Characterizing Fault-Tolerant Systems by Means of Simulation Relations

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

[1] Department of Computing and Software,
McMaster University, Hamilton, Ontario, Canada
demasira@mcmaster.ca, tom@maibaum.org
[2] Departamento de Computación, FCEFQyN,
Universidad Nacional de Río Cuarto, Río Cuarto, Córdoba, Argentina
{pcastro,naguirre}@dc.exa.unrc.edu.ar
[3] Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Argentina

Abstract. In this paper, we study a formal characterization of fault-tolerant behaviors of systems via simulation relations. This formalization makes use of particular notions of simulation and bisimulation in order to compare the executions of a system that exhibit faults with executions where no faults occur. By employing variations of standard (bi)simulation algorithms, our characterization enables us to algorithmically check fault-tolerance, i.e., to verify that a system behaves in an acceptable way even under the occurrence of faults.

Our approach has the benefit of being simple and supporting an efficient automated treatment. We demonstrate the practical application of our formalization through some well-known case studies, which illustrate that the main ideas behind most fault-tolerance mechanisms are naturally captured in our setting.

1 Introduction

The increasing demand for highly dependable and constantly available systems has brought attention to providing strong guarantees for software correctness, especially for safety 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 deserves attention is that of capturing *faults*, understood as unexpected events that affect a system and may corrupt or degrade its performance, as well as expressing and reasoning about the properties of systems in the presence of faults.

The field of fault-tolerant systems is concerned with providing techniques that can be used to increase the fault-tolerance 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 fault-tolerance are: *component*

E.B. Johnsen and L. Petre (Eds.): IFM 2013, LNCS 7940, pp. 426–440, 2013.

replication, *N-version programming*, *exception mechanisms*, *transactions*, etc. Some emerging approaches try to deal with fault-tolerance in formal settings, with the aim of mathematically *proving* that a given system effectively tolerates faults. For example, in [9], 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 formal approach to fault-tolerance is that presented in [3–5], 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. 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, 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 a negative impact on analysis, since the performance of model checking algorithms depends on the length of the formula being analyzed.

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. The notion of fault-tolerance is characterized by defining simulation/bisimulation relations, between the desired "fault-free" program, and that which tolerates faults. Since, as it is well known, a system may tolerate faults exhibiting different degrees of so called *fault-tolerance*, different simulation/bisimulation 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*. *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; 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. Since in this approach fault-tolerance is captured via bisimulation, one is able to check that a system tolerates faults to some degree (masking, nonmasking, failsafe), without the need for user intervention, by employing (bi)simulation algorithms.

2 Preliminaries

In this section we introduce some concepts that will be necessary throughout the paper. For the sake of brevity, we assume some basic knowledge on model checking; the interested reader may consult [7]. We model fault-tolerant systems by means of *colored Kripke structures*, as introduced in [8]. Given a set of propositional letters $AP = \{p, q, s, \ldots\}$, a *colored Kripke structure* is a 5-tuple $\langle S, I, R, L, \mathcal{N}\rangle$, where S is a set of states, $I \subseteq S$ is a 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. 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*. Then, normal executions are those transiting only through green states. The set of normal executions is denoted by $\mathcal{NT}$. We assume that in every colored Kripke structure, and for every normal state, there exists at least one successor state that is also normal, and that at least one initial state is green. This guarantees that every system has at least one normal execution, i.e., that $\mathcal{NT} \neq \emptyset$.

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 adjacent wrt. R. When a trace of M starts in an initial state, it is called an *execution* of M, with *partial* executions corresponding to non-maximal sequences of adjacent states starting in an initial state. Given a trace $\sigma = s_0, s_1, s_2, s_3, \ldots$, the ith state of σ is denoted by $\sigma[i]$, and the final segment of σ starting in position i is denoted by $\sigma[i..]$. 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 $Post_N(s) = \{s \in S |\ s \to s'\}$ as the set of successors of s reachable via non-faulty (or good) transitions; similarly, $Post_F(s) = \{s \in S |\ s \dashrightarrow s'\}$ represents the set of successors of s reachable via faulty arcs. Analogously, we define $Pre_N(s')$ and $Pre_F(s')$ as the set of predecessor of s' via normal and faulty transitions, respectively. Moreover, $Post^*(s)$ denotes the states which are reachable from s. Without loss of generality, we assume that every state has a successor [7]. We denote by $\Rightarrow^*$ the transitive closure of $\dashrightarrow \cup \to$.

