigma^*[\![j+1]\!], \sigma'[j+1])$. By definition of *match*, this implies that: $\nexists w \in Post_N(\sigma^*[\![j+1]\!]) : w\,\mathbf{M}\,\sigma'[j+1]$, which contradicts $B.2$. This finishes the proof.

Now we prove some useful properties about function g as defined in Lemma 3.1 that will be useful to prove the properties used in the proof of Lemma 3.1.

Property A.1. f is monotone.

Proof. Assume $n \leq m$, that is, $m = n + h$, then:

$$\begin{aligned}
f(n) &= \#\{k \mid \sigma^*[k] \neq * \wedge 0 < k \leq n\} && \text{(Def. of } f\text{)}\\
&\leq \#\{k \mid \sigma^*[k] \neq * \wedge 0 < k \leq n+h\} && \text{(monotonicity of } \#\text{)}\\
&= f(n+h) && \text{(Def. } f\text{)}\\
&= f(m) && \text{(assumption)}
\end{aligned}$$

□

Proof of Property: $\forall i : g(i) = last(g(i+1), \sigma^*)$

Proof. The proof is by using the definition of *last*:

$$\begin{aligned}
last(g(i+1), \sigma^*) &= max\{k \mid 0 \leq k < g(i+1) : \sigma^*[k] \neq *\} && \text{(Def. of } g\text{)}\\
&= max\{k \mid 0 \leq k \leq g(i) \vee g(i) < k < g(i+1) : \sigma^*[k] \neq *\} && \text{(properties of } \leq\text{)}\\
&= max\{k \mid 0 \leq k \leq g(i) : \sigma^*[k] \neq *\} && \text{(Def. of } g\text{)}\\
&= g(i) && \text{(Def. of } max\text{)}
\end{aligned}$$

□

Proof of Property: $\forall i : \sigma^*[g(f(i))] = \sigma^*[\![i+1]\!]$.

Proof. First, let us prove that:

$$\#\{k \mid \sigma^*[k] \neq * \wedge 0 \leq k \leq k'\} = \#\{k \mid \sigma^*[k] \neq * \wedge 0 \leq k \leq last(i+1, \sigma^*)\} \Rightarrow k' \geq last(i+1, \sigma^*)$$

the proof is by contradiction, assume the antecedent holds and $k' < last(i+1, \sigma^*)$. That is: $last(i+1, \sigma^*) = k' + h$ for some h, but then:

$$\begin{aligned}
\#\{k \mid \sigma^*[k] \neq * \wedge 0 \leq k \leq k'\} &= \#\{k \mid \sigma^*[k] \neq * \wedge 0 \leq k \leq last(i+1, \sigma^*)\} && \text{(assumption)}\\
&= \#\{k \mid \sigma^*[k] \neq * \wedge 0 \leq k \leq k' \vee k' < k \leq last(i+1, \sigma^*)\} && \text{(properties of } \leq\text{)}\\
&= \#\{k \mid \sigma^*[k] \neq * \wedge 0 \leq k \leq k'\} + \{k' < k \leq last(i+1, \sigma^*) \wedge \sigma^*[k] \neq *\} && \text{(properties of } \#\text{)}
\end{aligned}$$

thus $\{k' < k \leq last(i+1, \sigma^*[k]) \wedge \sigma^*[k] \neq *\} = 0$, and then, $\sigma^*[last(i+1, \sigma^*)] = *$ which is a contradiction by definition of *last*. Now, we can prove the property:

$$
\begin{aligned}
\sigma^*[g(f(i))] &= \sigma^*[min\{k \mid f(k) = f(i)\}] && \text{(Def. of } g\text{)} \\
&= \sigma^*[min\{k \mid \#\{k \mid \sigma^*[k] \neq * \wedge 0 \leq k \leq k'\} = \#\{k \mid \sigma^*[k] \neq * \wedge 0 \leq k \leq i\}\}] && \text{(Def. of } f\text{)} \\
&= \sigma^*[min\{k \mid \#\{k \mid \sigma^*[k] \neq * \wedge 0 \leq k \leq k'\} = \#\{k \mid \sigma^*[k] \neq * \wedge 0 \leq k \leq last(i+1, \sigma^*)\}\}] && \text{(Def. of } last\text{)} \\
&= \sigma^*[last(i+1, \sigma^*)] && \text{(property above)} \\
&= \sigma^*[\![i+1]\!] && \text{(Def. of } \sigma^*[\![i+1]\!]\text{)}
\end{aligned}
$$

□