In order to state properties of systems, we use a fragment of dCTL [8], a branching time temporal logic with deontic operators designed for fault-tolerant system verification. Formulas in this fragment, that we call dCTL-, refer to properties of behaviors of colored Kripke structures, in which a distinction between *normal* and *abnormal* states (and therefore also a distinction between normal and abnormal traces) is made. The logic dCTL is defined over the Computation Tree Logic CTL, with its novel part being the deontic operators $\mathbf{O}(\psi)$ (obligation) and $\mathbf{P}(\psi)$ (permission), which are applied to a certain kind of path formula ψ. The intention of these operators is to capture the notion of *obligation* and *permission* over traces. Intuitively, these operators have the following meaning:

- $\mathbf{O}(\psi)$: property ψ is obliged in every future state, reachable via non-faulty transitions.
- $\mathbf{P}(\psi)$: there exists a normal execution, i.e., not involving faults, starting from the current state and along which ψ holds.

Obligation and permission will enable us to express intended properties which should hold in *all* normal behaviors and *some* normal behaviors, respectively. These deontic operators have an implicit *temporal* character, since ψ is a path formula. Let us present the syntax of dCTL-. Let $AP = \{p_0, p_1, \ldots\}$ be a set of atomic propositions. The sets Φ and Ψ of *state formulas* and *path formulas*, respectively, are mutually recursively defined as follows:

$$\Phi ::= p_i \mid \neg\Phi \mid \Phi \rightarrow \Phi \mid \mathsf{A}(\Psi) \mid \mathsf{E}(\Psi) \mid \mathbf{O}(\Psi) \mid \mathbf{P}(\Psi)$$
$$\Psi ::= \mathsf{X}\Phi \mid \Phi\ \mathcal{U}\ \Phi \mid \Phi\ \mathcal{W}\ \Phi$$

Other boolean connectives (here, state operators), such as $\wedge$, $\vee$, etc., are defined as usual. Also, traditional temporal operators G and F can be expressed, as $\mathsf{G}(\phi) \equiv \phi\ \mathcal{W}\ \bot$, and $\mathsf{F}(\phi) \equiv \top\ \mathcal{U}\ \phi$. The standard boolean operators and the CTL quantifiers A and E have the usual semantics. Now, we formally state the semantics of the logic. We start by defining the relation $\vDash$, formalizing the satisfaction of dCTL- state formulas in colored Kripke structures. For the deontic operators, the definition of $\vDash$ is as follows:

- $M, s \vDash \mathbf{O}(\psi) \Leftrightarrow$ for every $\sigma \in \mathcal{NT}$ such that $\sigma[0] = s$ we have that for every $i \geq 0$ $M, \sigma[i..] \vDash \psi$.
- $M, s \vDash \mathbf{P}(\psi) \Leftrightarrow$ for some $\sigma \in \mathcal{NT}$ such that $\sigma[0] = s$ we have that for every $i \geq 0$ $M, \sigma[i..] \vDash \psi$.

For the standard CTL operators the definition of $\vDash$ is as usual (see [7]).

We denote by $M \vDash \varphi$ the fact that $M, s \vDash \varphi$ holds for every state s of M, and by $\vDash \varphi$ the fact that $M \vDash \varphi$ for every colored Kripke structure M.

In order to illustrate the semantics of the deontic operators, let us consider the colored Kripke structure in Figure 1, where the set of propositional variables is $\{p, q, r, t\}$, and each state is labeled by the set of propositional variables that hold in it. The states that are the target of dashed arcs are abnormal states (those in which something has gone wrong); faulty states are also drawn with dashed lines, while the others represent normal configurations. Notice that the unique faulty state in this model is that named t. In this simple model, for every non-faulty execution, $p \wedge q$ is always true. In dCTL- this is expressed by the formula $\mathbf{O}(p \wedge q)$. Note that there also exist normal executions for which $p \wedge q \wedge r$ holds. This fact is expressed as $\mathbf{P}(p \wedge q \wedge r)$. Other deontic operators such as *prohibition* can be expressed by using those introduced above (see [8]).

One of the interesting characteristics of dCTL- is the possibility of distinguishing between formulas that state properties of good executions and the standard formulas, which state properties over all possible executions. For every formula φ, a formula φ^N can be built, which captures the same property as φ but restricted to good executions. This leads to the notion of *normative formula* of a given formula, and is defined as follows.

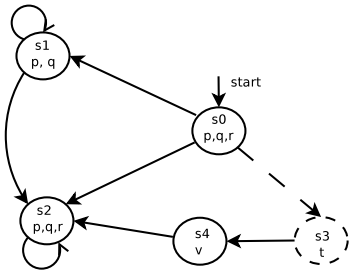

Fig. 1. A Simple Colored Kripke Structure

Definition 1. *Given a dCTL- formula φ over an alphabet AP, its* normative formula φ^N*, is defined by the following rules:*

- $(p_i)^N \stackrel{\text{def}}{=} p_i$, $(\neg\varphi)^N \stackrel{\text{def}}{=} \neg\varphi^N$, $(\varphi \wedge \varphi')^N \stackrel{\text{def}}{=} \varphi^N \wedge \varphi'^N$,
- $(\mathsf{A}(\varphi\, \mathcal{U}\, \varphi'))^N \stackrel{\text{def}}{=} \mathbf{O}(\varphi^N\, \mathcal{U}\, \varphi'^N)$, $(\mathsf{A}(\varphi\, \mathcal{W}\, \varphi'))^N \stackrel{\text{def}}{=} \mathbf{O}(\varphi^N\, \mathcal{W}\, \varphi'^N)$,
- $(\mathsf{E}(\varphi\, \mathcal{U}\, \varphi'))^N \stackrel{\text{def}}{=} \mathbf{P}(\varphi^N\, \mathcal{U}\, \varphi'^N)$, $(\mathsf{E}(\varphi\, \mathcal{W}\, \varphi'))^N \stackrel{\text{def}}{=} \mathbf{P}(\varphi^N\, \mathcal{W}\, \varphi'^N)$,
- $(\mathbf{O}(\varphi\, \mathcal{U}\, \varphi'))^N \stackrel{\text{def}}{=} \mathbf{O}(\varphi^N\, \mathcal{U}\, \varphi'^N)$, $(\mathbf{O}(\varphi\, \mathcal{W}\, \varphi'))^N \stackrel{\text{def}}{=} \mathbf{O}(\varphi^N\, \mathcal{W}\, \varphi'^N)$,
- $(\mathbf{P}(\varphi\, \mathcal{U}\, \varphi'))^N \stackrel{\text{def}}{=} \mathbf{P}(\varphi^N\, \mathcal{U}\, \varphi'^N)$, $(\mathbf{P}(\varphi\, \mathcal{W}\, \varphi'))^N \stackrel{\text{def}}{=} \mathbf{P}(\varphi^N\, \mathcal{W}\, \varphi'^N)$.

3 (Bi)Simulations and Fault-Tolerance

In this section we present a number of simulation relations that allow us to capture various kinds of fault-tolerance, namely *masking*, *nonmasking*, and *fail-safe*. In order to define these relations, we follow the basic definitions regarding simulation and bisimulation relations given in [7]. Due to space restrictions, the technical proofs of the theorems presented in this section are omitted; the interested reader can find them in [10].

We will assume that the properties of interest of a system will be safety and liveness properties (recall that any temporal specification can be written as a conjunction of safety and liveness properties [1]). Basically, in order to check fault-tolerance, we consider two colored Kripke structures of 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 purpose will be to capture, via appropriate (bi)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 sublabeling: we say that L_0 is a sublabeling 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 sublabeling allows us to focus on certain properties of models.

Let us start by introducing the notion of masking tolerance relations.

Definition 2. *(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* $\prec_{Mask} \subseteq S \times S'$ *is* masking fault-tolerant *for sublabelings* $L_0 \subseteq L$ *and* $L'_0 \subseteq L'$ *iff:*

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

We say that state s_2 is masking fault-tolerant for s_1 when $s_1 \prec_{Mask} s_2$. 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. Conditions A, $B.1$, $B.2$ and $B.3$ imply that we have a bisimulation between the normative parts of M and M'. 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 fault-tolerant for s_1.

Notice that, if there exists a self-loop at state s'_2, then we can stay forever satisfying $s_1 \prec_{Mask} s'_2$. Therefore, a fairness condition needs to be imposed on $B.4$ to ensure that masking implementations preserve liveness properties. For example, we can assume that, if a set of non-faulty transitions are enabled infinitely often, then these transitions will be executed infinitely often. We denote by $M \models_f \varphi$ the restriction of $\models$ to fair executions. 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'.

We say that M' masks faults for M iff for every initial state s_0 of M there exists an initial state s'_0 of M' such that $s_0 \prec_{Mask} s'_0$, for some masking fault-tolerant relation $\prec_{Mask}$; we denote this situation by $M \prec_{Mask} M'$. Let us not present a simple example to illustrate the above definition.

Example 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.

A potential fault in this problem would be that the 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, known as *voting*, and the ready value is written back in all the bits.

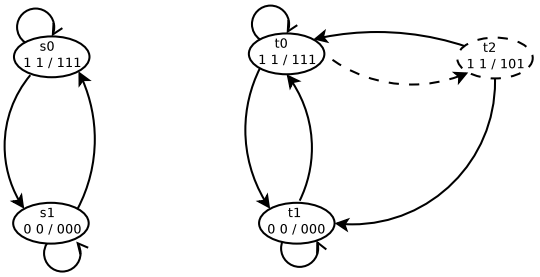

Fig. 2. Two masking fault-tolerance colored Kripke structures

In a model of 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. 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 2, 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 2. M contains only 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. It is straightforward to show that in this simple case there exists a masking fault-tolerance relation (specifically, the relation $R_1 = \{(s_0, t_0), (s_1, t_1), (s_0, t_2)\}$) between M and M' with the sublabelings L_0 and L'_0 obtained by restricting L and L' to propositions m and w, respectively.

An important property of masking fault-tolerance is that both safety and liveness properties of normative (i.e., non-faulty) executions are preserved by masking tolerant implementations under fairness restrictions.

Theorem 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, $s_1 \in S$ and $s_2 \in S'$. If $s_1 \prec_{Mask} s_2$ for sublabelings L_0 and L'_0 obtained by restricting L and L' to AP', respectively, then $M, s_1 \vDash_f \varphi^N \Rightarrow M', s_2 \vDash_f \varphi$, where all the propositional variables of φ are in AP'.*

Let us now focus on nonmasking fault-tolerance. This kind of tolerance is less strict than masking tolerance, since it allows for the existence of some states which 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 returning to a normal behavior. More technically, the normative liveness properties of the system are always preserved, whereas the normative safety properties may not be fully preserved, but must be *eventually* restated. The characterization of this kind of fault-tolerance is the following.

Definition 3. *(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 $\prec_{Nonmask} \subseteq S \times S'$ is nonmasking for sublabelings $L_0 \subseteq L$ and $L'_0 \subseteq L'$, iff:*

(A) $\forall s_1 \in I : (\exists s_2 \in I' : s_1 \prec_{Nonmask} s_2)$ *and* $\forall s_2 \in I' : (\exists s_1 \in I : s_1 \prec_{Nonmask} s_2)$.

(B) for all $s_1 \prec_{Nonmask} s_2$ *the following holds:*

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

(2) if $s'_1 \in Post_N(s_1)$, *then there exists* $s'_2 \in Post_N(s_2)$ *with* $s'_1 \prec_{Nonmask} s'_2$.

(3) if $s'_2 \in Post_N(s_2)$, *then there exists* $s'_1 \in Post_N(s_1)$ *with* $s'_1 \prec_{Nonmask} s'_2$.

(4) if $s'_2 \in Post_F(s_2)$, *then there exists* $s'_1 \in Post_N(s_1)$ *with* $s'_1 \prec_{Nonmask} s'_2$, *or*

(5) if $s'_2 \in Post_F(s_2)$ *with* $s'_1 \not\prec_{Nonmask} s'_2$ *for all* $s'_1 \in Post_N(s_1)$, *then there exists a finite path fragment* $s_2 \dashrightarrow s'_2 \Rightarrow^* s''_2$ *such that either* $s'_1 \prec_{Nonmask} s''_2$ *for some* $s'_1 \in Post_N(s_1)$, *or* $s_1 \prec_{Nonmask} s'_2$.

Let us explain this definition. Conditions A, $B.1$, $B.2$, $B.3$, $B.4$ are similar to the conditions of Def. 2. Condition $B.5$ asserts that if $s_1 \prec_{Nonmask} s_2$ and every "faulty" successor state s'_2 of s_2 is not in a nonmasking relation with any normal successor of s_1, then there exists a path fragment that leads from s_2 to s''_2 such that $s'_1 \prec_{Nonmask} s''_2$ for some normal successor state s'_1 of s_1, or s'_2 is nonmasking fault-tolerant for s_1.

We say that M' is nonmasking fault-tolerant wrt. M iff for every initial state s_0 of M there exists an initial state s'_0 of M' such that $s_0 \prec_{Nonmask} s'_0$, for some nonmasking fault-tolerance $\prec_{Nonmask}$(indicated by $M \prec_{Nonmask} M'$).

At first sight, nonmasking fault-tolerance seems similar to the notion of weak bisimulation used in process algebra [2], where silent steps are taken into account. Notice however that, as opposed to weak bisimulation where silent steps produce only nonobservable (i.e., internal) changes, faults may produce observable changes in a nonmasking fault-tolerance relation. Let us present an example of nonmasking tolerance.

Example 2. For the memory cell introduced in Example 1, consider now the colored Kripke structures M (left) and M' (right) depicted in Figure 3. Now we consider that two faults may occur: up to two bits may lose its charge before any normal transition is taken. The relation $R_2 = \{(s_0, t_0), (s_1, t_1), (s_0, t_2)\}$ is nonmasking tolerant for (M, M') and the sublabelings L_0 and L'_0, obtained by restricting L and L' to propositions m and w, respectively.

An important property is that if s_2 is nonmasking fault-tolerant for s_1 and for every state of normal paths starting in s_1, φ holds, then in fair executions starting in s_2, φ eventually holds even in the presence of faults.

Theorem 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,* $s_1 \in S$ *and* $s_2 \in S'$. *If* $s_1 \prec_{Nonmask} s_2$ *for sublabelings* L_0 *and* L'_0 *obtained by restricting* L *and* L' *to* AP', *respectively, then* $M, s_1 \models_f \varphi^N \Rightarrow M', s_2 \models_f \mathsf{AFAG}(\varphi)$, *where all the propositional variables of* φ *are in* AP'.

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.

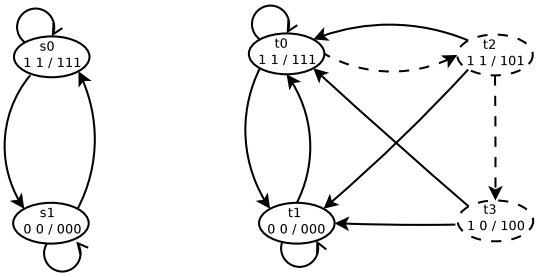

Fig. 3. Two nonmasking fault-tolerance colored Kripke structures

Definition 4. *(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* $\prec_{Failsafe} \subseteq S \times S'$ *is failsafe for sublabelings* $L_0 \subseteq L$ *and* $L'_0 \subseteq L'$ *iff:*

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

Whenever two states s_1 and s_2 are related by a failsafe fault-tolerant relation $\prec_{Failsafe}$, i.e., $s_1 \prec_{Failsafe} s_2$, we say that s_2 is failsafe fault-tolerant for s_1. We say that M' is failsafe fault-tolerant for M iff for every initial state s_0 of M there exists an initial state s'_0 of M' such that $s_0 \prec_{Failsafe} s'_0$, for some failsafe fault-tolerant relation $\prec_{Failsafe}$; we denote this situation by $M \prec_{Failsafe} M'$.

Let us explain the above definition. Conditions A, $B.1$, $B.2$, and $B.3$ are similar to those of Def. 2, regarding masking fault-tolerance. Condition $B.4$ states that if $s_1 \prec_{Failsafe} s_2$, then every outgoing faulty transition from s_2 either must be matched to an outgoing normal transition from s_1, requiring states s'_1 and s'_2 to be labeled with the same propositions, or s'_2 must be failsafe fault-tolerant for s_1. We now present a simple example to illustrate this notion.

Example 3. Consider the colored Kripke structures M (left) and M' (right) depicted in Figure 4. M is the specification of the expected ideal, fault-free, behavior. M', on the other hand, involves the occurrence of one fault. The relation $R_3 = \{(s_0, t_0), (s_1, t_1)\}$ is a failsafe fault-tolerance relation for (M, M') and the sublabelings that are obtained by restricting L and L' to propositions m and w.

Our definition of failsafe fault-tolerance preserves safety properties.

Theorem 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,* $s_1 \in S$ *and* $s_2 \in S'$*. If* $s_1 \prec_{Failsafe} s_2$ *for sublabelings* L_0

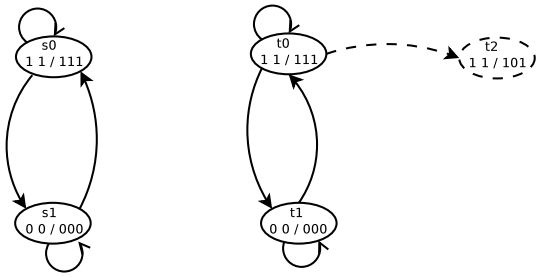

Fig. 4. Two failsafe fault-tolerance colored Kripke structures

and L'_0 obtained by restricting L and L' to AP', respectively, and φ is a safety property, then $M, s_1 \vDash \varphi^N \Rightarrow M', s_2 \vDash \varphi$, where all the propositional variables of φ are in AP'.

This property says that if we have a failsafe relation between s_1 and s_2 and for every state in normal paths starting in s_1, φ holds in the absence of faults, then φ is always true even in the presence of faults in paths starting in s_2.

The following lemma provides important information regarding all the fault-tolerance relations defined above.

Lemma 1. *Given relations $\prec_{Mask}$, $\prec_{Nonmask}$ and $\prec_{Failsafe}$, we have the following properties:*

- *$\prec_{Mask}$, $\prec_{Nonmask}$, $\prec_{Failsafe}$ are transitive,*
- *If M does not have faults, then: $M \sqsubset M' \Rightarrow M' \sqsubset M$, where $\sqsubset \in \{\prec_{Mask}, \prec_{Nonmask}, \prec_{Failsafe}\}$*
- *$\prec_{Mask}$, $\prec_{Nonmask}$ and $\prec_{Failsafe}$ are not necessarily reflexive.*

We also have properties of these relations corresponding to *inclusions*:

Theorem 4. *Let Mask, NMask and FSafe be the sets of masking, nonmasking and failsafe relations between two colored Kripke structures M and M'. Then we have:*

$$\mathsf{Mask} \subseteq \mathsf{FSafe} \text{ and } \mathsf{Mask} \subseteq \mathsf{NMask}$$

3.1 Checking Fault-Tolerance Properties

Simulation and bisimulation relations are amenable to efficient computational treatment. For instance, in [7, 11] algorithms for calculating several simulation and bisimulation relations are described and proved to be polynomial with respect to the number of states and transitions of the corresponding models. We have adapted these algorithms to our setting, thus obtaining efficient procedures to prove masking, nonmasking and failsafe fault-tolerance. Such algorithms can be used to verify whether $M \sqsubset M'$, with $\sqsubset \in \{\prec_{Mask}, \prec_{Nonmask}, \prec_{Failsafe}\}$. We discuss the algorithm for computing masking fault-tolerance. The algorithms

Algorithm 1. Computation of masking fault-tolerant

Input: colored Kripke structure M
Output: masking fault-tolerant $\prec_{Mask}$

1: **for all** $s_2 \in \mathcal{F}$ **do**
2: $Mask(s_2) := \{s_1 \in \mathcal{N} \mid L_0(s_1) = L_0(s_2)\}$
3: $Remove(s_2) := \mathcal{N} \backslash Pre_N(Mask(s_2))$
4: **end for**
5: **while** $\exists\ s_2' \in \mathcal{F}$ 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: **for all** $s_2 \in Pre_N(s_2')$ **do**
9: **if** $s_1 \in Mask(s_2)$ **then**
10: $Mask(s_2) := Mask(s_2) \backslash \{s_1\}$
11: **for all** $s \in Pre_N(s_1)$ with $Post_N(s) \cap Mask(s_2) = \emptyset$ **do**
12: $Remove(s_2) := Remove(s_2) \cup \{s\}$
13: **end for**
14: **end if**
15: **end for**
16: **end for**
17: $Remove(s_2') := \emptyset$
18: **end while**
19: **for all** $s_2 \in \mathcal{F}$ **do**
20: **if** $Post^*(s_2) \neq Post^*(Mask(s_2))$ **then**
21: $Mask(s_2) := \emptyset$
22: **end if**
23: **end for**
24: **return** $\{(s_1, s_2) \mid s_1 \in Mask(s_2)\}$

for the other relations can be obtained in a similar way; the interested reader is referred to [10]. The basic scheme for checking masking fault-tolerance is sketched in Algorithm 1. This algorithm takes as input a colored Kripke structure $M = \langle S, I, R, L, \mathcal{N} \rangle$ and a sublabeling $L_0 \subseteq L$, and produces a masking fault-tolerance relation $\prec_{Mask}$. In order to check fault-tolerance properties, we take two colored Kripke structures M and M' over AP (the system specification and the fault-tolerant implementation), and combine them in a single structure $M \oplus M'$ via disjoint union, to feed as input to the algorithm. Notice that Algorithm 1 only deals with the case of faulty states/transitions (condition $B.4$ of Def. 2), since a standard bisimulation algorithm can be used for checking that the normative behavior described in the specification is bisimilar with that exhibited by the fault-tolerant implementation (conditions $B.2$ and $B.3$). This algorithm is an adaptation of the similarity-checking algorithm for finite graphs defined in [11]. We have made slight changes (between lines 1 and 18) in order to explore all the faulty transitions $s_2 \dashrightarrow s_2'$ and look for those normal transitions $s_1 \rightarrow s_1'$ which mask faulty ones.

Let us briefly explain Algorithm 1. Consider $\mathcal{F} = S \backslash \mathcal{N}$ to be the set of faulty states in M. For each $s_2 \in \mathcal{F}$, the set $Mask(s_2)$ contains the normal states that

are candidates for masking s_2. Initially, $Mask(s_2)$ 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 outermost loop of lines 5–18 is $Remove(s_2') = \emptyset$ for all $s_2' \in \mathcal{F}$, in which case there are no normal states that need to be removed from the sets of simulators $Mask(s_2)$ for $s_2 \in Pre_N(s_2')$. Within the while-loop body, the main idea is to pick one pair (s_1, s_2') with $s_1 \in Remove(s_2')$ per iteration; for each one we scan through the predecessor list of s_2' and test for each normal state $s_2 \in Pre_N(s_2')$ to see whether $s_1 \in Mask(s_2)$. In the positive case, s_1 is removed from $Mask(s_2)$. Subsequently, we add to the set $Remove(s_2)$ all normal predecessors s of s_1 such that $Post_N(s) \cap Mask(s_2) = \emptyset$. The last for-loop checks whether from any faulty state s_2 which is masked by a normal state, we can reach a normal state to recover to the normal behavior. Therefore, for each faulty state s_2, we inspect the existence of the reachable normal states from s_2 (line 20). In the case that the set of successors of s_2 is not equal to the set of the normal successors of the normal state which mask s_2, then we remove all states from $Mask(s_2)$. Finally, the masking fault-tolerance $\prec_{Mask}$ is obtained from a colored Kripke structure M by performing the union between the set obtained from a bisimulation algorithm used to check that the system strongly bisimulates the specification for the normative part, and the set returned by Algorithm 1.

Regarding the complexity of the algorithm for checking masking fault-tolerance, it is polynomial. More precisely, the time complexity of the bisimulation quotient algorithm [7] is $\mathcal{O}(|\mathcal{N}| \cdot |AP'| + E \cdot \log |\mathcal{N}|)$, where E is the number of edges in M. On the other hand, Algorithm 1 can be computed in time $\mathcal{O}(E \cdot |\mathcal{F}| + |\mathcal{F}| \cdot AP' + |\mathcal{F}|)$. Hence, the masking fault-tolerance of a colored Kripke structure $M = \langle S, I, R, L, \mathcal{N} \rangle$ for sublabeling $L_0 \subseteq L$ obtained by restricting AP to AP', can be computed in a running time of $\mathcal{O}(E \cdot \log |\mathcal{N}| + E \cdot |F|)$.

4 An Example: The Muller C-element

The Muller C-element [14] 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. In [3], 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 circuit involving three inputs, where an extra input u in the circuit is added. Then, the

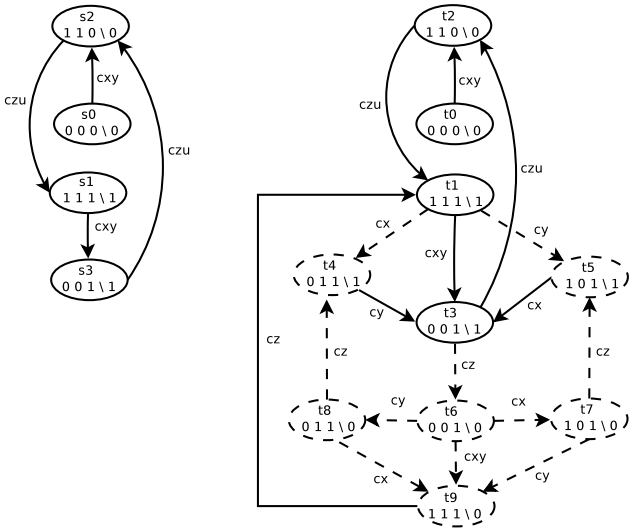

Fig. 5. A nonmasking fault-tolerance for the Muller C-element with a majority circuit

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. Figure 5 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 $z = 0$. Transitions are labeled by 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. 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. The relation $R_{c-element} = \{(s_0, t_0), (s_1, t_1), (s_2, t_2), (s_3, t_3)\}$ is a nonmasking fault-tolerance for (M, M') and the sublabelings obtained by restricting the original labelings to letters u, x, y, z. Therefore, when the majority circuit behaves correctly, this implementation masks delays of inputs.

We have developed other case studies illustrating the practical application of our framework, based on well known fault-tolerance models, including, e.g., Byzantine agreement. These can be found in [10].

5 Related Work

Our work is most closely related to formal approaches to fault-tolerance. One of these is that presented in [4], where the problem of multitolerance is addressed. In order to do so, the authors define the concepts of masking, nonmasking and failsafe tolerances using liveness and safety specifications in a linear-time framework. In our approach, we focus on branching time properties of programs. In our opinion, branching time is important for fault-tolerance specification. This view is also shared by Attie, Arora, and Emerson in [6], where an algorithm for synthesizing fault-tolerant programs from CTL specifications is presented. They consider CTL as the temporal logic specification for the input of their synthesis method. Instead of using CTL, we use a branching time temporal logic that has a convenient mechanism for stating fault-tolerance properties, via the use of deontic operators. We believe that our formalism is better suited for capturing fault-tolerance properties. Finally, we can mention the works presented in [12, 13], where various notions of bisimulation are investigated with the aim of capturing fault-tolerant properties, in the context of process algebras. An obvious difference wrt. 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, which is known to be less expressive than temporal logics. Also, the notions of masking, nonmasking and failsafe fault-tolerance are not investigated in the referenced works.

6 Conclusions and Future Work

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 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.

As future work, we are exploring the extension of our setting with *synthesis*, so that fault-tolerant programs may be automatically constructed from the system specification, a description of the faults and their consequences, and a desired degree of fault-tolerance. Synthesis of programs has been extensively investigated

in the context of linear time logic [15], while in our case it is necessary to deal with a branching time formalism. This is important since, as argued in [6], some important properties related to fault-tolerance require branching time operators. It is also in [6] where a framework for synthesis of programs from branching time specifications is introduced. We believe that some of the work presented in this paper can be used to extend that introduced in [6], to automatically synthesize programs that mask faults, recover from error situations or stay in safe states. Finally, we also plan to extend our framework to accommodate multitolerance [4], in which multiple classes of faults may occur simultaneously.

Acknowledgements. The authors would like to thank the anonymous referees for their helpful comments. This work was partially supported by a Fellowship from IBM Canada, in support of the Automotive Partnership Canada funded project NECSIS; by the Argentinian Agency for Scientific and Technological Promotion (ANPCyT), through grants PICT PAE 2007 No. 2772, PICT 2010 No. 1690 and PICT 2010 No. 2611; and by the MEALS project (EU FP7 programme, grant agreement No. 295261).

References

1. Alpern, B., Schneider, F.: Defining Liveness. Inf. Process. Lett. 21(4) (1985)
2. Milner, R.: Communication and Concurrency. PHI Series in Computer Science. Prentice-Hall (1989)
3. Arora, A., Gouda, M.: Closure and Convergence: A Foundation of Fault-Tolerant Computing. IEEE Trans. Soft. Eng. 19(11) (1993)
4. Arora, A., Kulkarni, S.: Component Based Design of Multitolerant Systems. IEEE Trans. Software Eng. 24(1) (1998)
5. Arora, A., Kulkarni, S.: Detectors and Correctors: A Theory of Fault-Tolerance Components. In: Proc. of ICDCS (1998)
6. Attie, P., Arora, A., Emerson, A.: Synthesis of fault-tolerant concurrent programs. ACM Trans. Program. Lang. Syst. 26(1) (2004)
7. Baier, C., Katoen, J.-P.: Principles of Model Checking. MIT Press (2008)
8. Castro, P.F., Kilmurray, C., Acosta, A., Aguirre, N.: dCTL: A Branching Time Temporal Logic for Fault-Tolerant System Verification. In: Barthe, G., Pardo, A., Schneider, G. (eds.) SEFM 2011. LNCS, vol. 7041, pp. 106–121. Springer, Heidelberg (2011)
9. Cristian, F.: A rigorous approach to fault-tolerant programming. IEEE Trans. Software Eng. (1985)
10. Demasi, R., Castro, P., Maibaum, T., Aguirrer, N.: Characterizing Fault-Tolerant Systems by Means of Simulation Relations, Tech. Report, http://www.cas.mcmaster.ca/~demasira/reportSimFTS.pdf
11. Henzinger, M., Henzinger, T., Kopke, P.: Computing Simulations on Finite and Infinite Graphs. In: Proc. of FOCS (1995)
12. Janowski, T.: Bisimulation and Fault-Tolerance. PhD thesis (1995)
13. Janowski, T.: On Bisimulation, Fault-Monotonicity and Provable Fault-Tolerance. In: Proc. of *AMAST* (1997)
14. Mead, C., Conway, L.: Introduction to VLSI systems. Addison-Wesley (1980)
15. Pnueli, A., Rosner, R.: On the Synthesis of a Reactive Module. In: Proc. of POPL (1989)